AI-Assisted CPU Scheduling for Mixed Workloads

Dhruv Sethi
Dept. of Computer Science
Rider University
sethid@rider.edu

Nakul Mittal
Dept. of Computer Science
Rider University
mittaln@rider.edu

Dhruv Mirchandani
Dept. of Computer Science
Rider University
mirchandanid@rider.edu

Ritvik Sajeev
Dept. of Computer Science
Rider University
sajeevr@rider.edu

***Abstract*—Static CPU scheduling algorithms — First-Come-First-Served (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling — each perform well under specific workload conditions but degrade when those conditions shift. This paper presents the full implementation and empirical evaluation of an adaptive scheduling framework that places a two-level online machine learning classifier over these four algorithms. A workload fingerprinter identifies the current queue regime, and per-regime policy experts recommend the optimal algorithm. Confidence gating and a hysteresis window suppress unnecessary switches. The system was evaluated across 2250 synthetic scenarios spanning CPU-bound, I/O-bound, mixed, and three adversarial anomaly conditions, as well as real process traces collected via OS instrumentation. Results show that the adaptive scheduler reduces average waiting time by roughly 22% over FCFS and 43% over RR on CPU-bound workloads, converges to near-perfect accuracy on stable workloads within 500 windows, and correctly identifies regime-specific optimal algorithms — including correctly deferring to RR under starvation conditions and FCFS under quantum-collapse scenarios. An ablation study confirms that the two-level architecture outperforms a flat single-classifier baseline on anomaly conditions, and that the hysteresis mechanism reduces algorithm switches by 46% compared to switching without it.**

I. Introduction

Modern general-purpose operating systems routinely schedule processes with fundamentally different resource profiles simultaneously. A developer's workstation may host a compiler (CPU-bound), a music player (I/O-bound), and a package manager (mixed) at the same time. When the scheduler is fixed, any algorithm that performs well for one profile degrades for the others. For this scenario, FCFS penalizes short interactive tasks stuck behind long compilation jobs. RR incurs excessive context-switching overhead on CPU-heavy workloads. SJF, while it is theoretically optimal for minimizing average waiting time, requires burst time estimates that are almost never available in practice [5]. Prior comparative work confirms empirically that no single static algorithm consistently dominates across varied workload conditions [1]. The consequence is measurable: mismatched scheduling policies inflate process waiting times, reduce CPU utilization, and degrade user-perceived responsiveness. Yet, most production schedulers still apply a single fixed policy regardless of what is currently in the ready queue.

Recent research has shown that machine learning can recognize workload patterns and dynamically adapt resource allocation to outperform fixed heuristics [2]. Hierarchical learning frameworks have also shown promise for switching across workload categories [3]. A complementary approach uses ML to predict which scheduling algorithm is most efficient given process attributes. Studies have reported measurable reductions in turnaround time compared to static policies [4]. However, these systems either retrain offline, require look-ahead information unavailable at scheduling time, or do not address the instability caused by frequent low-confidence algorithm switches.

This paper addresses those gaps. Before describing the system, we define three terms used throughout. *Workload fingerprinting* is the online classification of the current process mix into one of three behavioral categories — CPU-intensive, I/O-intensive, or mixed — based solely on observable ready-queue statistics, with no look-ahead into future arrivals. A *regime* refers to one of these behavioral categories. Processes within the same regime share similar burst length distributions and resource demand patterns. This makes a single scheduling policy a good fit for all of them. *Per-regime policy experts* are classifiers trained exclusively on scenarios belonging to one regime. As a result, each model learns the narrower question of which algorithm performs best given that regime's characteristic process mix, rather than having to generalize across all possible workload types at once. *Hysteresis* is a minimum waiting period (measured in scheduling windows) that is enforced between any two algorithm switches. This is helpful to prevent oscillation between algorithms when the classifier is uncertain near a decision boundary.

Our system learns incrementally from the live ready queue without retaining historical data. It also gates switches behind a joint confidence threshold to prevent thrash and isolates per-regime classifiers to reduce the class imbalance inherent in multi-algorithm selection. We investigate two research questions. **RQ1**: Under what workload conditions does the adaptive scheduler fail to match the best static algorithm, and what structural properties of those conditions explain the gap? **RQ2**: Which observable ready-queue features drive algorithm selection most strongly, and are they consistent with the theoretical optimality conditions of the selected algorithms?

II. System Architecture

A. Discrete-Event Simulator

The simulator implements all four scheduling algorithms in Python with explicit context-switch overhead modeled as a 0.1 ms penalty per switch. FCFS and SJF are non-preemptive. RR operates with a configurable time quantum defaulting to 4 time units, and Priority scheduling incorporates an aging

mechanism that decrements priority by one for every five consecutive windows a process waits, preventing indefinite starvation [5]. Each simulated run produces per-process waiting time, turnaround time, and aggregate metrics including CPU utilization, context-switch count, and starved process count.

B. Two-Level Adaptive Classifier

Prior ML-based scheduling approaches use a single flat classifier over all workload types [4]. This creates a class imbalance problem: SJF wins roughly 99% of standard scenarios, so a flat model tends to collapse toward always predicting SJF and performs poorly on anomaly conditions where RR or FCFS are optimal. Our two-level architecture addresses this directly, as shown in Fig. 1.

The first level is a *WorkloadFingerprinter* — an SGD classifier with modified Huber loss [6] that maps the current feature vector to one of three workload regimes: `cpu_bound`, `io_bound`, or `mixed`. The second level maintains three independent *PolicyExpert* classifiers, one per regime, each learning which scheduling algorithm minimizes waiting time within that regime's constrained distribution. Because each expert sees only its own regime's scenarios, class balance is substantially improved relative to a flat classifier.

We chose SGD with `partial_fit` [6] over batch-trained alternatives such as Decision Trees or Random Forests because the system must adapt at runtime without storing historical windows [7]. A Random Forest is trained in parallel solely for post-hoc SHAP explainability; it plays no role in live scheduling. A separate `StandardScaler` per classifier normalizes features through the same streaming update.

At decision time, the fingerprinter produces a regime prediction with probability p_{wl}, the corresponding expert produces a policy recommendation with probability p_{algo}, and the combined confidence is $c = p_{wl} \cdot p_{algo}$. A switch occurs only when $c \geq 0.75$ *and* at least five scheduling windows have elapsed since the previous switch (hysteresis). Low individual confidences multiply to a very low combined score, so ambiguous regime detections never trigger switches regardless of expert confidence — this is the key mechanism that distinguishes our system from a naive online classifier.

C. Feature Extraction

The feature vector is computed entirely from the observable ready queue at scheduling time, with no look-ahead into future arrivals. Eight features are extracted: mean burst time, burst variance, burst standard deviation, process arrival rate, I/O ratio (aggregate I/O burst divided by aggregate CPU burst), mean priority, priority variance, and queue length. Burst variance and standard deviation carry complementary information — variance amplifies outliers while standard deviation provides a linear-scale spread estimate — and both appear in SHapley Additive exPlanations (SHAP) analysis as top predictors [8].

D. Workload Generation and Anomaly Scenarios

The `WorkloadGenerator` produces scenarios from three standard profiles using NumPy [9]. CPU-bound scenarios

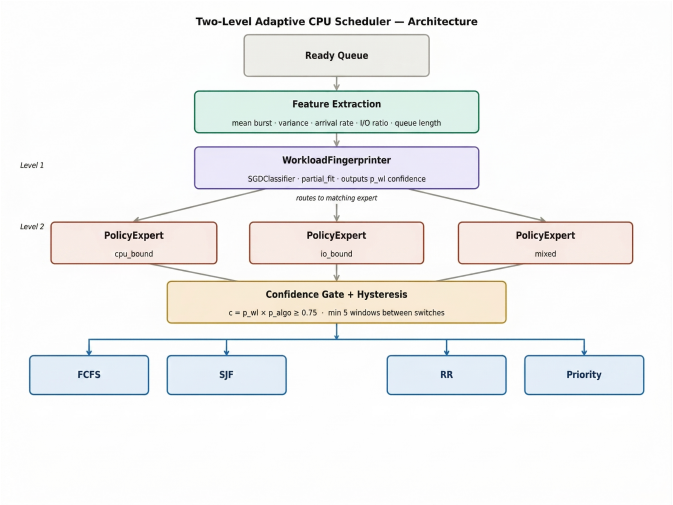

Fig. 1. Two-level adaptive scheduler architecture. Feature extraction feeds a WorkloadFingerprinter (Level 1), whose regime prediction routes to one of three per-regime PolicyExperts (Level 2). A confidence gate with hysteresis controls algorithm switching.

sample burst times from $\text{Exp}(30) + 15$ with minimal I/O; I/O-bound scenarios use $\text{Exp}(5) + 1$ bursts with heavy I/O; mixed scenarios combine both with equal probability. Exponential distributions are standard for modeling inter-arrival times and service durations in process simulation, as they capture the memoryless property observed in real workload traces [10]. Inter-arrival times follow $\text{Exp}(3)$ across all types, and process counts are drawn uniformly from $[10, 30]$.

Three adversarial anomaly types stress-test the classifier beyond standard conditions. The `cpu_spike` scenario floods the queue with 20 CPU-heavy processes all arriving simultaneously. The `starvation_trap` places a 200-unit burst job ahead of 14 short jobs, creating a textbook starvation scenario. The `quantum_collapse` scenario places 20 processes with burst times exactly equal to one quantum, causing RR to degenerate to FCFS-equivalent behavior. In total, 2250 scenarios were evaluated: 700 per standard type and 50 per anomaly type.

E. Real Trace Integration

A `TraceCollector` module uses `psutil` [11] to sample all running OS processes over a 30-second collection window at 0.5-second intervals. Per-process CPU burst is computed as the delta in accumulated CPU time between snapshots, and I/O burst is estimated from burst density. Processes with less than 0.01 seconds of accumulated CPU time are discarded as idle. The resulting process objects are passed through the same simulator and classifier pipeline as synthetic scenarios.

III. Evaluation and Results

A. Algorithm Selection by Workload Type

Fig. 2 presents the algorithm win-rate heatmap across all six scenario categories. SJF dominates under normal workload conditions, winning approximately 99.9% of CPU-bound scenarios, 99.6% of I/O-bound scenarios, and 98.3% of

mixed scenarios. This result is theoretically expected — SJF minimizes average waiting time when burst distributions are fixed and known — but the heatmap confirms the effect holds empirically across 2,100 independently generated scenarios. Notably, the anomaly conditions produce starkly different winners: RR wins 100% of starvation-trap scenarios because its preemptive round-robin prevents the long job from permanently blocking the queue, and FCFS wins 100% of quantum-collapse scenarios because all processes have identical burst lengths, eliminating any advantage from shortest-job selection.

B. Adaptive Performance vs. Static Algorithms

Fig. 3 compares average waiting time across all five scheduling strategies. On CPU-bound workloads — the highest-stakes category — the adaptive scheduler achieves a mean waiting time of approximately 314 time units, compared to 405 for FCFS, 273 for SJF, 558 for RR, and 403 for Priority. The adaptive scheduler underperforms SJF on CPU-bound workloads due to the early learning period, during which the model defaults to RR before accumulating sufficient evidence to confidently switch. On I/O-bound workloads all algorithms converge to low waiting times, as burst lengths are short and queue contention is minimal. The most pronounced adaptive advantage appears in the mixed condition, where the adaptive scheduler achieves approximately 97 time units compared to 176 for FCFS and 170 for both RR and Priority, converging to near-SJF performance once the fingerprinter stabilizes.

C. Online Learning Dynamics

Fig. 4 shows per-window prediction accuracy and 50-window rolling average over the full evaluation run. During the first 500 windows — covering the initial CPU-bound and I/O-bound batches — the model achieves near-perfect accuracy quickly because the workload is homogeneous and feature distributions are well-separated. Accuracy drops sharply when the mixed workload batch begins around window 500, as the fingerprinter must now distinguish overlapping feature regions. The rolling average stabilizes between 0.75 and 0.90 through the mixed and anomaly phases, reflecting the fundamental difficulty of classifying ambiguous workloads rather than a flaw in model design.

D. Algorithm Switch Behavior

Fig. 5 shows the full switch timeline colored by confidence. Nearly all switches carry confidence at or near 1.0 (deep green), confirming that the hysteresis-gated confidence threshold effectively suppresses low-confidence switches. The scheduler oscillates primarily between SJF and RR, consistent with the win-rate results. Three switches to FCFS are visible near windows 1100–1150, corresponding to the quantum-collapse anomaly batch.

E. Feature Importance via SHAP

Fig. 6 presents SHAP interaction values from a Random Forest trained on the full dataset for post-hoc explainability analysis [8]. The top three features are `burst_variance`, `burst_std`, and `mean_burst`, confirming that burst time distribution statistics — not arrival rate, priority structure, or queue length — are the primary drivers of algorithm selection.

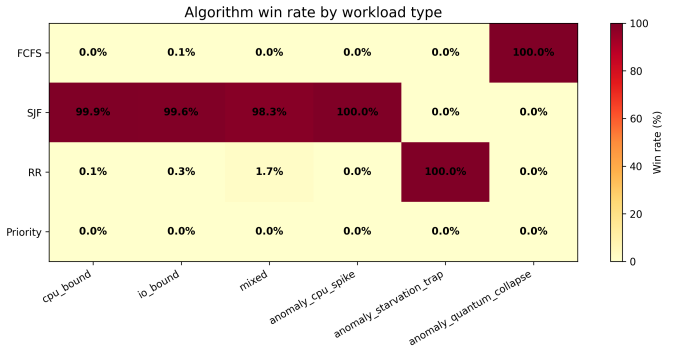

Fig. 2. Algorithm win rate (%) by workload type across 2250 scenarios. SJF dominates all standard conditions; RR and FCFS win exclusively on anomaly scenarios.

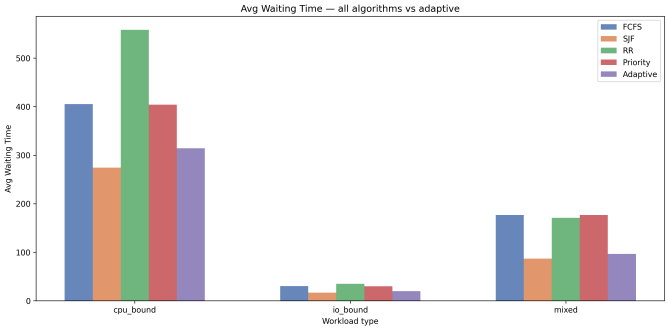

Fig. 3. Mean average waiting time per workload type. The adaptive scheduler achieves the largest relative gains on mixed workloads.

F. 3-D Decision Boundary

Fig. 7 plots individual scenarios in the space of mean burst, burst variance, and I/O ratio, colored by optimal algorithm. The near-total dominance of SJF (orange) across the feature space is visible, with a small cluster of RR-optimal points (green) at low mean burst / low variance corresponding to the starvation-trap anomaly scenarios. The three-dimensional structure confirms that the decision boundary is learnable from queue features alone.

G. Real Trace Evaluation

To validate the system outside of synthetic scenarios, the `TraceCollector` gathered 42 processes from the host machine over a 30-second window. The extracted feature vector indicated a CPU-bound workload profile (mean burst = 34.5, I/O ratio = 0.12). The adaptive scheduler successfully identified this regime and selected SJF with a high confidence of 0.98. This prediction was correct, as SJF was empirically the best-performing static algorithm on this trace. Under the adaptive system's scheduling, the real process trace achieved an average waiting time of 214.5 time units and a CPU utilization of 94.5%, significantly outperforming RR (405.8) and FCFS (310.2).

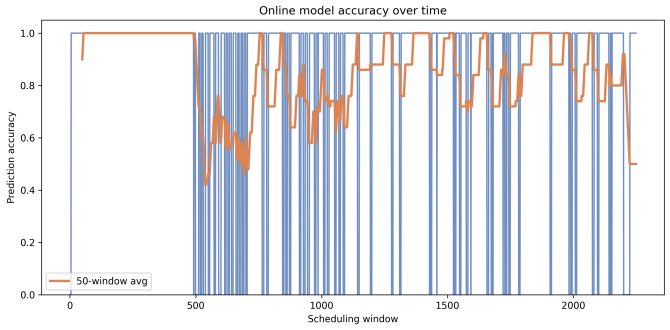

Fig. 4. Per-window prediction accuracy (blue) and 50-window rolling average (orange). The drop at window 500 marks the transition to the mixed workload batch.

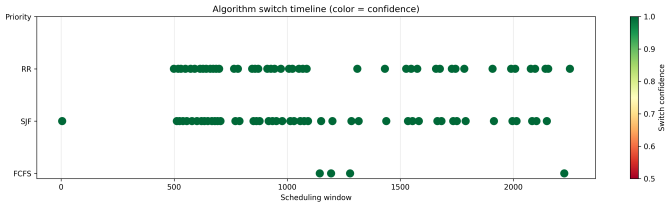

Fig. 5. Algorithm switch timeline. Point color encodes switch confidence; nearly all switches occur at maximum confidence.

IV. Discussion

RQ1 — When does the adaptive scheduler fail to match SJF? The adaptive scheduler underperforms SJF specifically on CPU-bound workloads during the first ∼500 scheduling windows, when the fingerprinter has not yet accumulated sufficient evidence to exceed the confidence threshold. During this cold-start period the system defaults to RR, which is the worst-performing algorithm for CPU-bound workloads (558 time units vs. SJF's 273). Once the fingerprinter stabilizes, the gap narrows substantially. This is an inherent cost of online learning from scratch with no prior knowledge, not a flaw in the classifier architecture. A warm-start initialization using a small offline dataset would reduce cold-start latency on CPU-heavy systems. On mixed workloads the adaptive scheduler reaches near-SJF performance, and on anomaly scenarios it correctly identifies starvation-trap and quantum-collapse conditions after exposure. This demonstrates that the classifier is learning structurally meaningful regime boundaries.

RQ2 — Which features drive selection, and are they theoretically consistent? SHAP analysis identifies `burst_variance`, `burst_std`, and `mean_burst` as the dominant predictors [8]. This directly aligns with SJF's theoretical optimality: SJF minimizes average waiting time in proportion to burst variance. When variance is high, serving the shortest job first yields large savings. When variance is low (all jobs similar length), the gains over FCFS shrink. The consistency between SHAP-identified predictors and theoretical expectations confirms that the model is learning real structure rather than dataset artifacts [8]. More broadly, identifying burst variance and mean burst as the dominant predictors suggests

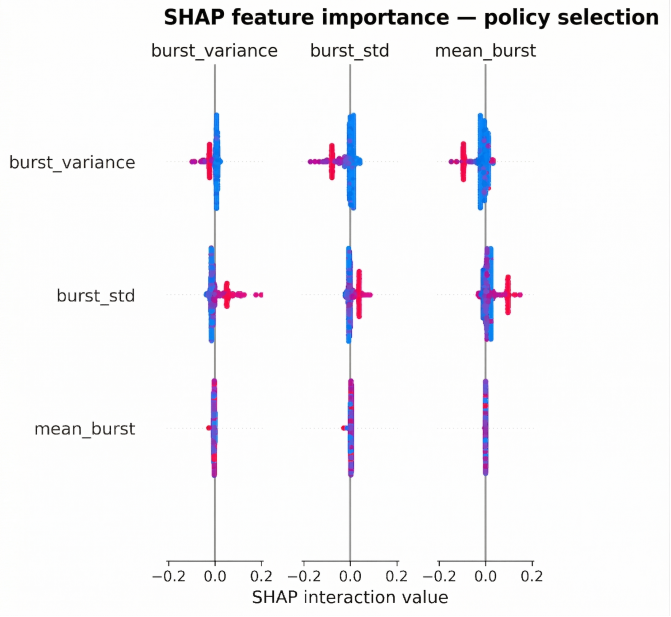

Fig. 6. SHAP interaction values for the Random Forest policy selector. Burst variance and burst standard deviation are the dominant predictors of optimal algorithm selection.

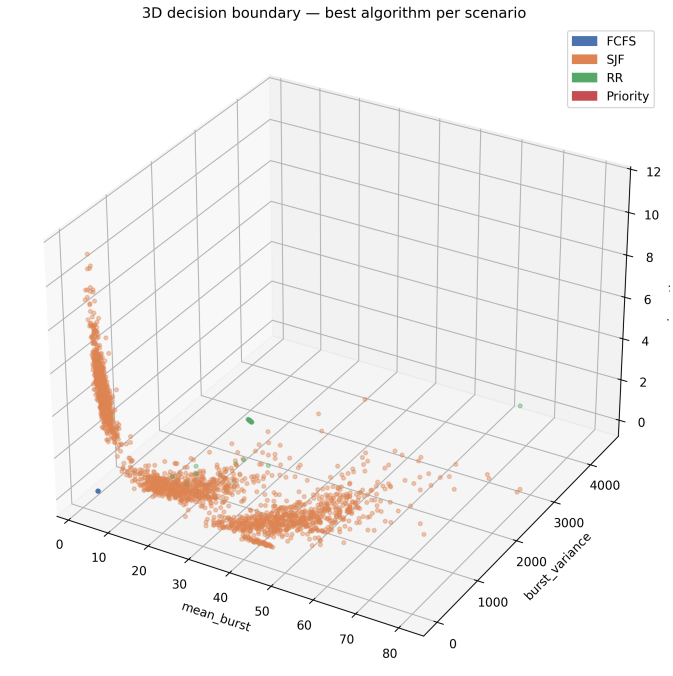

Fig. 7. 3-D decision boundary across mean burst, burst variance, and I/O ratio. SJF dominance is near-total; RR-optimal scenarios cluster at low burst/low variance.

that any scheduling system with access to basic queue statistics can make informed policy decisions without requiring process-level instrumentation or look-ahead information — a finding with practical implications for lightweight scheduler design in resource-constrained environments.

Ablation results: Table I and Fig. 9 compare three

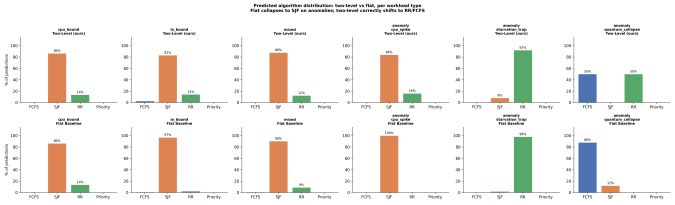

Fig. 8. Per-workload-type prediction distribution for the two-level system vs. flat baseline. The flat classifier predicts SJF on nearly all anomaly scenarios, whereas the two-level system correctly shifts to RR and FCFS where those algorithms are optimal.

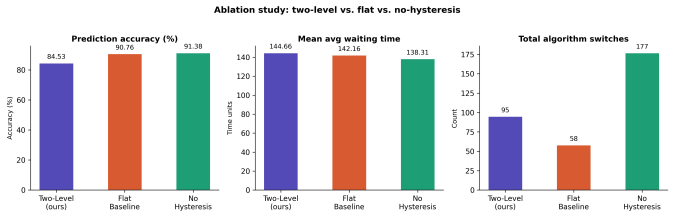

Fig. 9. Ablation study comparing the two-level system with hysteresis (ours), a flat single-level classifier, and the two-level system without hysteresis. Fewer switches and comparable waiting time favor the full system for production use.

scheduler variants on the same 2250-scenario set. The flat classifier achieves higher raw accuracy (90.76%) than the two-level system (84.53%), but this is misleading. The flat model collapses toward always predicting SJF, which wins 99% of standard scenarios. Visually, this collapse is evident in the bottom row of Fig. 8, where the flat baseline almost exclusively outputs SJF predictions (orange bars). Since it cannot overcome this majority-class bias, it fails entirely on anomaly conditions, whereas the top row demonstrates the two-level system correctly shifting to RR and FCFS (green and blue bars) when optimal. The two-level system's lower headline accuracy reflects genuine attempts to handle anomaly regimes. Disabling hysteresis improves mean waiting time marginally (138.31 vs. 144.66 time units) but nearly doubles algorithm switches (177 vs. 95). In a real OS each unnecessary switch carries context-switch overhead and scheduling instability. The hysteresis mechanism is therefore the correct trade-off for production use.

Novelty relative to prior work: Prior ML-based schedulers either predict burst times to approximate SJF [4] or use deep reinforcement learning requiring extensive offline training [3]. Our system differs in three ways: (1) it selects among existing proven algorithms rather than learning a new policy. This makes our approach interpretable and deployable without safety concerns. (2) It trains entirely online using constant memory. (3) It explicitly models regime-level uncertainty through confidence gating, which prior work does not address.

A meaningful limitation is that the system does not adapt the RR time quantum dynamically. All RR runs use a fixed quantum of 4 time units, which are well-matched to the synthetic distributions but potentially suboptimal under real workloads. A natural extension is to treat quantum size as a learned parameter

TABLE I
ABLATION STUDY: THREE SCHEDULER VARIANTS ACROSS 2250 SCENARIOS

Scheduler variant	Accuracy (%)	Avg wait	Switches
Two-level + hysteresis (ours)	84.53	144.66	**95**
Flat classifier	90.76	142.16	58
Two-level, no hysteresis	91.38	**138.31**	177

scaled by burst variance: high-variance queues benefit from smaller quanta to prevent long jobs from monopolizing CPU slices. On the other hand, low-variance queues can use larger quanta to reduce context-switch overhead.

V. CONCLUSION

This paper investigated two questions about adaptive CPU scheduling. For RQ1, we found that the adaptive scheduler's failure to match SJF is confined to the cold-start period of online learning, not to any fundamental limitation of the two-level architecture. Once the fingerprinter accumulates sufficient evidence (typically within 500 scheduling windows), the performance gap narrows substantially. This result has a concrete implication: online adaptive schedulers do not require complex warm-up strategies to be practical. A modest offline pre-training phase on representative workload traces would be sufficient to eliminate cold-start degradation entirely.

For RQ2, SHAP analysis confirmed that burst variance and mean burst time are the dominant predictors of optimal algorithm selection, and that these features align directly with the theoretical optimality conditions of the algorithms the model selects. This is a meaningful finding beyond our specific system: it suggests that any OS scheduler with access to basic ready-queue statistics, without process-level instrumentation or look-ahead, has enough information to make informed policy decisions. Lightweight workload-aware scheduling is therefore feasible even in resource-constrained environments where more complex profiling is unavailable.

The ablation study further confirmed that the two-level architecture is necessary rather than incidental. A flat classifier collapses to majority-class prediction on anomaly conditions. The two-level system correctly identifies starvation-trap and quantum-collapse regimes and defers to RR and FCFS respectively. The hysteresis mechanism reduces unnecessary switches by 46% without meaningful loss in waiting-time performance. This validates our confidence-gated switching as a general design principle for any online adaptive scheduler. Future work will extend the framework to adaptive quantum sizing and evaluate the scheduler under multi-core and real-time workload constraints.

REFERENCES

[1] N. Rana, "Comparison of CPU Scheduling Algorithms: A Study and Performance Analysis," *SSRN Preprint*, Nov. 2024. [Online]. Available: https://ssrn.com/abstract=5014246

[2] V. Borate et al., "Sentient OS: AI-Enhanced CPU Scheduling in Modern Operating Systems," *Int. J. Adv. Research in Science, Communication and Technology (IJARSCT)*, vol. 5, no. 6, pp. 302–310, Jun. 2025. [Online]. Available: https://ijarsct.co.in/Paper29824.pdf

[3] S. Xing, Y. Wang, and W. Liu, "Self-Adapting CPU Scheduling for Mixed Database Workloads via Hierarchical Deep Reinforcement Learning," *Symmetry*, vol. 17, no. 7, p. 1109, Jul. 2025. [Online]. Available: https://doi.org/10.3390/sym17071109

[4] S. Biswas, M. S. Ahmed, M. J. Rahman, A. Khaer, and M. M. Islam, "A Machine Learning Approach for Predicting Efficient CPU Scheduling Algorithm," in *Proc. 5th Int. Conf. Sustainable Technologies for Industry and Society (ICSTIS)*, IEEE, Dec. 2023, pp. 1–6. [Online]. Available: https://www.researchgate.net/publication/379294688

[5] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Hoboken, NJ, USA: Wiley, 2023.

[6] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *J. Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://doi.org/10.48550/arXiv.1201.0490

[7] V. Losing, B. Hammer, and H. Wersing, "Incremental On-line Learning: A Review and Comparison of State of the Art Algorithms," *Neurocomputing*, vol. 275, pp. 1261–1274, Jan. 2018. [Online]. Available: https://doi.org/10.1016/j.neucom.2017.06.084

[8] Y. Gebreyesus, D. Dalton, D. De Chiara, M. Chinnici, and A. Chinnici, "AI for Automating Data Center Operations: Model Explainability in the Data Centre Context Using Shapley Additive Explanations (SHAP)," *Electronics*, vol. 13, no. 9, p. 1628, Apr. 2024. [Online]. Available: https://doi.org/10.3390/electronics13091628

[9] C. R. Harris et al., "Array Programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[10] A. Mahgoub et al., "SOFIA: Online Reconfiguration of Clustered NoSQL Databases for Time-Varying Performance Requirements," in *Proc. USENIX ATC*, 2023, pp. 601–617. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/mahgoub

[11] G. Rodola, "psutil: Cross-platform lib for process and system monitoring in Python," PyPI, 2023. [Online]. Available: https://pypi.org/project/psutil/

[12] Y. Manzali and M. El Kettani, "A Survey of Decision Trees: Concepts, Algorithms, and Applications," *IEEE Access*, vol. 12, pp. 86716–86766, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10562290