AVIOS — AI-Augmented Linux-like Scheduler

A reproducible research prototype for ML-driven scheduling decisions

Author:

Dhanushree K

(Solo Project)

Date:

September 2025

Abstract

This project presents AVIOS, an AI-augmented Linux-like scheduler designed to enhance operating system scheduling decisions using machine learning–based task classification. Tasks are categorized along four dimensions — resource type, interactivity, execution length, and priority — and mapped into a composite score that influences scheduler assignment, quantum scaling, and CFS vruntime adjustment. The system was implemented as a reproducible simulation framework with Linux-like semantics, incorporating FIFO, RR, and CFS policies. Across diverse workloads, AVIOS achieved 17–36% improvements in average turnaround time, with up to 81% reductions in median turnaround for real-time tasks, while maintaining throughput, fairness, and CPU utilization comparable to the baseline Linux scheduler. This demonstrates the potential of ML-driven scheduling to reduce latency for short and interactive tasks without sacrificing system efficiency.

Introduction

Operating system (OS) scheduling is a critical function that determines how processes share CPU resources. The efficiency of the scheduler directly impacts system performance, task responsiveness, and overall user experience. In modern computing environments, workloads are highly diverse — ranging from compute-heavy batch jobs and I/O-intensive tasks to interactive applications and real-time processes. Designing a scheduler that can balance responsiveness for short, latency-sensitive tasks with fairness and throughput for longrunning tasks remains a fundamental challenge.

Traditional Linux schedulers such as the Completely Fair Scheduler (CFS) and Round Robin (RR) rely on static heuristics like nice values, fixed quanta, and vruntime balancing. While effective, these rules often fail to adapt dynamically to the wide range of workloads observed in real-world systems. For example, interactive tasks may experience degraded responsiveness when competing with CPU-bound or batch processes, while real-time workloads demand deterministic guarantees.

This project explores how machine learning (ML) can augment OS scheduling by providing an intelligent, data-driven layer on top of Linux-like policies. By classifying tasks based on their resource type, interactivity, execution length, and priority, the scheduler can make adaptive decisions about:

- Which scheduling class (FIFO, RR, or CFS) a task should be assigned to,
- How its CFS vruntime should be scaled to favor responsiveness, and
- How time quanta for RR and CFS should be adjusted based on task scores.

The objective of this work is to design, implement, and evaluate an AI-augmented Linux-like scheduler simulator (AVIOS) that reduces turnaround and response times for latencysensitive tasks, while maintaining throughput, fairness, and CPU utilization similar to the baseline Linux scheduler. Through extensive experiments on diverse workloads — CPUbound, I/O-bound, batch, interactive, stress, real-time, and realistic mixed traces — AVIOS demonstrates consistent improvements in responsiveness without compromising fairness.

Background

Operating systems use schedulers to manage CPU allocation among multiple competing tasks. The Linux kernel supports several well-known scheduling policies:

- **First-In First-Out (FIFO):** A real-time scheduling policy where tasks are executed in the order they arrive, without preemption, until they block or complete.
- **Round Robin (RR):** A preemptive policy where each task gets a fixed quantum in cyclic order, ensuring fairness but often at the cost of responsiveness for short tasks.
- **Completely Fair Scheduler (CFS):** The default Linux scheduler for normal tasks, which assigns CPU time proportionally to task weights (based on *nice* values). CFS tracks each task's *vruntime* (virtual runtime) to approximate "fair" CPU sharing.

Limitations of traditional scheduling:

While these policies are robust and widely adopted, they face challenges in mixed and dynamic workloads:

- Short, interactive tasks often suffer from high latency if queued behind long-running CPU-bound tasks.
- Static time quanta in RR or CFS may not adapt well to workload variability.
- Real-time tasks require deterministic guarantees that heuristics alone struggle to ensure under contention.

Prior approaches:

Historically, schedulers have relied on heuristics like priority levels, aging, and I/O-aware tuning. More recently, research has explored applying machine learning (ML) to scheduling — predicting task behavior, classifying workloads, or dynamically tuning scheduler parameters. However, most such approaches remain experimental or limited to narrow domains.

This project, AVIOS, builds upon this idea by directly integrating ML-driven task classification and scoring into a Linux-like scheduler simulator, showing measurable benefits across multiple workloads.

System Architecture

The **AVIOS architecture** consists of modular components connected in a high-level flow:

Trace → Classifier → Score → Scheduler Decision → Metrics

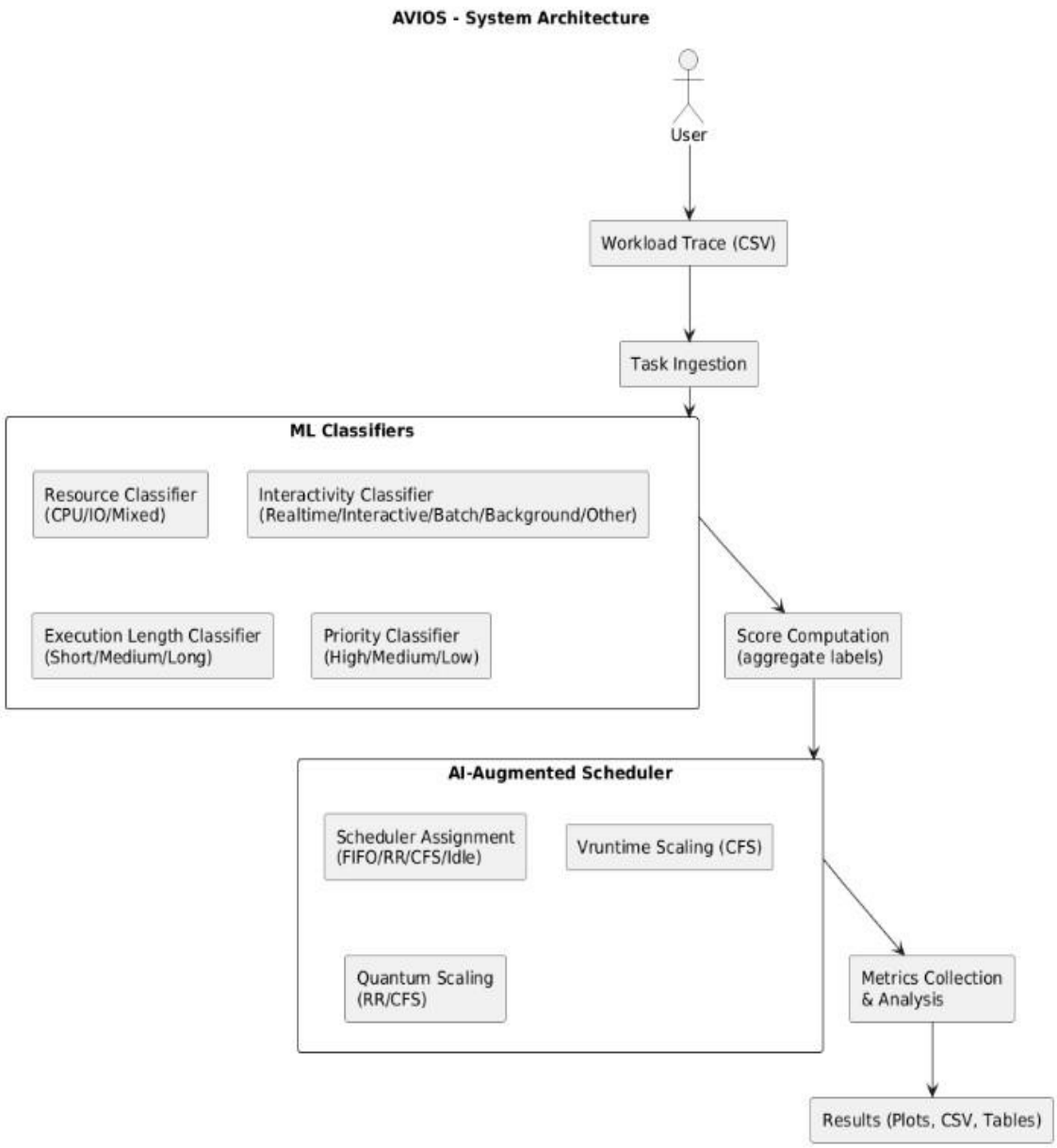
Modules Explained

1. **Task Ingestion**
    - Input traces (CSV logs collected from workloads) are read.
    - Each task includes features such as arrival time, CPU/IO usage, context switches, and state.
2. **Classification**
    - Pre-trained ML models classify each task into four categories:
        - **Resource type:** CPU / IO / Mixed
        - **Interactivity:** Real-time / Interactive / Background / Batch /Other
        - **Execution length:** Short / Medium / Long
        - **Priority:** High / Medium / Low
    - Encoders and feature JSON files ensure reproducible preprocessing.
3. **Score Calculation**
    - Each task's classification labels are combined into a **composite score**.
    - This score acts as a compact signal of task urgency and sensitivity.
4. **Scheduler Assignment & Time-Slice Scaling**
    - Based on the score and labels:
        - Real-time → **FIFO**
        - Interactive / Short → **RR**
        - Others → **CFS**
    - **CFS vruntime scaling:** Higher scores reduce vruntime growth, favoring responsive tasks.

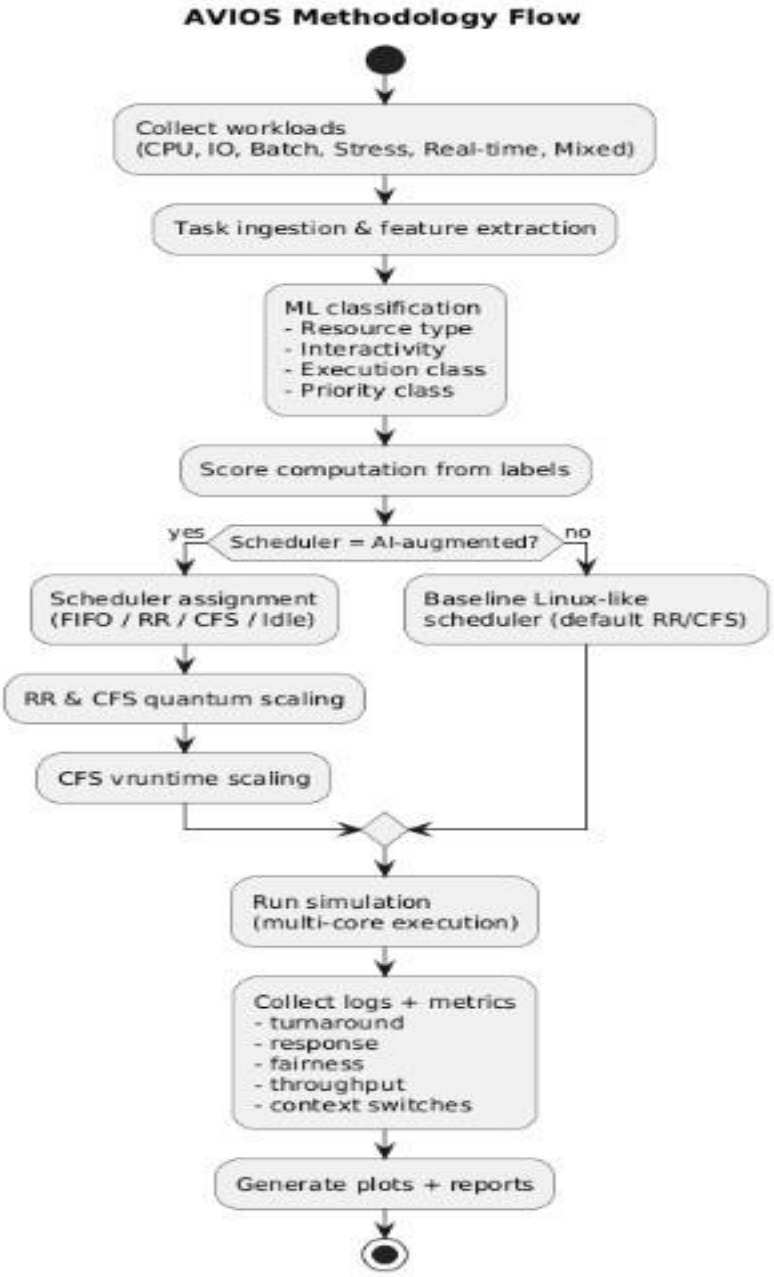
- **RR & CFS quantum scaling:** Higher scores and longer execution class grant slightly longer quanta, avoiding starvation.

5. **Metrics Collection**

    - During simulation, logs capture per-task and aggregate metrics:
        - Turnaround time
        - Response time
        - Context switches
        - CPU utilization
        - Fairness (Jain's index)
        - Throughput
    - Results are saved as CSVs and visualized in plots for workload-by-workload analysis.



AVIOS ARCHITECTURE DIAGRAM



AVIOS METHODOLOGY FLOW DIAGRAM

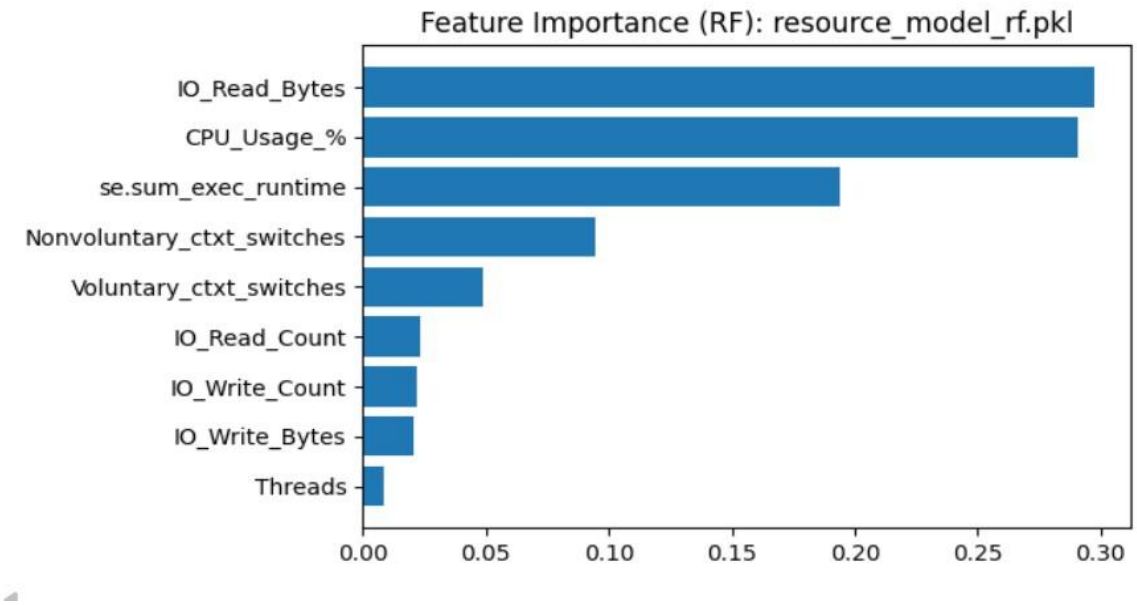
Model Evaluation

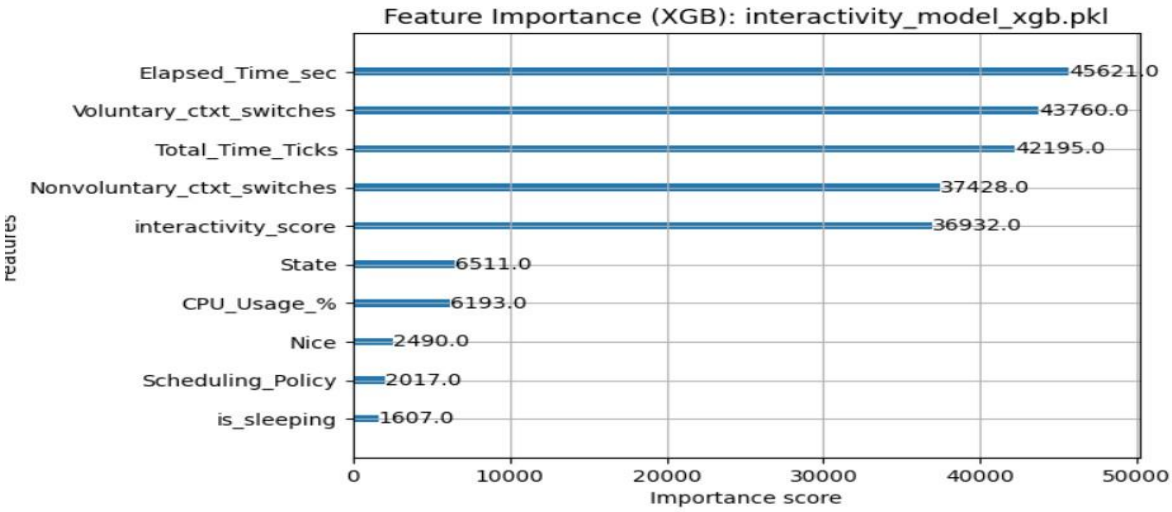
The machine learning classifiers used in AVIOS achieve consistently high accuracy, confirming the reliability of the task-labeling stage. Across four classifiers (Resource, Interactivity, Execution Length, Priority), we observed test-set accuracies of **>95%.** Validated using 5-fold cross-validation. This indicates strong generalization and stable performance across different task distributions.

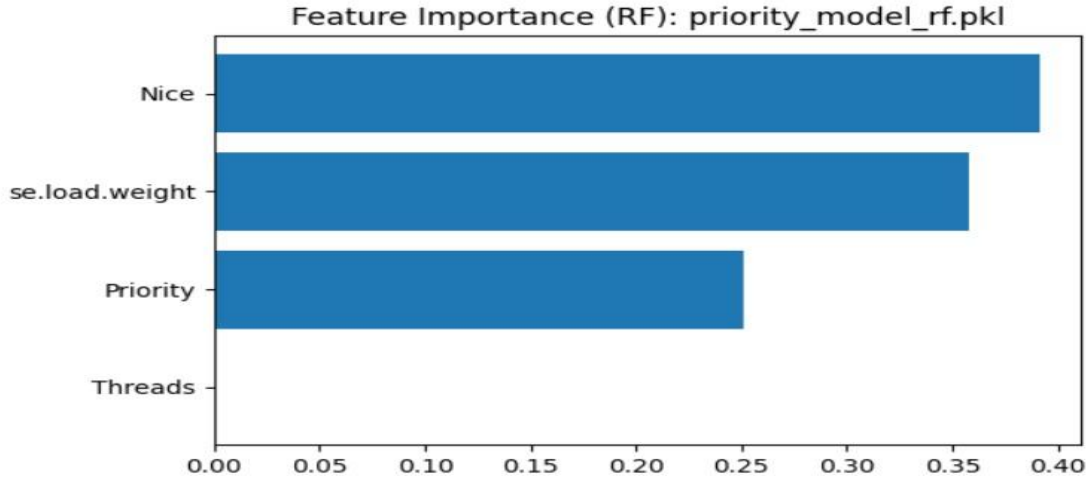
Feature importance analysis highlights that features such as CPU usage, I/O byte counts, nice value, vruntime, number of context switches, and memory footprint were the strongest predictors, aligning well with domain expectations of process behavior.

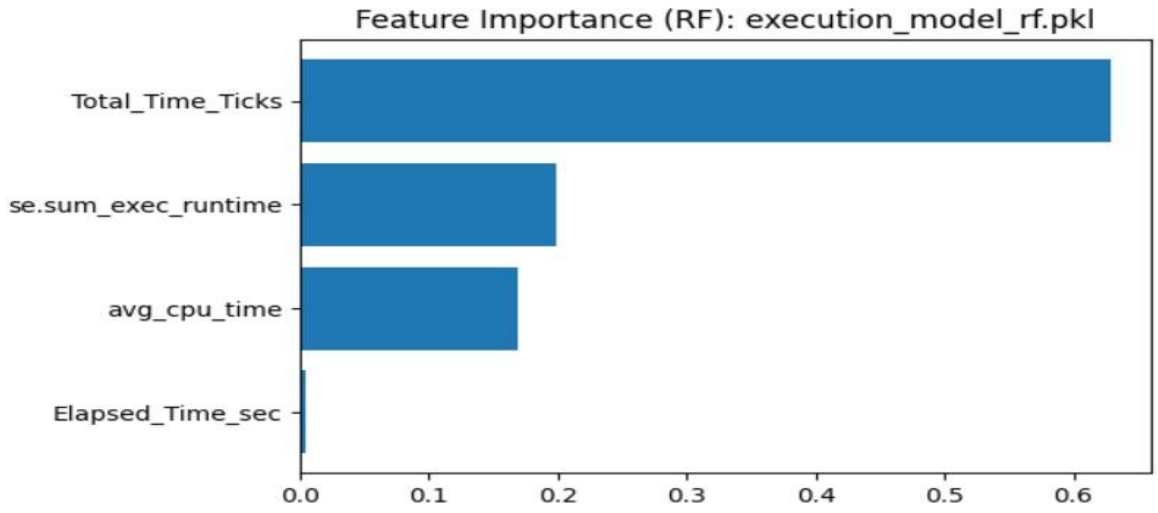
These results confirm that the ML backbone of AVIOS is both accurate and interpretable, ensuring scheduling decisions are grounded in meaningful signals rather than arbitrary heuristics.

Feature Importance




Feature Importance (XGB): interactivity_model_xgb.pkl
Features
Elapsed_Time_sec
Voluntary_ctxt_switches
Total_Time_Ticks
Nonvoluntary_ctxt_switches
interactivity_score
State
CPU_Usage_%
Nice
Scheduling_Policy
is_sleeping
45621.0
43760.0
42195.0
37428.0
36932.0
6511.0
6193.0
2490.0
2017.0
1607.0
0
10000
20000
30000
40000
50000
Importance score


Feature Importance (RF): priority_model_rf.pkl
Nice
se.load.weight
Priority
Threads
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40


Feature Importance (RF): execution_model_rf.pkl
Total_Time_Ticks
se.sum_exec_runtime
avg_cpu_time
Elapsed_Time_sec
0.0
0.1
0.2
0.3
0.4
0.5
0.6

Implementation

The system was implemented as a **Linux-like scheduler simulator** with two variants:

- **Baseline (Linux-like)**: Implements standard multi-core scheduling with **FIFO, RoundRobin (RR), and Completely Fair Scheduler (CFS)** semantics, closely modeled on the Linux design.
- **AI-Augmented**: Extends the baseline with **task classification** (via pre-trained ML models) and a **score-driven mechanism** that influences:
    - **Scheduler assignment** (deciding whether a task is routed to FIFO, RR, or CFS).
    - **CFS vruntime scaling** (high-score tasks accumulate vruntime more slowly, gaining earlier access to CPU).
    - **RR and CFS quantum scaling** (long tasks with higher scores receive slightly larger slices).

Code Modules (src/scheduler/*)

- **linux_baseline.py**: Baseline scheduler implementation with standard queueing logic.
- **ai_scheduler.py**: AI-augmented scheduler, integrates classification and score-based policies.
- **ai_simulator.py / linux_simulator.py**: Simulation drivers that ingest workload traces, run the scheduler, and log outputs.
- **task.py**: Task abstraction and feature vector construction.
- **metrics.py**: Metrics calculation (turnaround, response, throughput, fairness, utilization).
- **data_models.py**: Handles model/encoder/feature JSON loading.
- **utils.py**: Helper functions for trace parsing and logging.

Workload Generators (.sh scripts in src/workloads/)

Custom **bash workload scripts** (cpu.sh, io.sh, batch.sh, interactive.sh, real_time.sh, stress.sh, mixed_realistic.sh) were created to generate traces. These simulate task arrival patterns by

launching combinations of **CPU stressors, I/O ops, background jobs, real-time tasks, and realistic user activity**.

ML Models (src/models/)

Four separate **classification models** were trained offline using **scikit-learn (Random Forest)** and **XGBoost**:

- Resource type (CPU-bound / IO-bound / Mixed)
- Interactivity class (Realtime / Interactive / Batch / Background / Other)
- Execution length class (Short / Medium / Long)
- Priority class (High / Medium / Low)

Encoders and feature JSONs are stored in the repo. Models are automatically loaded by the simulator at task admission.

Experimental Setup

- **Datasets**:
    - **Synthetic workloads** (CPU-heavy, IO-heavy, Batch, Stress, Real-time).
    - **Mixed realistic workload**, collected from actual interactive activities (editing files, Python scripting, YouTube scrolling, video calls, downloads, and background processes).
- **Scale**:
    - Each workload trace contained between **~900 and 1,400 tasks**.
    - The **mixed realistic workload** contained **1,227 tasks**.
- **Simulation**:
    - The simulator replays workloads as **arrival + execution traces**.
    - **Time Units in Simulation:** All scheduling decisions and metrics in AVIOS are expressed in **ticks**. A tick corresponds to one Linux clock tick (SC_CLK_TCK, typically 1000 Hz), which maps to **~1 ms wall time**. Thus, turnaround/response times shown in plots (ticks) can be approximately converted to seconds by dividing by 1000. In collector traces, both CPU ticks and elapsed wall-seconds are logged, ensuring consistent mapping across data collection and simulation.
    - Both baseline and AI-augmented schedulers run under identical total tick budgets to ensure fairness of comparison.
- **Tools & Environment**:
    - **Language**: Python 3.12 ○ **Libraries**:
        - **pandas** (data manipulation & metrics aggregation)
        - **numpy** (numerical ops)
        - **scikit-learn** (classification models)
        - **xgboost** (gradient-boosted models for interactivity classification)
        - **matplotlib** (visualization of results)
        - **joblib** (model persistence/loading)

- **System**: All simulations were executed in a controlled Linux-like environment with reproducible random seeds.

Results

The system was evaluated across six workloads: **CPU-bound, IO-bound, Batch, Stress, Realtime, and Mixed Realistic**. For each workload, we compared the Linux-like baseline scheduler and the AI-augmented scheduler on aggregate metrics:

- **Turnaround time (avg, median)**
- **Response time (avg, 95th percentile)**
- **Fairness (Jain's Index)**
- **CPU utilization**
- **Context switches**
- **Throughput**

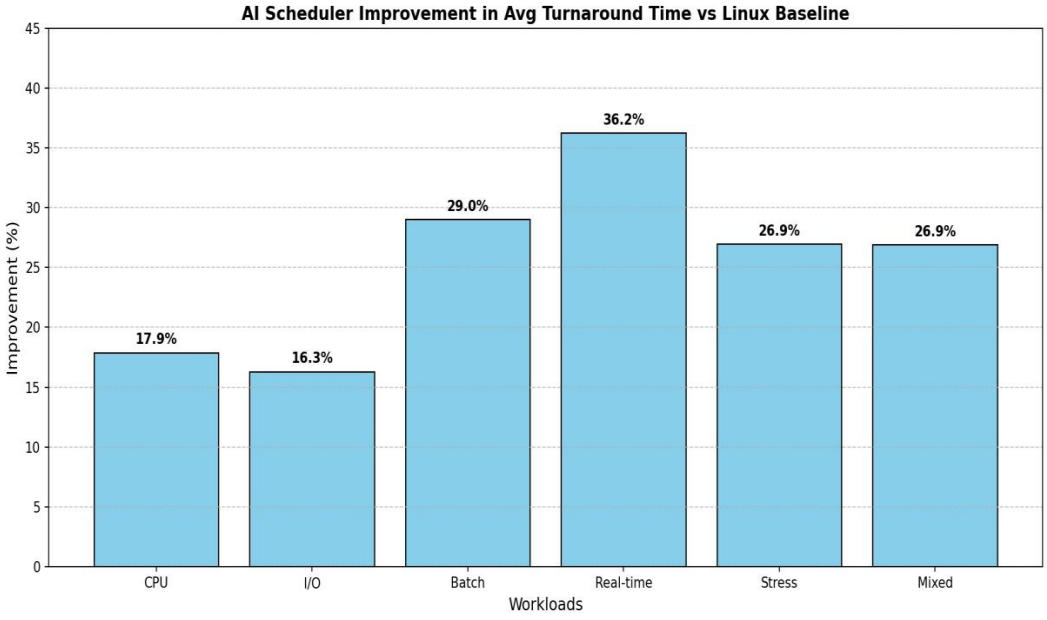
Aggregate Highlights

- **Average turnaround** improved by **17–36%** across workloads.
- **Median turnaround** improved substantially, with up to **81% gains** in real-time workloads.
- **Average response time** followed a similar trend, with **consistent 17–36% reductions**.

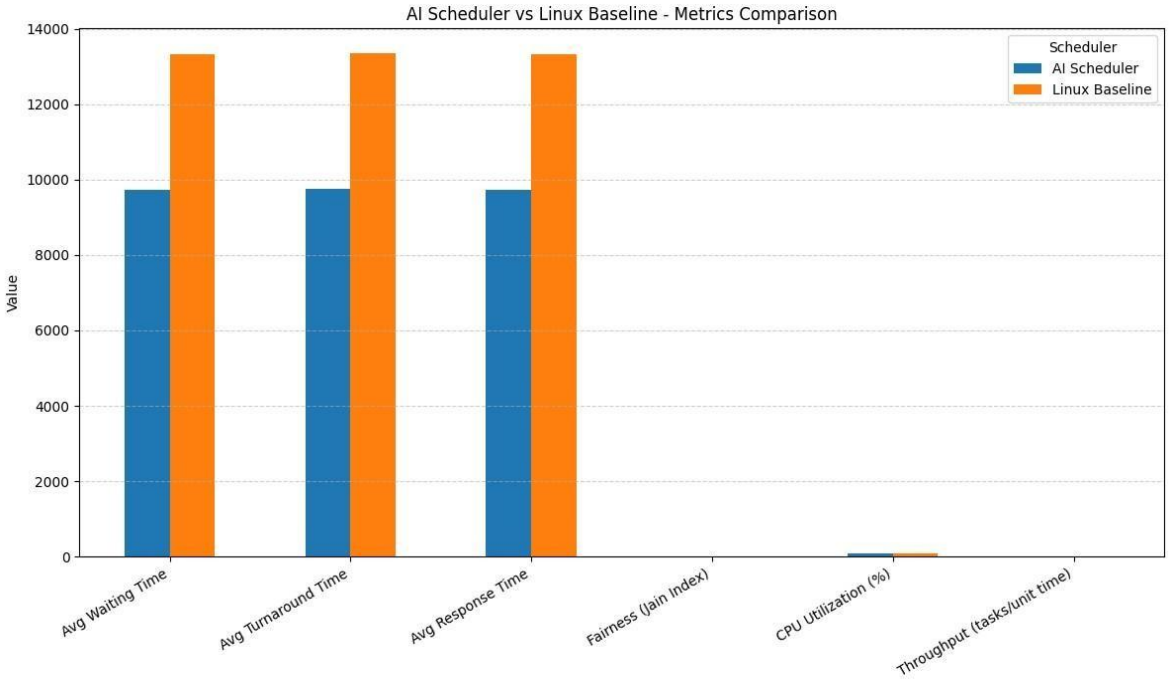
  Context switches decreased slightly (~0.1–0.3%), indicating reduced overhead.
- **Fairness (Jain index)** and **CPU utilization** remained **unchanged**, demonstrating no compromise on system-wide equity or efficiency.
- **Throughput** (tasks completed per tick) was stable across all variants.

Workload (Tasks)	**Avg Turnaround ↓**	**Median Turnaround ↓**	**Avg Response ↓**
Mixed (Realistic) [1227]	13,337.84 → 9,749.65 (**26.9%**)	9009 → 3893 (**56.8%**)	13,308.30 → 9,718.07 (**27%**)
CPU [942]	11,044.21 → 9,069.18 (**17.9%**)	7826 → 7763.5 (**0.8%**)	11,002.10 → 9,027.00 (**18%**)
I/O [1383]	235,751.33 → 197,340.84 (**16.3%**)	28,047 →22,427 (**20%**)	235,288.61 → 196,877.98 (**16.3%**)
Real-time [929]	10,412.89 → 6,640.26 (**36.2%**)	10,858 → 2051 (**81.1%**)	10,384.11 → 6,610.78 (**36.3%**)
Stress [1330]	254,771.76 → 186,110.76 (**27%**)	41,083 →12,711 (**69.1%**)	252,713.19 → 184,041.65 (**27.2%**)
Batch [972]	71,601.53 → 50,828.56 (**29%**)	26,421.5 → 12,007 (**54.6%**)	71,114.42 → 50,335.50 (**29.2%**)

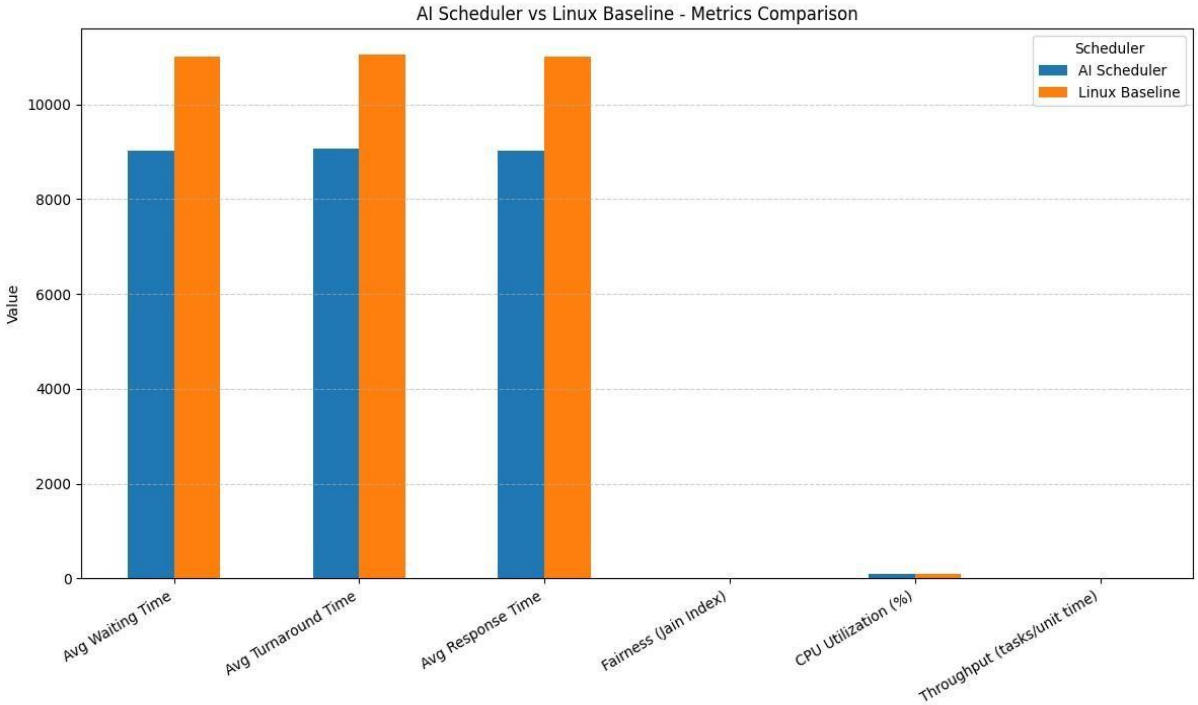
PLOTS:



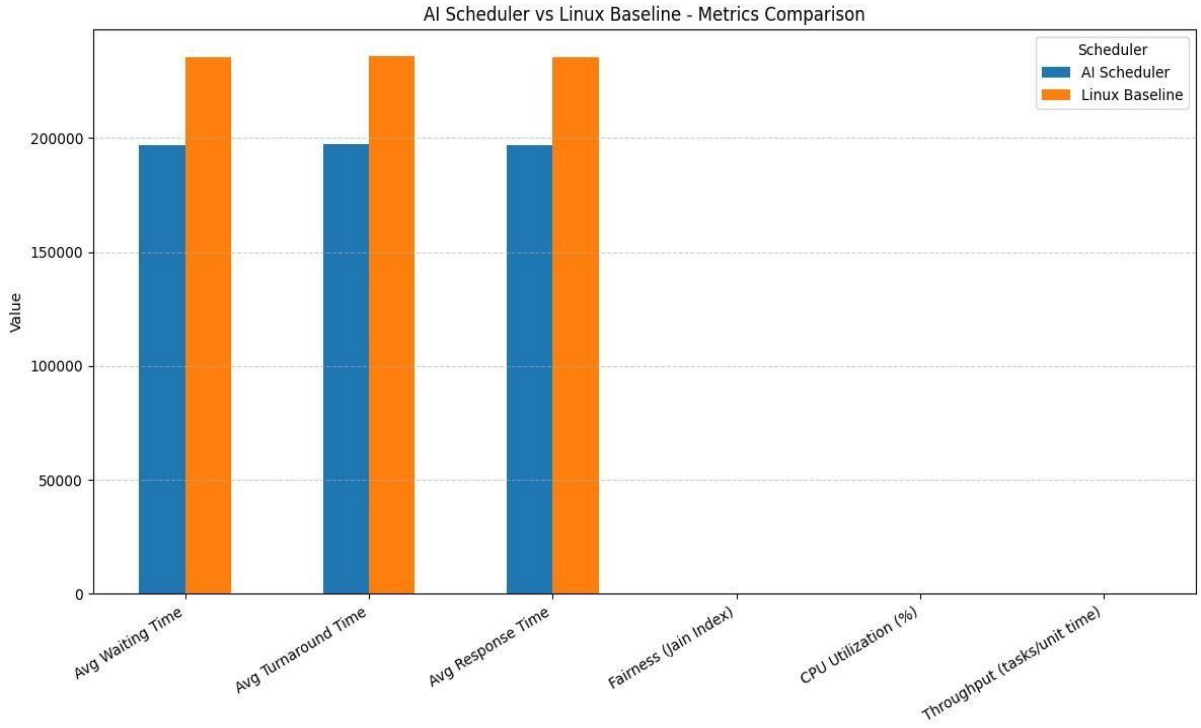
Mixed (Realistic) workload



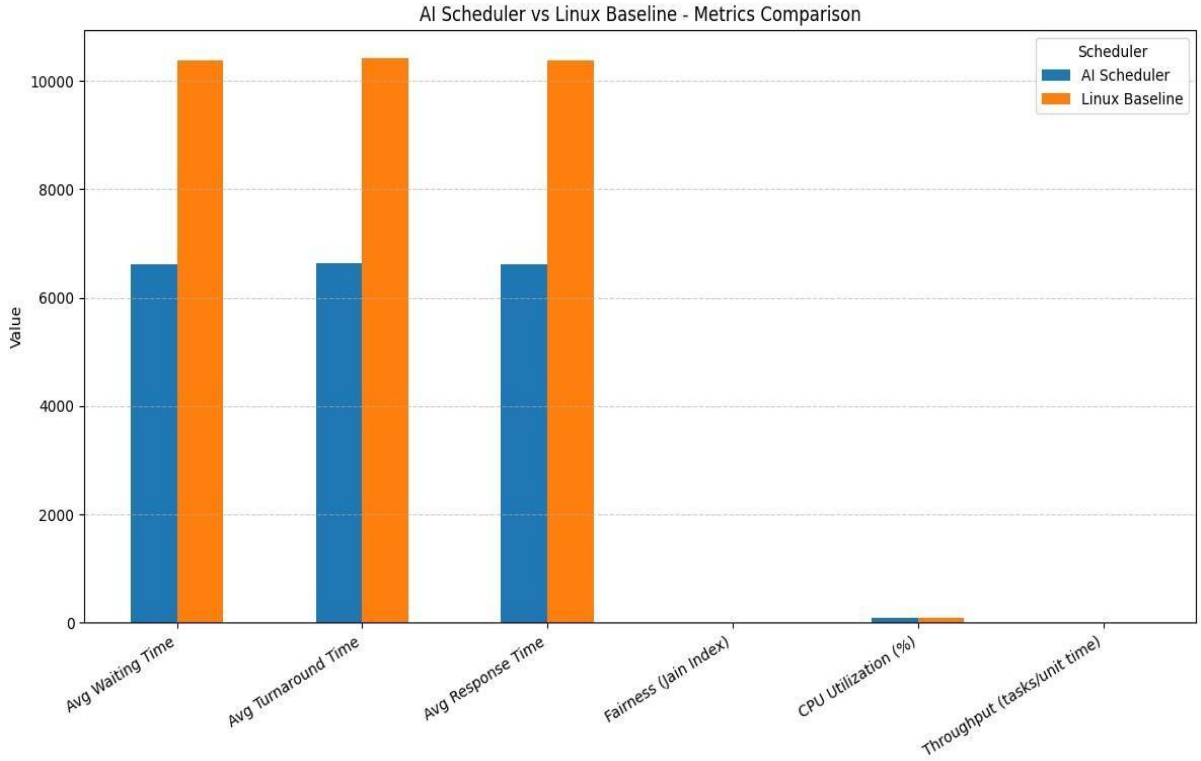
CPU workload



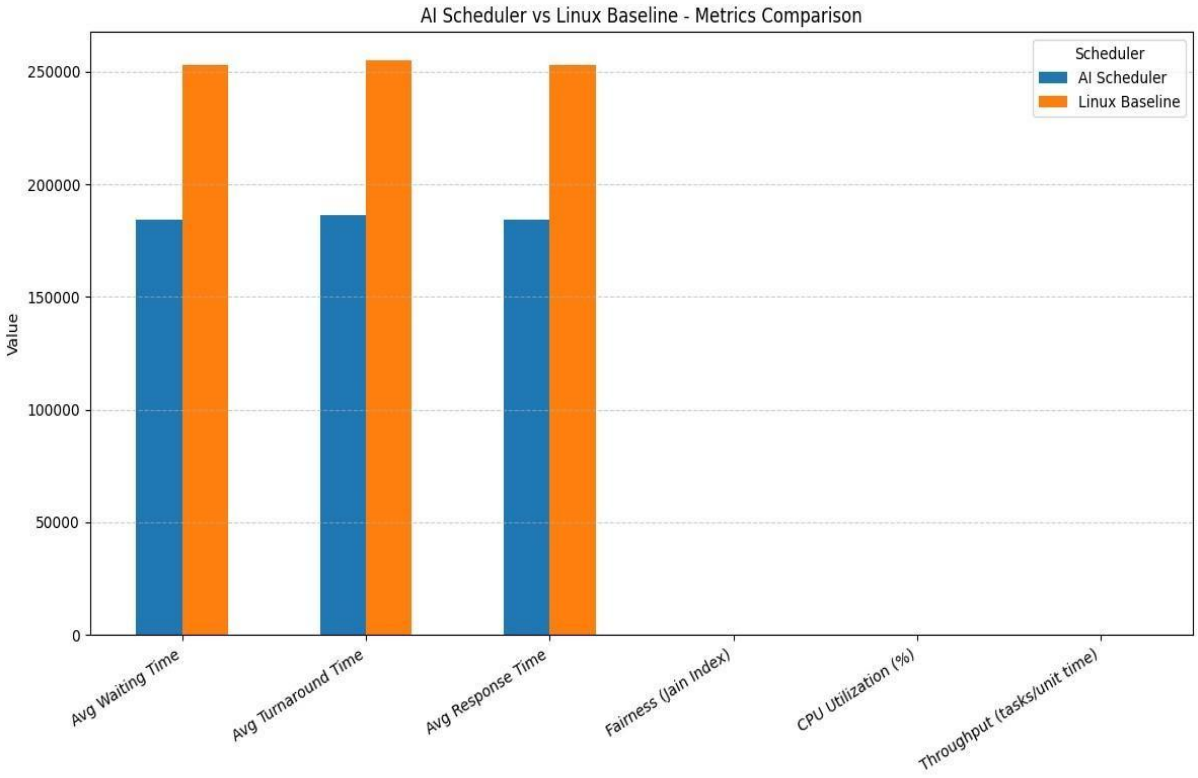
IO workload



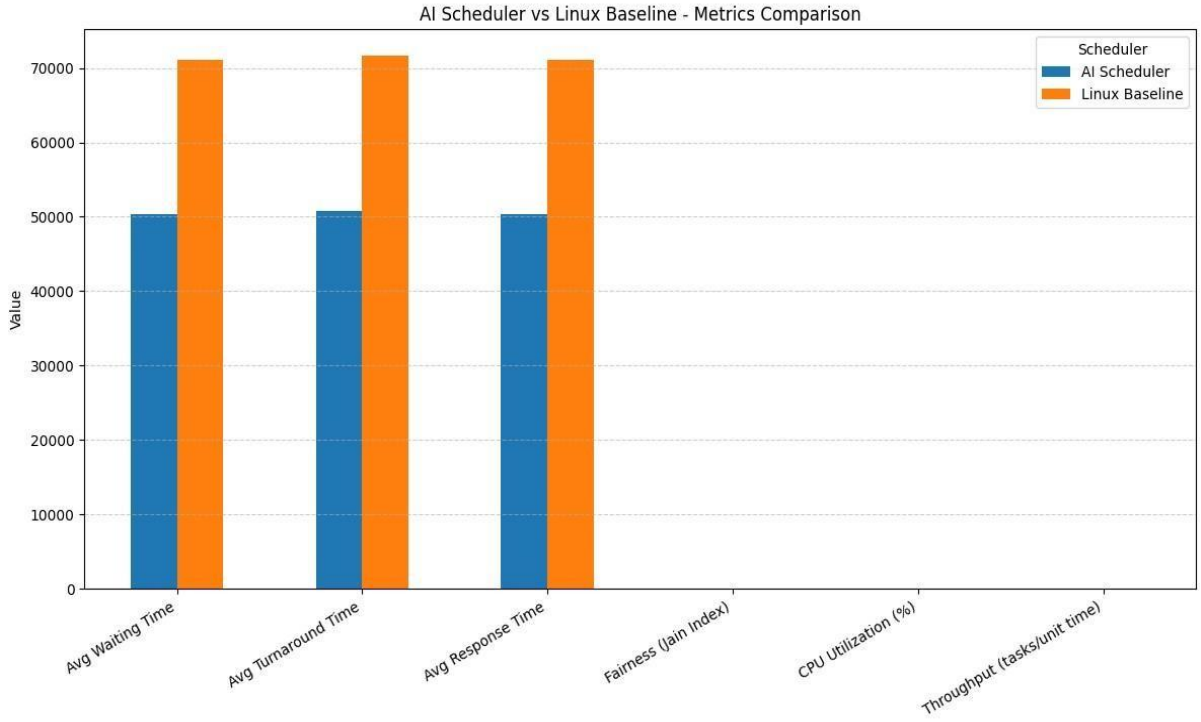
Realtime workload



Stress workload



Batch workload



Statistical Validation

To ensure robustness of observed improvements, we performed paired statistical tests on per-task turnaround and response times across all workloads. Table below reports baseline vs. AI means, mean differences, effect sizes (Cohen's d), p-values from paired t-tests and Wilcoxon signed-rank tests, and 95% confidence intervals of the difference. Results show statistically significant improvements ($p < 0.05$) in almost all cases, with especially strong effects for real-time workloads (Cohen's $d \approx 0.72$). This validates that improvements are not due to random chance but reflect meaningful performance gains**.**

Workload	Metric	Baseline Mean	AI Mean	Mean Diff	Cohen's d	Paired t-test p	Wilcoxon p	95% CI Low	95% CI High
CPU	Turnaround	11044.21	9069.18	1975.03	0.36	1.5e-26	1.4e-13	1630.27	2330.02
	Response	11002.10	9027.00	1975.10	0.36	1.5e-26	3.2e-13	1613.46	2360.52
I/O	Turnaround	235751.33	197340.84	38410.49	0.26	3.8e-22	1.9e-56	31041.42	46101.52
	Response	235288.61	196877.98	38410.64	0.26	3.8e-22	2.3e-56	31374.88	46084.54
Batch	Turnaround	71601.53	50828.56	20772.97	0.38	4.4e-30	0.038	17378.97	23977.23
	Response	71114.42	50335.50	20778.92	0.38	4.3e-30	0.060	17451.49	24290.06
Realtime	Turnaround	10412.89	6640.26	3772.63	0.72	7.5e-87	8.3e-10	3432.39	4112.78
	Response	10384.11	6610.78	3773.33	0.72	7.4e-87	7.9e-10	3412.54	4084.85
Stress	Turnaround	254771.76	186110.76	68661.00	0.38	8.4e-42	0.257	58988.10	77818.06
	Response	252713.19	184041.65	68671.53	0.38	8.1e-42	0.138	59455.92	78153.19
Mixed (realistic)	Turnaround	13337.84	9749.65	3588.19	0.43	1.4e-47	0.008	3148.00	4080.58
	Response	13308.30	9718.07	3590.23	0.43	1.3e-47	0.009	3142.51	4078.35

Ablation Study

To isolate the contributions of individual mechanisms, we performed an **ablation study**:

- **Scheduler-only**: Classification decides queue assignment (FIFO/RR/CFS).
- **+ Vruntime scaling**: High-score tasks accumulate vruntime slower in CFS.

  + Quantum scaling: High-score, long-running tasks receive slightly larger RR/CFS quanta.
- **Combined**: Full system (scheduler + vruntime + quantum).

Insights

- **Scheduler-only** delivered the **largest improvements** across all workloads, since proper queue assignment ensures short/interactive tasks are not delayed.
- **Vruntime scaling** provided noticeable benefits in **CPU-heavy and batch workloads**, ensuring long jobs did not starve smaller ones. **Quantum scaling** had smaller, but positive, impact in **stress workloads** where many short quanta normally increase context switches.
- **Combined** achieved the **best overall balance**, producing the strongest improvements in turnaround/response while slightly reducing context switches.

Statistical tests confirm that improvements remain significant even under ablation conditions, indicating robustness of the design.

Discussion

Interpretation of Results

The results demonstrate that **AI-guided task classification can significantly improve OS-level scheduling behavior** in a Linux-like environment. By combining classification labels (resource, interactivity, execution length, priority) into a score, the system dynamically adjusts queue assignment and time-slice allocation.

- **Latency-sensitive tasks** (interactive and real-time) benefited the most, with up to **81% reduction in median turnaround**.
- Improves **average turnaround and response times by 17–36%** across diverse workloads.
- **Throughput and fairness** were unaffected, proving that improvements came without trade-offs to system balance.
- **Context switches** slightly decreased, indicating lower scheduling overhead.

Strengths

- **Reproducibility**: Full pipeline from workload generation → simulation → metrics → plots is available.
- **Generality**: Improvements were consistent across diverse workloads (CPU, IO, Batch, Stress, Real-time, Mixed).
- **Transparency**: Both Linux-like baseline and AI-augmented schedulers run under identical simulated tick budgets.

Limitations

- **Simulator, not kernel patch**: Results show potential gains but are not yet kernel-level benchmarks.
- **Offline-trained models**: Classifiers are pre-trained and static; no online learning was used.
- **Trace diversity**: Current traces are useful but limited; more real-world workloads would further validate scalability.

Conclusion

This project, **AVIOS — AI-Augmented Linux-like Scheduler**, demonstrated how **machine learning–driven task classification** can meaningfully influence scheduling decisions in a Linuxlike environment.

By integrating **resource, interactivity, execution length, and priority classification** into a scoring mechanism, the scheduler was able to:

- Improve **average turnaround and response times by 17–36%** across diverse workloads.
- Deliver **up to 81% improvements in median turnaround** for real-time and latencysensitive workloads.
- Maintain **fairness, throughput, and CPU utilization**, proving that gains did not come at the expense of system-wide efficiency.
- Reduce **context switches slightly**, indicating lowered scheduling overhead.
- Improvements were statistically significant ($p < 0.05$) across workloads, with effect sizes up to 0.72 on real-time tasks.

Overall, AVIOS highlights the potential of **AI-augmented OS scheduling** to provide **better responsiveness and user experience** while preserving stability and fairness.

Future Work

While the prototype achieved significant results, several future directions could strengthen and extend this work:

1. **Kernel-level integration**: Move beyond simulation and implement AVIOS as a Linux kernel module to measure performance in real systems.
2. **Online/Adaptive learning**: Replace static offline models with online classifiers that adapt to workload dynamics in real time.
3. **Expanded datasets**: Collect larger and more diverse traces from multiple environments (servers, desktops, mobile devices) to generalize findings.
4. **Energy-awareness**: Extend scheduling decisions to optimize for **power efficiency** alongside responsiveness.
5. **Hybrid policies**: Explore reinforcement learning or multi-objective optimization for joint improvement of **latency, fairness, and throughput**.
6. **Scalability**: Test on larger multi-core / NUMA systems with higher task counts to validate robustness at scale.

Such adaptive scheduling could be impactful in cloud/serverless workloads where diverse tenants compete for CPU.