Meta TaskWave: Meta-Selection for Hybrid Dynamic Scheduling in FaaS Environments

Theresa Rivera To
240419758
Dr. Chris Phillips
MSc Computing and Information Systems

Abstract— **Function-as-a-Service (FaaS) platforms require efficient task scheduling across diverse infrastructures. Despite recent proposals for complex optimisation methods, there is a lack of systematic evaluation of basic scheduling algorithms in FaaS environments, leaving performance trade-offs ambiguous. This paper introduces Meta TaskWave, a hybrid scheduling framework that combines job-side algorithms (Round Robin, Earliest Deadline First, Urgency First) with worker-side algorithms (Random, Round Robin, Least Loaded Fair, Fastest Worker Fair, Network Optimal Fair) to create 15 distinct strategies. These are evaluated using a new thermal efficiency metric and benchmarking methodology. Our analysis of 109,568 jobs shows that worker selection is crucial for performance under light workloads (Cramér's V = 0.33), while job prioritisation has little effect ($p > 0.05$, $V < 0.01$). We establish the priority order as: container lifecycle management > worker selection > job selection. Further analysis indicates that penalties contribute to over 98% of performance variation (Coefficient of Variation reduced from 227% to 23-101%). Cross-validation shows that thermal stability patterns predict penalty avoidance ($r = -0.89$, $p < 0.001$), supporting the framework's potential for adaptive meta-scheduling through clear algorithm selection.**

Meta TaskWave provides the first transparent framework for comparative analysis of hybrid scheduling strategies in FaaS environments, bringing to light the specific trade-offs between different approaches rather than treating scheduling as a black-box optimisation problem. Our open-source implementation enables reproducible research in distributed scheduling algorithm selection and generates labelled performance data essential for future adaptive meta-scheduling systems.

Keywords—Function-as-a-Service, Hybrid Scheduling, Distributed Systems, Algorithm Selection, Performance Evaluation

I. INTRODUCTION

A. *Context and Market Growth*

Serverless computing represents "a cloud computing paradigm encompassing a class of cloud computing platforms that allow one to develop, deploy, and run applications (or components thereof) in the cloud without allocating and managing virtualized servers and resources or being concerned about other operational aspects" (Kounev et al., 2023). This paradigm, exemplified by Function-as-a-Service (FaaS) platforms such as AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions, has experienced unprecedented growth in recent years.

The FaaS market is evolving, with a 2022 survey identifying diverse applications of serverless technology as essential building blocks. These applications span various domains, from mobile development (where they implement core backend functionality) to scientific computing, including tasks like metadata extraction, seismic imaging, and genomics processing (Eismann et al., 2022). This expansion reflects a growing trend: although estimates vary depending on scope and methodology, all major analyses point to significant market growth, with total addressable market projections ranging from $44.7 billion to $124.52 billion by 2030–2034. This corresponds to compound annual growth rates between 15.3% and 27.8% (Grand View Research Inc, 2024; Markets and Markets, 2024; Precedence Research, 2025).

The impressive expansion of serverless adoption is largely driven by its pay-per-execution billing model, which charges users based on actual resource usage rather than pre-allocated capacity. This shift transforms cost structures from fixed capital expenditure to variable operational expenses, a model that aligns well with scalable backend services and resource-intensive computing tasks in scientific domains. FaaS platforms promise simplified deployment, automatic scaling, and fine-grained pay-per-execution pricing models that eliminate server provisioning concerns for developers.

B. *The Fundamental Scheduling Challenge*

While removing operational responsibility is appealing for function developers utilising FaaS on cloud platforms, the abstraction comes at the cost of reduced control over resource allocation and scheduling decisions. Recent expert analysis has identified key performance challenges in serverless computing, with particular emphasis on "cold-start latencies" and "autoscaling" as critical issues that demand further research attention (Kounev et al., 2023). This challenge arises in capacity allocation decisions, as platforms strive to balance resource efficiency with performance (Eismann et al., 2021). When demand fluctuates, serverless platforms encounter a critical trade-off: keeping application resources idle leads to memory costs, while scaling down to zero results in cold-start penalties. These penalties can be alleviated through techniques such as function prebaking (Silva, Fireman & Pereira, 2020) or enhanced caching approaches (Fuerst & Sharma, 2021).

Current serverless platforms utilise proprietary, black-box scheduling systems that lack transparency in their decision-making processes. While these systems may perform adequately, their opacity hinders systematic analysis, reproducible research, and principled improvements. Offloading operational responsibilities to cloud providers has created a significant research gap, as no systematic framework exists to evaluate how various scheduling strategies manage critical resource allocation decisions that affect serverless computing performance under different workloads. Most academic research has focused on single-algorithm optimisations or specific aspects of performance within proprietary platforms, leaving a considerable gap in understanding how different scheduling strategies perform across diverse scenarios.

C. *Research Gap and Motivation*

FaaS platforms abstract infrastructure management but create opacity around resource allocation decisions, hindering systematic analysis and environmental accountability. This abstraction also introduces uncertainty, not just about how resources are allocated, but also about what "serverless" truly entails — the definition remains fluid, often masking trade-offs between cold start delays, resource provisioning, and cost (Kounev et al., 2023).

More critically, this opacity extends beyond performance concerns to environmental impact. Decisions about when and where functions are executed directly affect energy usage, system load, and carbon footprint. Yet in most FaaS platforms, these decisions are hidden from users. As a result, developers cannot meaningfully assess the true cost, whether financial or environmental, of each invocation.

Recent work by Sarkar et al. (2024) demonstrates what becomes possible when resource illustrates what becomes possible when such decisions are made visible and optimised. Their DC-CFR framework coordinates cooling, load shifting, and battery usage in data centres, reducing carbon emissions by 14.46%. Crucially, this optimisation is only feasible because the system exposes its internal decision-making processes.

FaaS platforms function as black boxes, and there is no systematic examination of algorithmic trade-offs. This lack of transparency obstructs optimisation and environmental accountability. Few academic proposals on AI scheduling address the need for clarity in algorithmic trade-offs for developers and platform operators.

This project addresses that gap by proposing a benchmarkable, hybrid meta-scheduling framework for FaaS-like environments. Rather than introducing a new black-box optimiser, Meta TaskWave systematically evaluates combinations of job- and worker-side scheduling strategies under varied workload intensities. In doing so, it provides a foundation for more interpretable, stakeholder-aware optimisation in serverless computing.

D. Research Questions and Objectives

To guide this investigation, one primary and four supporting research questions were developed. These are summarised in TABLE I

TABLE I. SUMMARY OF THE RESEARCH QUESTIONS ADDRESSED IN THIS STUDY.

Question Type	Research Question
Primary	How do scheduling algorithms differ in observable behaviours within FaaS platforms, and when do these differences matter for performance?
Supporting 1	Can we build a framework to characterise scheduling trade-offs despite system-level noise?
Supporting 2	What distinct behavioural patterns emerge when algorithms are properly characterised?
Supporting 3	How can these patterns inform adaptive scheduling decisions?

These questions underpin the benchmarking experiments and analyses in the following chapters, aiming to evaluate the performance, transparency, and trade-offs of hybrid scheduling strategies under varying load conditions.

II. Related Works

This section examines prior research across five key areas that inform our hybrid scheduling approach: FaaS workload characterisation, performance metrics and evaluation methodologies, container scheduling in distributed systems, function execution patterns, and recent advances in AI-driven scheduling. We identify gaps in existing approaches that motivate our systematic hybrid framework.

A. FaaS Workload Characterisation

Understanding real-world serverless workload characteristics is fundamental to designing effective scheduling algorithms. Shahrad et al. (2020) and Shahrad, Balkind & Wentzlaff (2019) provided the first comprehensive characterisation of production FaaS workloads, analysing 14 days of Microsoft Azure Functions data. Their landmark study revealed that functions exhibit extreme heterogeneity with invocation frequencies varying across eight orders of magnitude. Most functions are invoked very infrequently, while a small subset experiences massive scale, particularly inter-arrival times ranging from sub-second bursts under heavy usage to minutes or hours between invocations for rarely used functions. Over half of the applications contained only a single function, and memory allocation patterns varied widely across deployments.

Joosen et al. (2023) extend these findings with long-term cloud traces involving 1.4 trillion function invocations, highlighting burst patterns of billions of calls per day as well as long idle gaps. Both studies underscore the highly non-uniform and bursty nature of real-world FaaS workloads.

B. Cold Start Analysis

Cold starts remain one of the most critical performance bottlenecks in FaaS systems, particularly due to the hidden cost of keeping functions in a warm state between invocations. Although serverless pricing follows pay-per-use models, developers incur charges for pre-allocated memory and warm idle periods, making cold starts both a performance and cost transparency concern. Beyond infrastructure-level mitigation techniques, recent research has explored predictive invocation modelling and runtime optimisation to reduce cold start frequency and its financial impact.

A most critical finding for scheduling research was made by Shahrad et al., (2020) on how hybrid keep-alive policies can reduce cold starts by 2.5× while maintaining the same memory footprint as fixed policies. This insight highlights the importance of scheduling strategies that optimise resource retention and invocation timing to minimise cold start impact

Moreover, understanding function execution patterns is crucial for effective scheduling decisions. Steinbach et al. (2022) used neural Temporal Point Processes (TPPs) to model function invocations in FaaS compositions, predicting probability distributions over time and class of following invocations. Their models achieved mean absolute errors below 22ms and correctly predicted function classes above 94% for most compositions. This temporal modelling approach provides precise prediction capabilities for function invocation patterns, enabling proactive scaling and resource allocation that supports our scheduling algorithms.

Complementing this work, Yu et al. (2020) introduced ServerlessBench (SeBs), a comprehensive benchmarking framework for characterising serverless platforms across multiple dimensions including communication efficiency, startup latency, and performance isolation. Their evaluation of AWS Lambda, OpenWhisk, and Fn platforms using realistic workloads revealed significant performance variations across platforms and programming languages, highlighting the importance of systematic evaluation methodologies.

Ustiugov et al., (2021) conducted a comprehensive analysis of snapshot-based serverless infrastructure using the vHive framework, finding that the execution time of functions started from snapshots is 95% higher on average than memory-resident functions. Their analysis of cold-start patterns and function execution characteristics provides crucial insights for scheduling algorithms that need to balance between cold starts and memory usage, directly informing our tiered cold start model.

C. *Performance Benchmarking in Serverless Systems*

Establishing appropriate metrics for serverless performance evaluation has been a significant challenge for researchers, particularly due to the black-box impediment that has restricted researchers to measuring only surface-level metrics while the underlying scheduling decisions remain hidden.

Copik et al. (2021) addressed this through SeBS, the first comprehensive benchmarking framework for FaaS platforms that systematically covers performance metrics including latency, throughput, cold start overhead, memory usage, and I/O performance across multiple providers (AWS Lambda, Azure Functions, Google Cloud Functions). Their framework defines standardised metrics for evaluating serverless performance and highlights key performance features, including cold start penalties for functions with large deployment packages. However, SeBS's emphasis on black-box commercial platforms limits its ability to investigate the underlying scheduling decisions that influence these performance results.

This limitation is part of a broader pattern — Scheuner & Leitner (2020) reviewed 112 studies revealing focus on microbenchmarks measuring CPU speed and platform overhead, whilst exposing gaps in concurrent workload evaluation and systematic scheduling analysis. Combined with SeBS's inability to explain why certain scheduling strategies succeed or fail under specific conditions, these gaps highlight the need for controlled, transparent benchmarking environments.

Building on this foundation, Mahmoudi & Khazaei (2022) proposed the first analytical performance model for serverless platforms that captures unique details including cold start probability, average response time, resource utilisation, and cost metrics. While their model enables workload-aware optimisation that balances cost and performance based on user preferences, it still operates within the constraints of black-box platforms, unable to examine the scheduling mechanisms that determine these outcomes.

Recent research has identified significant cost implications of scheduling decisions in serverless environments. Zhao et al. (2024) demonstrate that OS-level scheduling choices can increase FaaS costs by up to 10X, with simple FIFO strategies outperforming complex schedulers for short-lived functions. These findings highlight the need for systematic evaluation of how algorithm complexity affects performance in serverless environments.

These studies reveal a fundamental research gap: although we can measure and predict performance in serverless systems, we cannot systematically evaluate different scheduling strategies under various conditions. This gap drives Meta TaskWave's configurable simulation approach, which transparently connects performance measurement with scheduling algorithm evaluation.

D. *Container Scheduling Approaches*

While our work focuses on FaaS scheduling, significant insights can be drawn from container orchestration research, particularly hybrid approaches that address similar challenges around dynamic resource allocation and multi-objective optimisation in distributed environments.

Thinakaran et al., (2019) presented Kube-Knots, a GPU-aware resource orchestration system that combines two scheduling techniques to improve cluster-wide GPU utilisation by up to 80% and reduce job completion times by up to 36%. Their success with hybrid scheduling strategies directly validates our approach of combining job- and worker-side algorithms rather than relying on single-algorithm solutions. This multi-objective theme is reinforced by Kaur et al. (2020), whose KEIDS scheduler considers carbon footprints, interference, and energy consumption simultaneously, achieving 14.42% and 31.83% improvements through integer linear programming optimisation.

Recent adaptive scheduling work includes Chu, Li & Qin (2025) who proposed dynamic load balancing using resource-aware task assignment with utility functions encompassing CPU/memory use, historical load metrics, and performance considerations — elements mirroring our design. Whilst enhancing throughput and fairness through adaptive resource distribution in resource-constrained decentralised systems akin to FaaS environments, their AI-driven complexity obscures which decisions contribute to improvements and when simpler methods might suffice. Conversely, Yin, Zhao & Wang, (2024) explored static task allocation for Kubernetes, emphasising predictability in real-time systems, yet their approach cannot adapt to workload fluctuations, precisely the limitation motivating dynamic scheduling research.

While container orchestration platforms like Kubernetes enhance resource management with declarative scaling and adaptive scheduling, serverless computing still faces challenges. Research indicates hybrid and adaptive methods outperform static strategies in resource-constrained settings. However, a significant issue remains: container scheduling systems offer transparency, while FaaS platforms function as black boxes, lacking visibility into scheduling processes. This opacity prevents systematic performance analysis and environmental accountability. Moreover, the serverless field lacks systematic frameworks for understanding fundamental trade-offs informing sophisticated approaches. Meta TaskWave addresses this gap by providing an interpretable foundation for evaluating when different scheduling strategies succeed, enabling more informed development and optimisation of both simple and complex schedulers in serverless environments.

III. Methodology

A. *Design Rationale and Project Philosophy*

Unlike existing benchmarking frameworks such as SeBS (Copik et al., 2021) or vHive (Ustiugov et al., 2021) that operate within black-box commercial platforms, Meta TaskWave was deliberately developed as a transparent, configurable simulation system. This allows fine-grained control over system variables and complete visibility into scheduling decisions, uncovering elements often hidden in real-world FaaS platforms.To maintain practical experiment duration and realistic dynamics, MetaTaskwave's container lifecycle events were time-scaled for cold starts and lifecycle durations.

Other components, such as function execution, job inter-arrival, and network latency, operate in real time to ensure system fidelity.

Each worker in this simulation represents a container instance, providing a warm execution environment for function invocations. The system models a single function (*process_math()*) capable of performing operations like addition and multiplication. The four-worker pool represents the maximum concurrent warm containers for this function, similar to AWS Lambda's dynamic scaling. Bursts of jobs correspond to high-traffic events that trigger multiple concurrent invocations.

By modelling core components of a distributed serverless system, including job generation, scheduling, worker heterogeneity, and cold start behaviour, this framework enables systematic exploration of hybrid scheduling strategies under realistic workload patterns. Ultimately, we hope to expose the essential trade-offs that underpin scheduling decisions — a necessary step towards the development of future meta-scheduling systems.

B. *System Architecture and Parameter Configuration*

Having established the architectural foundation, the next key consideration is ensuring that Meta TaskWave's parameters accurately reflect real-world FaaS characteristics. The following section explains how these system parameters are based on established serverless computing research, ensuring experimental validity and facilitating systematic investigation of scheduling trade-offs.

1) *Job Generator*

TABLE II. JOB GENERATOR CONFIGURATION

Component	Parameter	Simulation Value	Literature Source
Job Workload Profile	Batch size, Load	4-8 jobs, 200-600MB	Shahrad et al., (2020)
	Priority ratio	30% high priority	Vandebon, Coutinho & Luk, 2021; Mahgoub et al., 2022)
Job Algorithm	Deadline (EDF)	Creation time + max wait time	Standard EDF
	Urgency factor	Priority x time	Hybrid framework

The Job Generator (TABLE II) emits synthetic workloads with bursty batch arrivals and configurable job attributes. Each job varies in load and urgency, with 30% flagged as high-priority to reflect differentiated service patterns observed in production FaaS systems.

2) *Central Scheduler:*

The scheduler implements 15 hybrid strategies by combining 3 job-side algorithms with 5 worker-side algorithms:

Job-Side Algorithms:

- **Round Robin** (FIFO): Processes jobs in arrival order, providing baseline first-come-first-served behaviour
- **Earliest Deadline First** (EDF): Prioritises jobs with the earliest deadlines, optimising for time-sensitive workloads
- **Urgency First**: Combines priority and time pressure using priority × time factor, balancing importance with deadline proximity

Worker-Side Algorithms:

- **Random**: Randomly selects available workers, providing baseline distribution
- **Round Robin**: Cycles through workers systematically, ensuring equal job distribution
- **Least Loaded** Fair: Selects workers with lowest current load, optimising for load balancing
- **Fastest Worker Fair**: Chooses workers with highest processing speed, maximising throughput
- **Network Optimal Fair**: Prioritises workers with best network performance, minimising communication overhead

Fairness-aware algorithms switch to least-loaded selection once a worker exceeds 50% of its capacity.

TABLE III. SCHEDULING MATRIX TOTALLING 15 COMBINATIONS

	Random	**Round Robin**	**Least Loaded Fair**	**Fastest Worker Fair**	**Network Optimal Fair**
Round Robin	RR+ Ran	RR+ RR	RR+ LLF	RR+ FWF	RR+ NOF
EDF	EDF+ Ran	EDF+ RR	EDF+ LLF	EDF+ FWF	EDF+ NOF
Urgency First	UF+ Ran	UF+ RR	UF+ LLF	UF+ FWF	UF+ NOF

3) *Worker Pool:*

TABLE IV. WORKER POOL CONFIGURATION

Component	Parameter	Simulation Value	Literature Source
Worker Resources	Memory capacity	600-1200 MB	AWS Lambda allocation model
Worker Performance	Processing speed variation	0.8-1.2×	Mahgoub et al. (2022); Copik et al. (2021)
Worker N`etwork	Latency range	0.05-0.2s	Shahrad, Balkind & Wentzlaff (2019); Yu et al. (2020)
	Proximity factor	Distance-based routing	Chu, Li & Qin (2025) - edge computing scheduling
Worker Scaling	CPU-memory allocation	1,769 MB = 1 vCPU	AWS Lambda resource model
Worker Heterogeneity	Resource variation	Multi-node clustering	Mahgoub et al. (2022)

The Worker Pool simulates heterogeneous clustering across different nodes with varied memory capacity, processing speed, network latency, and proximity factors. Configuration details are presented in TABLE IV, reflecting real serverless infrastructure characteristics from distributed environments (Mahgoub et al., 2022).

C. *System Implementation Details*

1) *Cold Start Lifecycle*

Cold start thresholds are scaled using a 1:60 compression factor. A container transitions to COLD after 60 seconds of simulated inactivity, equivalent to 60 minutes in real FaaS environments. This scaling enables cold start effects to emerge within a 3-minute experimental window, simulating AWS Lambda's container lifecycle dynamics under high-frequency loads, whilst the stabilisation wait of 10 seconds follows SeBS benchmarking practices(Copik et al., 2021).

The cold start model adopts a three-tier state system based on user experience thresholds (Nielsen, 1994). States are defined as HOT (0-10% idle time), WARM (10–80%, 20% penalty), and COLD (>80%, 300ms penalty). Thresholds align with AWS Lambda standards and empirical cold start behaviour (Shahrad, Balkind & Wentzlaff, 2019).

Cooling Penalty Parameters:

- **Base Penalty**: 300ms (Shahrad, Balkind & Wentzlaff, 2019)
- **Warm Threshold**: 60 seconds simulation, equivalent to 60-minute real-world duration with 1:60 scale applied (Amazon Web Services, Inc., 2025)
- **State Penalties**: HOT (0%), WARM (20%), COLD (100%)

2) Workload Intensities

Each hybrid configuration is evaluated over three workload intensities, with time compression for lifecycle simulation whilst preserving realistic invocation patterns:

- **Light**: 8-15s inter-arrival delays (low-frequency function calls)
- **Moderate**: 2-5s delays (typical enterprise API usage)
- **Heavy**: 0.5-1.5s delays (high-frequency invocation scenarios)

3) Single Benchmark Trial Process:

Each benchmark trial follows a structured protocol:

Benchmarking Parameters:

- **Warm-up Period**: 30 seconds simulation (30-minute real-world equivalent)
- **Measurement Duration**: 180 seconds simulation (3-hour real-world equivalent)
- **Stabilisation Wait**: 10 seconds simulation (10-minute real-world equivalent) (SeBS standard)

Bursts of jobs correspond to high-traffic events that trigger multiple invocations concurrently.

D. Data Collection Strategy

1) Benchmark Run Configuration

The experimental design includes a total of 1,050 benchmark runs across all algorithm combinations and workload intensities. To address variations in completion rates, iteration balancing is employed: light workloads undergo 30 trials per algorithm, while moderate and heavy workloads are limited to 20 iterations to mitigate lower completion rates observed under extended inter-arrival patterns.

2) Sample Filtering and Metrics

Analysis focuses on 109,568 measurement-phase completions, filtered from 320,373 total job attempts to exclude warm-up period and incomplete assignments. Five core metrics are extracted from individual job completion records:

Primary Response Metrics:

- **Queue Time**: Delay from job submission to worker assignment (0ms or ~1000ms binary pattern due to discrete 4-worker pool vs production's ~100+ containers)
- **Processing Time:** Actual computation duration (~1ms baseline, 24% CV across algorithms)
- **Cooling Penalty**: Container initialisation overhead (0ms HOT, 50ms WARM, 300ms COLD categorical states due to 1:60 time compression vs continuous real-world degradation)
- **Completion Status**: Job outcome classification: Completed, Assigned (incomplete due to time constraints).

Derived Analysis Metric:

- **Total Response Time:** Queue Time + Processing Time + Cooling Penalties

Step-function patterns in penalties reflect simulation design constraints whilst preserving statistical relationships essential for algorithm comparison.

E. Statistical Evaluation Framework

We employ a four-stage statistical analysis using standard techniques for variance analysis, categorical association, and correlation.(Field & Iles, 2016) Jobs are stratified by penalty profile: (1) No Delay, (2) Cooling Penalty Delay, (3) Queue Time Delay, enabling isolation of algorithmic effects.

1) Distribution Analysis and Stratification Validation

We use the Coefficient of Variation (CV) to evaluate our grouping strategy. It assesses data spread; if our groups effectively isolate different causes of delays, their variation should be significantly lower than that of the overall dataset. The CV for each stratum was calculated as:

$$CV_{stratum} = \frac{\sigma_{stratum}}{\mu_{stratum}} \times 100\%$$

where:

- $\sigma_{\mathrm{stratum}}$: standard deviation of processing times within the stratum
- $\mu_{stratum}$: mean processing time of the stratum

Processing time CV is calculated for each stratum-workload combination to confirm algorithmic differences remain detectable despite penalty effects

2) Variance Component Analysis

Two-way ANOVA (Algorithm × Workload) tests main effects and interactions within each stratum. Three-way ANOVA (Job × Worker × Workload) decomposes hybrid algorithm effects to identify dominant components. Statistical significance indicates if a difference exists, while effect size (η^2) shows if that difference matters. An η^2 of 0.02 means only 2% of performance variation is due to algorithm choice, which is negligible:

$$\eta^2_{component} = \frac{SS_{component}}{SS_{total}}$$

where:

- $SS_{component}$: Algorithmic component sum of squares
- SS_{total}: Total sum of squares

3) Categorical Association Analysis:

Chi-square tests evaluated the independence between algorithm selection and delay patterns across different workload intensities. Cramér's V assessed the strength of association for each algorithm-workload combination..

Standardised residuals were computed to identify algorithm-penalty relationships that deviate significantly from expected frequencies, using the formula:

$$SR_i = \frac{O_i - E_i}{\sqrt{E_i}}$$

where:

- O_i: Observed frequency in cell i
- SS_{total}: Total sum of squares

A threshold of ±1.96 was used to identify cells with significant over- or under-representation, signalling systematic patterns and risk profiles specific to certain algorithms.

4) Thermal Efficiency Analysis and Cross-Validation: Container state transition patterns are extracted from time-series data, classified into three categories:

- **Degradation:** Transitions ending in a colder state (e.g., HOT → WARM/COLD, WARM → COLD)
- **Recovery**: Transitions towards a warmer state (e.g., COLD → WARM/HOT)
- **Stability**: Maintaining the same non-cold state (e.g., HOT → HOT, WARM → WARM).

These transitions form a descriptive Markov chain where containers cycle through *Recovery → Stabilisation→ Degradation* sequences.

Thermal Efficiency (TE_i) is introduced as a novel metric to quantify algorithmic effectiveness in container lifecycle management. It measures how efficiently algorithms maintain container warmth relative to cooling frequency:

$$\mathrm{TE_i} = \frac{\mathrm{L_{stability,i}}}{\mathrm{R_{degradation,i}} \times 100}$$

where:

- $L_{stability,i}$: Average length of stability chains for algorithm i (consecutive HOT→HOT transitions before degradation)
- $R_{degradation,i}$: Degradation rate[1] for algorithm i (frequency of HOT→COLD transitions)

The rationale behind this metric creation is that higher thermal efficiency indicates superior resource management — algorithms that maintain longer periods of container availability while minimising costly cold start triggers.

Cross-validation employs Pearson correlation between thermal efficiency scores and standardised residuals from the Stage 3 (Categorical Association Analysis) to test convergent validity across independent analytical approaches.

IV. RESULTS

A. *Fundamental Priorities: Penalties vs Processing*

Our analysis reveals that container lifecycle management, rather than job processing algorithmic sophistication, dominates performance variability in FaaS platforms.

As shown in TABLE V, Stratification reveals that total response time variability (CV: 227%) compresses to 23-101% within penalty groups, whilst processing time remains consistently ~23-27% across all strata. Full workload-stratified analysis is provided in Appendix A (TABLE IX).

ANOVA tests confirm this finding: whilst algorithm effects achieve statistical significance ($p < 0.0001$), their practical impact is negligible ($\eta^2 < 0.02$ across all strata). Three-way decomposition shows worker-side algorithms marginally outperform job-side strategies, yet both remain within negligible effect boundaries ($\eta^2 < 0.007$). The largest algorithm-induced variation (0.086ms in No Delay stratum) represents only 8.4% on a sub-millisecond baseline imperceptible compared to lifecycle penalties of 50-300ms. The full breakdown of ANOVA results is provided in Appendix A (TABLE X).

TABLE V. COEFFICIENT OF VARIATION ANALYSIS DEMONSTRATING PENALTY-DRIVEN VARIABILITY

	Delay Stratum	Sample Size	CV	Mean (ms)	Std (ms)
Total Response Time	Overall	109,568	226.65%	158.24	358.65
	None	85,697	23.28%	1.03	0.24
	Cooling	7,356	100.90%	86.73	87.55
	Queue	16,515	3.00%	1,005.90	30.48
Job Processing Time	Overall	109,568	23.50%	1.03	0.24
	None	85,697	23.30%	1.02	0.24
	Cooling	7,356	26.70%	1.05	0.28
	Queue	16,515	23.20%	1.03	0.24

Critically, Cohen's d analysis of the largest workload effect (d = 0.340) confirms negligible algorithmic impact.

B. *Algorithm-Penalty Associations*

Having established penalties as the dominant factor, we will proceed to examine which specific algorithmic choices influence penalty occurrence.

1) Job Selection Strategies: Minimal Influence

Chi-square analysis reveals no significant association between job-side algorithms and penalty patterns across all workload intensities ($p > 0.05$). Job algorithms (Round Robin, EDF, Urgency First) demonstrate nearly identical delay stratum distributions: ~78% no delay, ~7% cooling delay, and ~15% queue delay, with Cramér's V values consistently below 0.01 indicating negligible effect sizes.

This uniformity persists across workload intensities. Even under light loads where system resources are most available for algorithmic differentiation, job selection strategies show no meaningful impact on penalty occurrence. Standardised residuals remain within ±1.42 across all conditions, well below the ±1.96 threshold for statistical significance, confirming that job prioritisation mechanisms cannot influence container lifecycle states.

2) Worker Assignment: Dominant Factor

In stark contrast, worker-side algorithms confirms significant associations with penalty patterns ($p < 0.0001$ across all workloads). Under light workloads, this relationship is particularly pronounced ($\chi^2 = 3255.30$, $V = 0.33$, moderate effect), whilst diminishing under moderate ($V = 0.07$) and heavy loads ($V = 0.08$) as system saturation limits algorithmic influence as seen in TABLE VI.

TABLE VI. CHI-SQUARE AND CRAMÉRS V AGAINST WORKER ALGORITHM AND DELAY STRATA

Work-load	Sample Size	Chi² (χ^2)	p-Value	Cra-mérs V	Effect Size
Overall	109,568	781.94	<0.0001	0.0597	Negligible
Light	14,807	3255.30	<0.0001	0.3315	Moderate
Moderate	39,397	352.59	<0.0001	0.0668	Negligible
Heavy	55,364	741.57	<0.0001	0.0818	Negligible

Standardised residuals (SR) reveal striking patterns under light workload conditions: Random worker algorithms show strong positive association with cold starts (residual = +30.2), whilst Fastest Worker Fair demonstrates strong negative association (residual = -17.8). This 48-point residual spread represents the largest algorithmic effect observed, occurring precisely where container thermal management matters most.

[1] $\text{Degradation Rate} = \frac{\sum(\text{HOT} \rightarrow \text{WARM/COLD, WARM} \rightarrow \text{COLD})}{\sum(\text{All transitions})} \times 100\%$

Full residual worker heatmaps across all workload conditions are provided in the Appendix B (Fig. 2)

Worker algorithm choice thus emerges as the primary lever for managing penalty occurrence, but only when system resources permit algorithmic differentiation. Under heavy load, all strategies converge as saturation eliminates room for optimisation.

3) Hybrid Algorithms: Worker Dominance Confirmed

Analysis of all 15 hybrid combinations (TABLE VI) confirms that worker algorithms drive performance outcomes. Under light workloads, hybrid strategies show significant penalty associations ($\chi^2 = 3277.62$, $p < 0.0001$, $V = 0.33$), with decomposition revealing this effect stems entirely from the worker component.

The standardised residuals pattern is unambiguous: the three worst-performing combinations all pair with RR, whilst the best performers all use Fastest Worker Fair (TABLE VII). Interestingly, this dominance reverses under heavy workloads where simple load-spreading algorithms outperform sophisticated thermal optimisers by utilising all workers rather than concentrating on preferred nodes.

TABLE VII. HYBRID ALGORITHM EXTREME STANDARDISED RESIDUALS UNDER LIGHT WORKLOAD

Rank	Best Performers	SR	Worst Performers	SR
1	RR x FWF	-10.69***	UF x RR	+17.86***
2	UF x FWF	-10.43***	RR x RR	+17.80***
3	EDF x FWF	-9.77***	EDF x RR	+16.60***

[a] *** p < 0.001. Average spread within same worker: 1.2 points; between workers: 28.0 points.

Job algorithm choice shows minimal differentiation among worker groups, with only a 1.2-point average spread compared to a 28-point spread between worker algorithms. This 23:1 ratio indicates that worker selection is significantly more influential than job prioritisation under light loads.. Full hybrid algorithm standardised residual heatmaps are provided in Appendix B (Fig. 3)

C. Thermal Stability as Performance Differentiator

Having established worker algorithm dominance, we now examine the underlying thermal mechanisms that drive this performance hierarchy. We focus our analysis on light workloads, where Section B.2 revealed moderate algorithmic effects ($V = 0.33$) on penalty patterns. Worker algorithms manage container "warmth" differently: some keep containers ready for reuse, others let them cool frequently. Degradation measures how often containers transition from hot to cold, whilst stability chains measure consecutive warm periods before cooling. Three metrics capture these thermal patterns:

1) Degradation Ratio

Under the light workload, algorithms show dramatic differences in container cooling frequency (TABLE VIII). Random Worker exhibits the highest degradation rate (34.3%), forcing containers through frequent HOT→COLD transitions. In contrast, Fastest Worker Fair maintains low degradation (16.2%), keeping containers warm between invocations. This 2:1 ratio in degradation frequency directly translates to cold start probability differences.

Interestingly, workload saturation inverts this pattern: Random Worker achieves the lowest degradation rate (0.01%) under heavy load, as its simple assignment ensures all containers remain continuously active, preventing cooling cycles.

2) Stability Chain Length

Stability chains — consecutive HOT-state transitions before degradation — reveal how long algorithms maintain container warmth.

Under light workload, Fastest Worker Fair sustains 4.0-transition stability chains — a 7× improvement over Random Worker (0.6) — by prioritising fast containers that complete jobs quickly without triggering reassignment.

TABLE VIII. THERMAL EFFICIENCY BY WORKER ALGORITHM[B]

Algorithm	Light		Moderate		Heavy	
	Stability	Degrade %	Stability	Degrade %	Stability	Degrade %
FWF	4.0	16.2%	28.5	3.4%	39.1	2.2%
NOF	2.9	20.2%	29.6	3.2%	55.0	1.7%
LLF	2.7	20.4%	16.8	4.2%	--	0.0%
RR	1.2	21.9%	50.8	1.9%	--	0.0%
Ran	0.6	34.3%	12.1	7.0%	16.0	0.0%

[b] Stability: Average consecutive HOT state transitions
Degrade %: Container cooling frequency, i.e., Degradation Rate
--: No thermal cycle detected, i.e. all containers were HOT

3) Discovering the Optimal Sensitivity Zone

The relationship between stability and degradation reveals distinct algorithm behaviours. Under light workload, superior algorithms achieve both long stability chains and low degradation rates, a complementary advantage. Notably, RR Worker demonstrates surprising efficiency under moderate workload (50.8 average stability chain, 1.9% degradation rate), suggesting deterministic assignment can exploit temporal locality when system load permits access to the Optimal Sensitivity Zone.

These thermal patterns directly explain the standardised residual findings from Section B: algorithms with negative residuals (fewer cold starts) achieve this through superior thermal management, whilst those with positive residuals suffer from rapid state oscillation. Full thermal analysis including hybrid algorithms is provided in Appendix C.

D. Cross-Validation of Thermal Mechanisms

Importantly, whilst Section B's chi-square analysis necessarily combined minor cooling (WARM states) with major cold starts (COLD states) due to sample size constraints, Section C's transition analysis reveals the complete thermal lifecycle: recovery (COLD→HOT), stability (HOT→HOT sequences), and degradation (HOT→COLD). This granular view validates that the penalty patterns detected statistically in Section B reflect genuine thermal management differences.

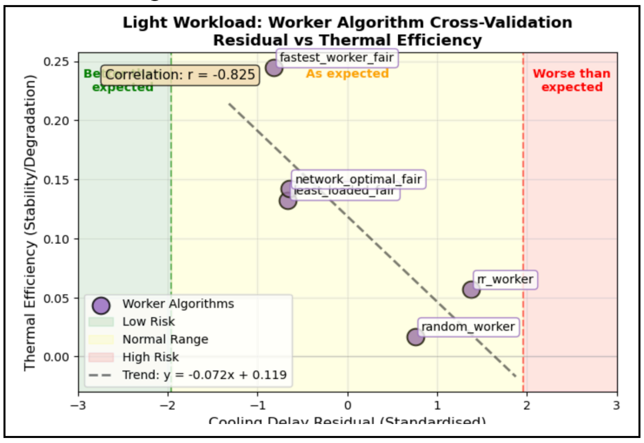

Fig. 1. Worker algorithms with superior thermal efficiency (y-axis) consistently show negative standardised residuals (x-axis), validating thermal management as the mechanism behind penalty avoidance.`

Cross-validation between thermal efficiency and standardised residuals validates our findings at multiple levels. Algorithm-level analysis (n=5 worker algorithms) reveals strong associations under light workload (r = -0.825) that diminish progressively through moderate (r = -0.798) to heavy workload (r = -0.385). Iteration-level analysis confirms these patterns remain significant despite run-to-run variation (Light: r = -0.220, p < 0.01, n=150).

These converging analyses confirm that thermal state management drives performance differences, with effects strongest under light load where algorithmic choice matters most. Full statistical details are provided in **Error! Reference source not found.**.

E. Performance Regime Identification

Synthesising our findings reveals three operational regimes for adaptive scheduling:

- **Light Workload**: Algorithm selection critical. Hybrid analysis confirms strong thermal management effects (r = -0.892, p < 0.001), with worker algorithms as the primary driver (Fastest Worker Fair achieves 7× stability improvement). We emphasise the deployment of thermal-aware strategies.
- **Moderate Workload**:Cross validation exhibits maximum responsiveness to thermal improvements despite transition zone shows diminishing effects (r = -0.770, p < 0.001 in hybrid cross-validation). This suggests this is the Optimal Sensitivity Zone to explore algorithmic optimisation against balanced container provisioning.
- **Heavy Workload**: Saturation removes differentiation; strategies converge to constant HOT states, therefore, we must prioritise low-overhead implementations.

This regime framework, validated through hybrid algorithm analysis (p < 0.001 for light and moderate workloads), enables workload-adaptive scheduling: sophisticated algorithms for light loads, transitioning to simple strategies under saturation, maximising performance and resource efficiency.

V. DISCUSSION / CONCLUSION

This study demonstrates the viability of future meta-scheduling for FaaS platforms through systematic algorithm characteristics. We begin by addressing our supporting questions, followed by our primary research question.

A. Which scheduling factors influence FaaS performance, and when do algorithmic choices matter?

Our penalty-stratified analysis successfully isolates algorithmic effects, revealing that container lifecycle management dominates performance (>98% variance). The framework's thermal stability metrics reveal that thermal stability, rather than raw processing speed, dictates FaaS performance across various workload conditions.

B. What distinct behavioural patterns emerge when algorithms are properly characterised?

Distinct patterns include: (1) Worker algorithm dominance over job selection, (2) Thermal state persistence as the key differentiator, (3) Non-linear sensitivity patterns suggest that moderate workloads reveal a "sweet spot" where sufficient load differentiates algorithms, avoiding saturation effects that obscure their differences, (4) Degradation-Recovery coupling validating thermal mechanisms.

C. How can these patterns inform adaptive scheduling decisions?

Three operational regimes guide adaptation:

- **Light workloads** reward sophisticated thermal management. Deploy algorithms like Fastest Worker Fair to maintain long stability chains and minimise cold starts
- **Moderate workloads** exhibit maximum sensitivity in thermal efficiency and cooling penalties. Prioritise container lifecycle management and explore the optimum number of containers to provide.
- **Heavy workloads** eliminate algorithmic differentiation through saturation. All containers remain continuously HOT, making algorithm choice largely irrelevant. Simple strategies may offer marginal benefits through reduced computational overhead, but performance differences are negligible

D. How do scheduling algorithms differ in observable behaviours within FaaS platforms, and when do these differences matter for performance?

Worker algorithms exhibit distinct thermal management patterns, from Random Worker's chaotic state transitions (34.3% degradation rate) to Fastest Worker Fair's stable container reuse (16.2% degradation, 4× longer stability chains). In terms of job algorithms, their impact is negligible, establishing a clear hierarchy: worker selection is significantly more influential than job prioritisation.

The differences in behaviour directly influence performance outcomes under specific workload conditions. This workload-dependent effect on thermal efficiency, alongside standardised residuals cross-validation, supports the validity of adaptive meta-scheduling as a practical solution for FaaS platforms.

VI. FUTURE WORK

The next step is to validate these findings against production workloads with continuous cooling, as real-world container thermal states degrade gradually. Developing real-time detection algorithms would allow dynamic strategy adjustments as workload intensities vary throughout the day.

The framework should scale to larger heterogeneous clusters, potentially uncovering new thermal management patterns. Multi-objective optimisation could balance thermal efficiency with throughput, cost, and fairness.

Exploring the Optimal Sensitivity Zone may reveal specific algorithm switching thresholds and mathematical boundaries for optimising thermal efficiency. Machine learning could predict workload transitions, enabling proactive scheduling adaptations. Finally, integrating these insights into orchestrators like Kubernetes would illustrate practical deployment paths for thermal-aware meta-scheduling in production FaaS environments.

Whilst our simulation values are tailored to our experimental framework, the identified patterns — such as thermal state dominance, worker algorithm primacy, and workload regime transitions — reflect fundamental scheduling dynamics relevant across various FaaS implementations. These insights focus on trends, rather than absolute metrics, forming our primary contribution.

References

Amazon Web Services, Inc. (2025) *AWS Lambda - Developer Guide*. https://docs.aws.amazon.com/pdfs/lambda/latest/dg/lambda-dg.pdf.

Chu, K., Li, X. & Qin, X. (2025) Optimizing Resource Utilization in Edge Computing Environment with Dynamic Load Balancing Scheduling Algorithm. In: *2025 6th International Conference on Computer Science, Engineering, and Education (CSEE)*. February 2025 pp. 49–56. doi:10.1109/CSEE64583.2025.00017.

Copik, M., Kwasniewski, G., Besta, M., Podstawski, M. & Hoefler, T. (2021) SeBS: a serverless benchmark suite for function-as-a-service computing. In: *Proceedings of the 22nd International Middleware Conference*. Middleware '21. 2 October 2021 New York, NY, USA, Association for Computing Machinery. pp. 64–78. doi:10.1145/3464298.3476133.

Eismann, S., Bui, L., Grohmann, J., Abad, C., Herbst, N. & Kounev, S. (2021) Sizeless: predicting the optimal size of serverless functions. In: *Proceedings of the 22nd International Middleware Conference*. 6 December 2021 Québec city Canada, ACM. pp. 248–259. doi:10.1145/3464298.3493398.

Eismann, S., Scheuner, J., Eyk, E.V., Schwinger, M., Grohmann, J., Herbst, N., Abad, C.L. & Iosup, A. (2022) The State of Serverless Applications: Collection, Characterization, and Community Consensus. *IEEE Transactions on Software Engineering*. 48 (10), 4152–4166. doi:10.1109/TSE.2021.3113940.

Field, A. & Iles, J. (2016) *An adventure in statistics: the reality enigma*. Los Angeles, SAGE.

Fuerst, A. & Sharma, P. (2021) FaasCache: keeping serverless computing alive with greedy-dual caching. In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 19 April 2021 Virtual USA, ACM. pp. 386–400. doi:10.1145/3445814.3446757.

Grand View Research Inc (2024) *Function-as-a-Service Market Size & Outlook, 2030*. 2024. Grand View Research: Global Function-as-a-Service Market Size & Outlook. https://www.grandviewresearch.com/horizon/outlook/function-as-a-service-market-size/global [Accessed: 12 June 2025].

Joosen, A., Hassan, A., Asenov, M., Singh, R., Darlow, L., Wang, J. & Barker, A. (2023) How Does It Function? Characterizing Long-term Trends in Production Serverless Workloads. In: *Proceedings of the 2023 ACM Symposium on Cloud Computing*. SoCC '23. 31 October 2023 New York, NY, USA, Association for Computing Machinery. pp. 443–458. doi:10.1145/3620678.3624783.

Kaur, K., Garg, S., Kaddoum, G., Ahmed, S.H. & Atiquzzaman, M. (2020) KEIDS: Kubernetes-Based Energy and Interference Driven Scheduler for Industrial IoT in Edge-Cloud Ecosystem. *IEEE Internet of Things Journal*. 7 (5), 4228–4237. doi:10.1109/JIOT.2019.2939534.

Kounev, S., Herbst, N., Abad, C.L., Iosup, A., Foster, I., Shenoy, P., Rana, O. & Chien, A.A. (2023) Serverless Computing: What It Is, and What It Is Not? *Commun. ACM*. 66 (9), 80–92. doi:10.1145/3587249.

Mahgoub, A., Yi, E.B., Shankar, K., Minocha, E., Elnikety, S., Bagchi, S. & Chaterji, S. (2022) WISEFUSE: Workload Characterization and DAG Transformation for Serverless Workflows. *Proc. ACM Meas. Anal. Comput. Syst.* 6 (2), 26:1-26:28. doi:10.1145/3530892.

Mahmoudi, N. & Khazaei, H. (2022) Performance Modeling of Serverless Computing Platforms. *IEEE Transactions on Cloud Computing*. 10 (4), 2834–2847. doi:10.1109/TCC.2020.3033373.

Markets and Markets (2024) *Serverless Computing Market - Global Forecast to 2029*. September 2024. Markets and Markets. https://www.marketsandmarkets.com/Market-Reports/serverless-computing-market-217021547.html [Accessed: 1 July 2025].

Nielsen, J. (1994) *Usability Engineering*. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc.

Precedence Research (2025) *Serverless Architecture Market Size, Share, and Trends 2025 to 2034*. February 2025. Precedence Research. https://www.precedenceresearch.com/serverless-architecture-market [Accessed: 1 July 2025].

Sarkar, S., Naug, A., Luna, R., Guillen, A., Gundecha, V., Ghorbanpour, S., Mousavi, S., Markovikj, D. & Babu, A.R. (2024) Carbon footprint reduction for sustainable data centers in real-time. In: *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI'24/IAAI'24/EAAI'24. 20 February 2024 AAAI Press. pp. 22322–22330. doi:10.1609/aaai.v38i20.30238.

Scheuner, J. & Leitner, P. (2020) Function-as-a-Service performance evaluation: A multivocal literature review. *Journal of Systems and Software*. 170, 110708. doi:10.1016/j.jss.2020.110708.

Shahrad, M., Balkind, J. & Wentzlaff, D. (2019) Architectural Implications of Function-as-a-Service Computing. In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 12 October 2019 Columbus OH USA, ACM. pp. 1063–1075. doi:10.1145/3352460.3358296.

Shahrad, M., Fonseca, R., Goiri, Í., Chaudhry, G., Batum, P., Cooke, J., Laureano, E., Tresness, C., Russinovich, M. & Bianchini, R. (2020) *Serverless in the Wild: Characterizing* *and Optimizing the Serverless Workload at a Large Cloud Provider*. In: 2020 pp. 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad.

Silva, P., Fireman, D. & Pereira, T.E. (2020) Prebaking Functions to Warm the Serverless Cold Start. In: *Proceedings of the 21st International Middleware Conference*. 7 December 2020 Delft Netherlands, ACM. pp. 1–13. doi:10.1145/3423211.3425682.

Steinbach, M., Jindal, A., Chadha, M., Gerndt, M. & Benedict, S. (2022) TppFaaS: Modeling Serverless Functions Invocations via Temporal Point Processes. *IEEE Access*. 10, 9059–9084. doi:10.1109/ACCESS.2022.3144078.

Thinakaran, P., Gunasekaran, J.R., Sharma, B., Kandemir, M.T. & Das, C.R. (2019) Kube-Knots: Resource Harvesting through Dynamic Container Orchestration in GPU-based Datacenters. In: *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. September 2019 pp. 1–13. doi:10.1109/CLUSTER.2019.8891040.

Ustiugov, D., Petrov, P., Kogias, M., Bugnion, E. & Grot, B. (2021) Benchmarking, analysis, and optimization of serverless function snapshots. In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '21. 17 April 2021 New York, NY, USA, Association for Computing Machinery. pp. 559–572. doi:10.1145/3445814.3446714.

Vandebon, J., Coutinho, J.G.F. & Luk, W. (2021) Scheduling Hardware-Accelerated Cloud Functions. *Journal of Signal Processing Systems*. 93 (12), 1419–1431. doi:10.1007/s11265-021-01695-7.

Yin, J., Zhao, Y. & Wang, H. (2024) A Static Task Allocation and Scheduling Algorithm for Kubernetes Cluster. In: *2024 IEEE 7th International Conference on Information Systems and Computer Aided Education (ICISCAE)*. September 2024 pp. 175–179. doi:10.1109/ICISCAE62304.2024.10761792.

Yu, T., Liu, Q., Du, D., Xia, Y., Zang, B., Lu, Z., Yang, P., Qin, C. & Chen, H. (2020) *Characterizing serverless platforms with serverlessbench*. In: SoCC '20. 12 October 2020 New York, NY, USA, Association for Computing Machinery. pp. 30–44. doi:10.1145/3419111.3421280.

Zhao, Y., Weng, W., van Nieuwpoort, R. & Uta, A. (2024) In Serverless, OS Scheduler Choice Costs Money: A Hybrid Scheduling Approach for Cheaper FaaS. In: *Proceedings of the 25th International Middleware Conference*. Middleware '24. 2 December 2024 New York, NY, USA, Association for Computing Machinery. pp. 172–184. doi:10.1145/3652892.3700757.

APPENDIX

Appendix A. Fundamental Priorities: Penalties vs Processing

TABLE IX. COEFFICIENT OF VARIATION BREAKDOWN PER STRATUM AND WORKLOAD ITENSITY

	Delay Stratum	**Workload Intensity**	**Sample Size**	**CV**	**Mean (ms)**	**Std (ms)**
Total Response Time	None	Overall	109,568	226.65%	158.24	358.65
		Light	14,807	222.70%	139.62	310.88
		Moderate	39,397	241.50%	143.68	346.98
		Heavy	55,364	217.50%	173.58	377.59
	No Delay	Overall	85,697	23.28%	1.03	0.24
		Light	7,653	28.36%	0.96	0.27
		Moderate	32,441	24.86%	1.01	0.25
		Heavy	45,603	21.00%	1.05	0.22
	Cooling Delay	Overall	7,356	100.94%	86.73	87.55
		Light	5,637	100.96%	86.87	87.70
		Moderate	1,483	100.84%	85.15	85.86
		Heavy	236	100.63%	93.51	94.09
	Queue Delay	Overall	16,515	3.03%	1,005.90	30.48
		Light	1,517	7.13%	1,035.11	73.82
		Moderate	5,473	2.81%	1,005.22	28.24
		Heavy	9,525	1.14%	1,001.63	11.46
Job Processing Time	Unstratified	Overall	109,568	23.50%	1.03	0.24
		Light	14,807	28.20%	0.99	0.28
		Moderate	39,397	24.80%	1.01	0.25
		Heavy	55,364	21.10%	1.05	0.22
	No Delay	Overall	85,697	23.30%	1.02	0.24
		Light	7,653	28.36%	0.96	0.27
		Moderate	32,441	24.86%	1.01	0.25
		Heavy	45,603	21.00%	1.05	0.22
	Cooling Delay	Overall	7,356	26.70%	1.05	0.28
		Light	5,637	27.44%	1.03	0.28
		Moderate	1,483	23.97%	1.10	0.26
		Heavy	236	21.32%	1.13	0.24
	Queue Delay	Overall	16,515	23.20%	1.03	0.24
		Light	1,517	28.73%	0.99	0.29
		Moderate	5,473	24.57%	1.01	0.25
		Heavy	9,525	21.34%	1.05	0.22

TABLE X. TWO-WAY AND THREE-WAY ANOVA: STATISTICAL SIGNIFICANCE AGAINST EFFECT SIZE

Stratum	**ANOVA**	**Effect Type**	**η² Effect Size**	**Effect Classification**	**p-value**	**Significant?**
No Delay	Two-way	Algorithm Combination	0.0079	Negligible	<0.0001	Yes
		Workload Intensity	0.0135	Small	<0.0001	Yes
		Algorithm × Workload Interaction	0.0075	Negligible	<0.0001	Yes
	Three-way	Job Algorithm	0.0002	Negligible	<0.0001	Yes
		Worker Algorithm	0.0068	Negligible	<0.0001	Yes
		Job × Worker Interaction	0.0011	Negligible	<0.0001	Yes
		Worker × Workload Interaction	0.0066	Negligible	<0.0001	Yes
Cooling Delay	Two-way	Algorithm Combination	0.0196	Small	<0.0001	Yes
		Workload Intensity	0	Negligible	1	No
		Algorithm × Workload Interaction	0.0057	Negligible	0.1561	No
	Three-way	Job Algorithm	0	Negligible	1	No
		Worker Algorithm	0	Negligible	1	No
		Job × Worker Interaction	0	Negligible	1	No
		Worker × Workload Interaction	0	Negligible	1	Ni
Queue Time Delay	Two-way	Algorithm Combination	0.005	Negligible	<0.0001	Yes
		Workload Intensity	0.0094	Negligible	<0.0001	Yes
		Algorithm × Workload Interaction	0.0094	Negligible	<0.0001	Yes
	Three-way	Job Algorithm	0.0003	Negligible	0.0768	No
		Worker Algorithm	0.0031	Negligible	<0.0001	Yes
		Job × Worker Interaction	0.0016	Negligible	<0.0001	Yes
		Worker × Workload Interaction	0.0073	Negligible	<0.0001	Yes

Appendix B. Algorithm Penalty Associations (Standardised Residuals)

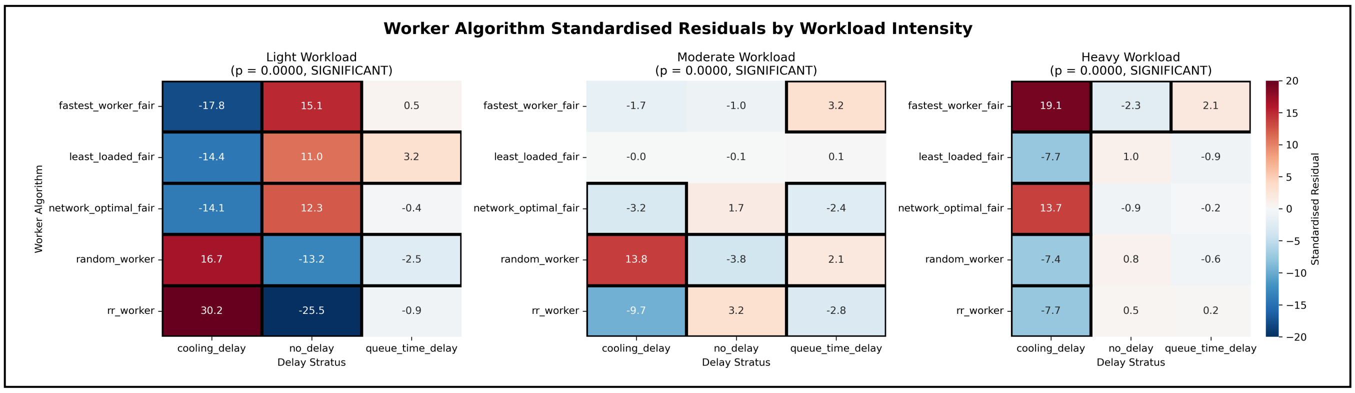

Fig. 2. Standardised Residuals: Worker Algorithm Performance Discovery

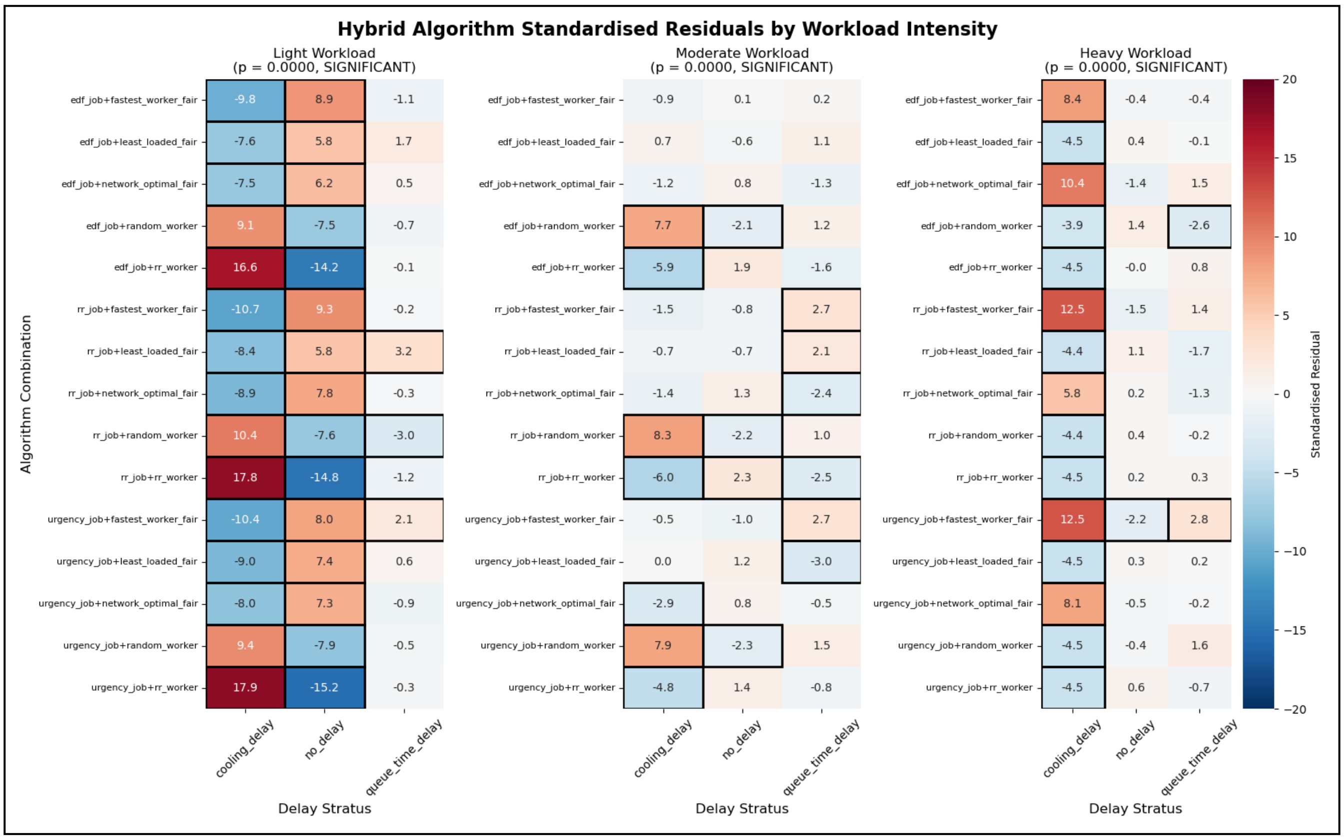

Fig. 3. Standardised Residuals: Hybrid Algorithm Consistency in Random and Round Robin Worker Assignments

Appendix C. Thermal Stability as Performance Differentiator

TABLE XI. WORKER ALGORITHM THERMAL PATTERNS: TRANSITION RATES AND STABILITY METRICS.

	Worker Algorithm	**Total Transitions**	**Degradation Rate HOT→COLD frequency**	**Recovery Rate COLD→HOT frequency**	**Average Stability Chain**	**Maximum Stability Chain**	**Thermal Efficiency**
Light	fastest_worker_fair	3,054	16.18%	15.59%	3.96	26.00	0.24
	least_loaded_fair	2,959	20.45%	19.91%	2.71	17.00	0.13
	network_optimal_fair	2,916	20.16%	18.96%	2.87	19.00	0.14
	random_worker	2,869	34.26%	31.37%	0.58	5.00	0.02
	rr_worker	2,954	21.87%	20.99%	1.25	14.00	0.06
Moderate	fastest_worker_fair	6,302	3.41%	3.44%	28.50	330.00	8.36
	least_loaded_fair	8,363	4.21%	3.87%	16.81	837.00	3.99
	network_optimal_fair	6,743	3.20%	3.07%	29.56	280.00	9.24
	random_worker	9,009	6.98%	6.95%	12.15	92.00	1.74
	rr_worker	8,920	1.89%	1.89%	50.78	279.00	26.87
Heavy	fastest_worker_fair	6,628	2.25%	2.22%	39.10	461.00	17.38
	least_loaded_fair	13,890	0.00%	0.00%	-	-	-
	network_optimal_fair	6,913	1.71%	1.69%	54.97	665.00	32.15
	random_worker	13,809	0.01%	0.01%	16.00	16.00	1600.00
	rr_worker	14,064	0.00%	0.00%	-	-	-

TABLE XII. HYBRID ALGORITHM THERMAL PATTERNS: TRANSITION RATES AND STABILITY METRICS.

	Hybrid Algorithm	Total Transitions	Degradation Rate	Recovery Rate	Average Stability Chain	Maximum Stability Chain	Thermal Effi-ciency
Light	edf_job+fastest_worker_fair	1,013.00	16.68%	16.68%	3.84	21.00	0.23
	edf_job+least_loaded_fair	979.00	20.94%	20.22%	2.86	26.00	0.14
	edf_job+network_optimal_fair	958.00	21.40%	19.94%	2.81	20.00	0.13
	edf_job+random_worker	945.00	34.71%	31.01%	0.55	6.00	0.02
	edf_job+rr_worker	982.00	22.51%	21.89%	0.49	7.00	0.02
	rr_job+fastest_worker_fair	1,010.00	15.15%	14.36%	4.43	32.00	0.29
	rr_job+least_loaded_fair	971.00	21.32%	20.60%	2.75	24.00	0.13
	rr_job+network_optimal_fair	1,012.00	19.27%	18.18%	3.33	19.00	0.17
	rr_job+random_worker	973.00	34.22%	30.94%	0.54	5.00	0.02
	rr_job+rr_worker	978.00	20.86%	20.25%	0.48	6.00	0.02
	urgency_job+fastest_worker_fair	1,031.00	16.68%	15.71%	4.20	21.00	0.25
	urgency_job+least_loaded_fair	1,009.00	19.13%	18.93%	3.20	24.00	0.17
	urgency_job+network_optimal_fair	946.00	19.87%	18.82%	3.01	26.00	0.15
	urgency_job+random_worker	951.00	33.86%	32.18%	0.58	5.00	0.02
	urgency_job+rr_worker	994.00	22.23%	20.82%	0.25	4.00	0.01
	edf_job+fastest_worker_fair	2,197.00	3.64%	3.69%	23.39	220.00	6.43
Moderate	edf_job+least_loaded_fair	2,756.00	4.32%	3.99%	9.59	129.00	2.22
	edf_job+network_optimal_fair	2,223.00	3.51%	3.42%	25.70	252.00	7.32
	edf_job+random_worker	2,979.00	6.78%	6.71%	12.51	62.00	1.85
	edf_job+rr_worker	2,917.00	1.78%	1.78%	52.06	251.00	29.25
	rr_job+fastest_worker_fair	2,047.00	3.22%	3.22%	29.83	323.00	9.26
	rr_job+least_loaded_fair	2,828.00	3.96%	3.71%	12.38	145.00	3.13
	rr_job+network_optimal_fair	2,275.00	3.21%	3.30%	25.75	184.00	8.02
	rr_job+random_worker	3,022.00	7.08%	7.08%	11.53	97.00	1.63
	rr_job+rr_worker	3,022.00	1.79%	1.79%	53.96	273.00	30.15
	urgency_job+fastest_worker_fair	2,058.00	3.35%	3.40%	23.06	212.00	6.88
	urgency_job+least_loaded_fair	2,779.00	4.35%	3.92%	9.93	83.00	2.28
	urgency_job+network_optimal_fair	2,245.00	2.90%	2.49%	36.68	391.00	12.65
	urgency_job+random_worker	3,008.00	7.08%	7.05%	11.88	59.00	1.68
	urgency_job+rr_worker	2,981.00	2.11%	2.11%	45.64	175.00	21.63

	Hybrid Algorithm	Total Transitions	Degradation Rate	Recovery Rate	Average Stability Chain	Maximum Stability Chain	Thermal Efficiency
	edf_job+fastest_worker_fair	2,487.00	1.81%	1.77%	40.10	220.00	22.15
	edf_job+least_loaded_fair	4,709.00	0.00%	0.00%	-	-	0.00
	edf_job+network_optimal_fair	1,837.00	2.23%	2.23%	31.49	143.00	14.12
	edf_job+random_worker	4,416.00	0.02%	0.05%	4.00	4.00	200.00
	edf_job+rr_worker	4,647.00	0.00%	0.00%	-	-	-
	rr_job+fastest_worker_fair	1,852.00	2.70%	2.65%	26.18	180.00	9.70
	rr_job+least_loaded_fair	4,456.00	0.00%	0.00%	-	-	0.00
	rr_job+network_optimal_fair	2,684.00	1.19%	1.27%	65.76	261.00	55.26
	rr_job+random_worker	4,641.00	0.00%	0.00%	-	-	-
	rr_job+rr_worker	4,664.00	0.00%	0.00%	-	-	-
	urgency_job+fastest_worker_fair	2,289.00	2.36%	2.36%	29.20	261.00	12.37
	urgency_job+least_loaded_fair	4,725.00	0.00%	0.00%	-	-	-
	urgency_job+network_optimal_fair	2,392.00	1.88%	1.76%	42.63	360.00	22.68
	urgency_job+random_worker	4,752.00	0.00%	0.00%	-	-	-
Heavy	urgency_job+rr_worker	4,753.00	0.00%	0.00%	-	-	-

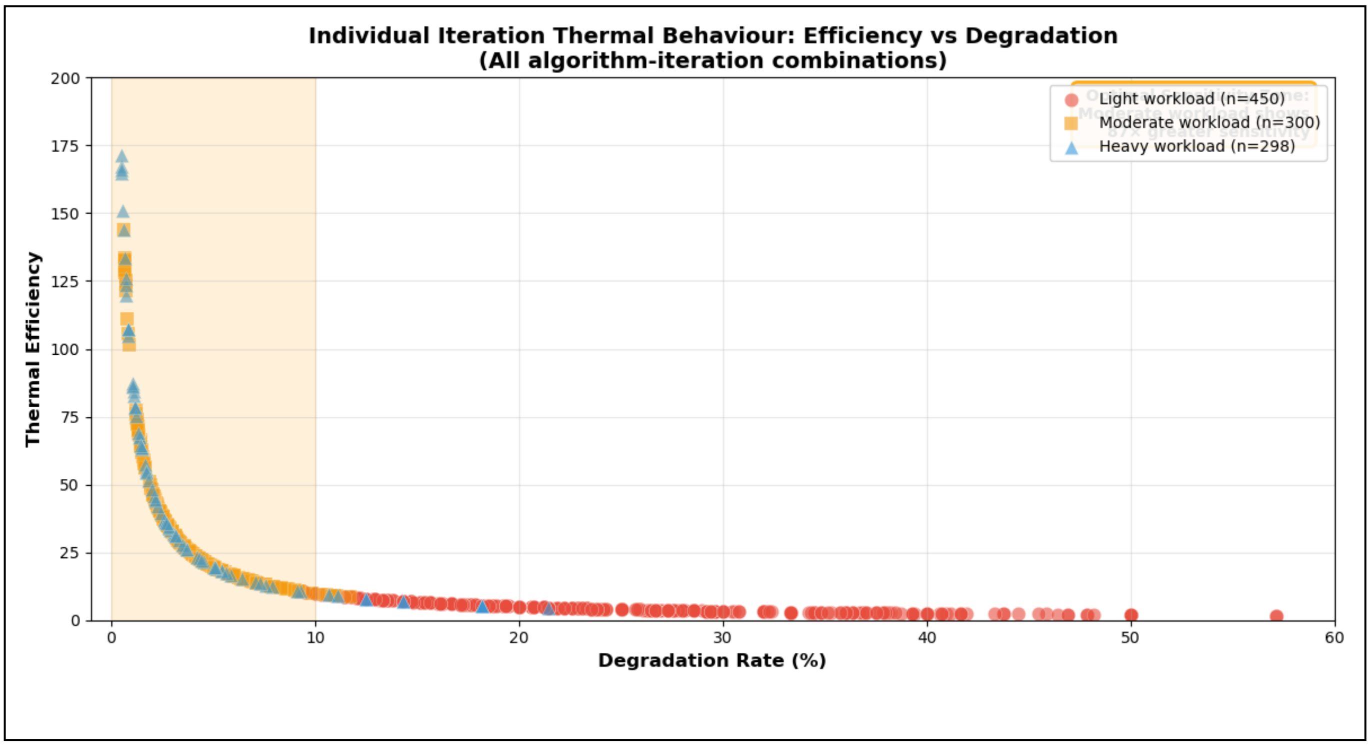

Fig. 4. Individual Container Thermal Efficiency Distribution across 1,048 algorithm-iteration combinations. The Optimal Sensitivity Zone (0-10% degradation, highlighted) shows where containers achieve maximum thermal efficiency (up to 200). Moderate workload iterations demonstrate greatest optimisation potential within this zone.

Appendix D. Cross Validation of Thermal Mechanism

1) Cross-Validation Approach

We employed two complementary analytical levels to validate the relationship between thermal efficiency and cooling penalties:

1. **Algorithm-Level Analysis**: Examines average performance across algorithms, where each data point represents one algorithm's mean metrics across all iterations. This approach reveals the expected performance characteristics of each scheduling strategy.
2. **Iteration-Level Analysis**: Examines individual iteration performance, treating each algorithm-iteration combination as an independent observation. This approach captures operational variability and validates pattern robustness.

TABLE XIII. CROSS-VALIDATION RESULTS AGAINST AGGREGATED ALGORITHM DATA AND GRANULAR ITERATION PERFORMANCE

Algorithm Type	**Workload**	**Algorithm-Level Analysis**				**Iteration-Level Analysis**			
		r	**p**	**sig.**	**n**	**r**	**p**	**sig.**	**n**
Worker	Light	-0.825	0.085	ns	5	-0.220	0.007	**	150
	Moderate	-0.798	0.106	ns	5	-0.276	0.006	**	100
	Heavy	-0.385	0.523	ns	5	-0.035	0.732	ns	100
Hybrid	Light	-0.892	0.000	***	15	-0.314	0.000	***	450
	Moderate	-0.770	0.001	***	15	-0.271	0.000	***	300
	Heavy	0.010	0.971	ns	15	-0.064	0.274	ns	298

2) Correlation Analysis

Pearson correlation coefficient (r) was calculated to measure the linear relationship between:

- *x* variable: Standardised residuals from chi-square analysis (Section B)
- *y* variable : Thermal efficiency ratio (stability/degradation) from transition analysis (Section C)

Statistical significance was assessed using two-tailed t-tests, with significance levels:

- ***: $p < 0.001$ (highly significant)
- **: $p < 0.01$ (very significant)
- *: $p < 0.05$ (significant)
- ns: $p \geq 0.05$ (not significant)

3) Algorithm-Level Cross Validation Scatter Plot

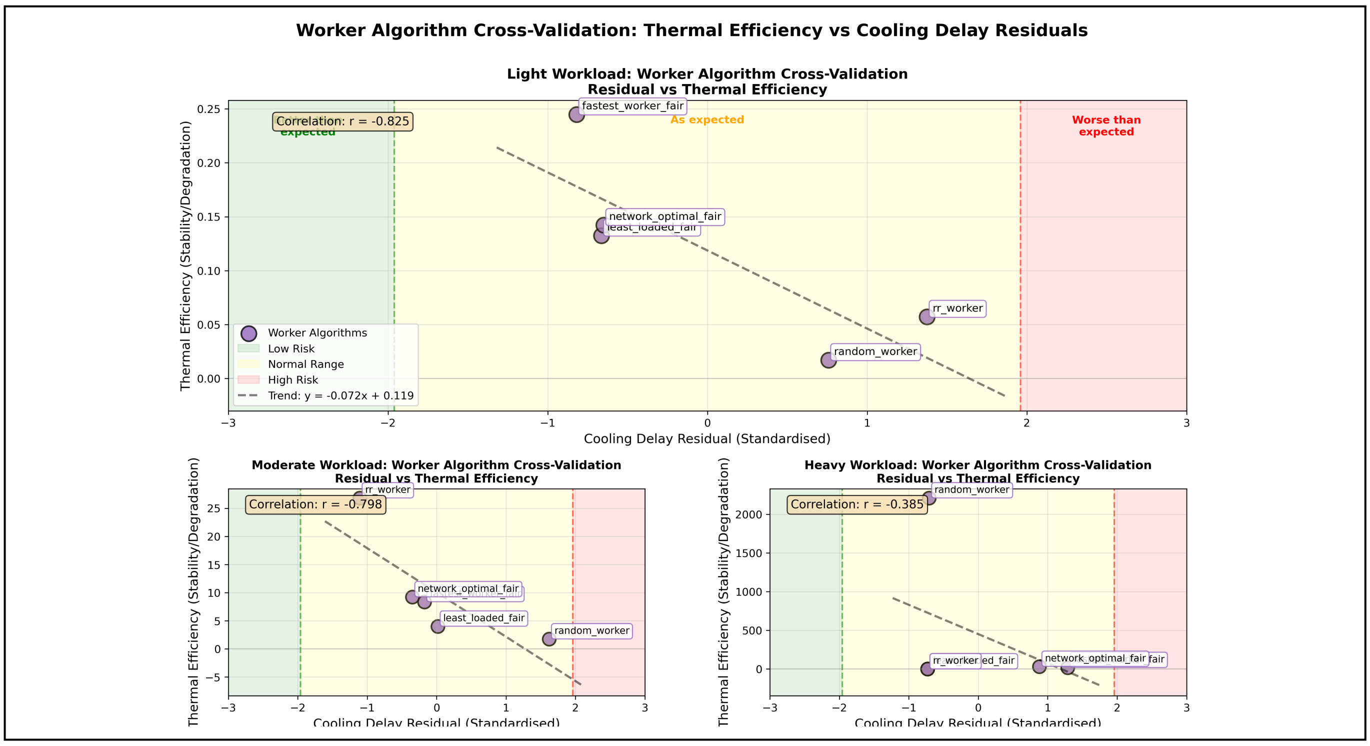

Fig. 5. Worker Algorithm Cross-Validation reveals the thermal efficiency sensitivity across workloads. Under moderate workload, algorithms show 109x greater slope sensitivity to cooling penalties compared to the light workload (-7.88 moderate vs -0.07 in light), suggesting this transition zone amplifies algorithmic differences before saturation eliminates them entirely.

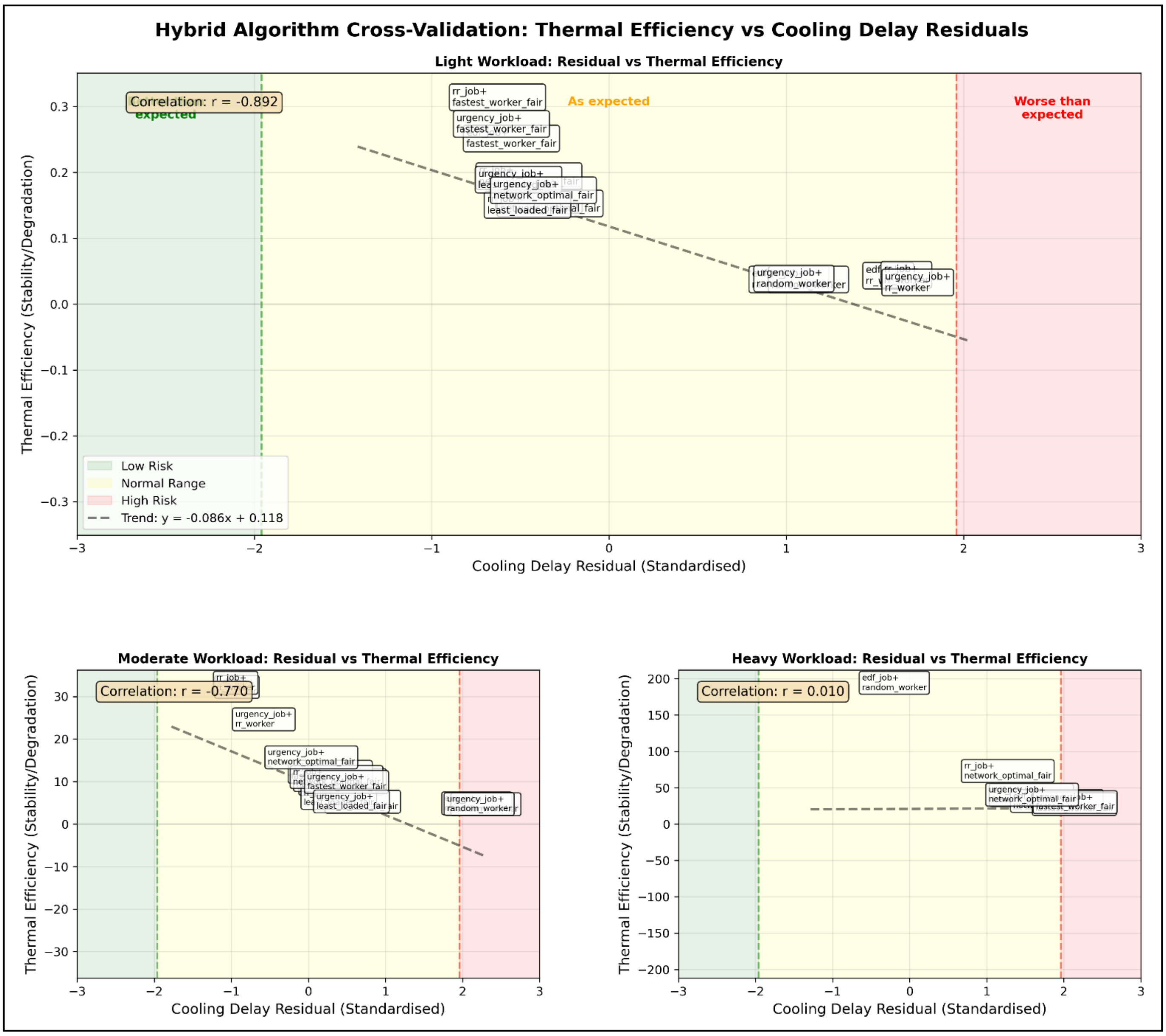

Fig. 6. Hybrid Algorithm Cross-Validation reveals amplified sensitivity under moderate workloads, with 87× greater responsiveness to cooling penalties compared to light workloads (slope: -7.49 vs -0.086). The varying intercepts across workload intensities (0.12 → 9.62 → 20.85) suggest non-linear saturation effects, where baseline thermal efficiency increases with load until system saturation eliminates algorithmic differentiation entirely.

TABLE XIV. TREND LINE EQUATIONS FOR CROSS-VALIDATION ANALYSIS

Analysis Type	Workload	Trend Line Equation	R^2	n
Worker Algorithm-Level	Light	y = -0.072x + 0.119	0.681	5
	Moderate	y = -7.8791x + 10.0237	0.637	5
	Heavy	y = -377.9926x + 451.8074	0.148	5
Hybrid Algorithm-Level	Light	y = -0.0855x + 0.1178	0.796	15
	Moderate	y = -7.4906x + 9.6236	0.593	15
	Heavy	y = 0.4700x + 20.8533	0.000	15

Appendix E. Performance Regime Identification

TABLE XV. PERFORMANCE REGIME METRICS SUMMARY

Algorithm Type	**Workload Intensity**	**Key Metrics from Experimental Analysis of Cooling Penalties**						
		Cramérs V	**Algorithm-Level Analysis**			**Iteration-Level Analysis**		
			r	**p**	**n**	**r**	**p**	**n**
Worker	Light	0.332	-0.825	0.085	5	-0.220	0.007	150
	Moderate	0.067	-0.798	0.106	5	-0.276	0.006	100
	Heavy	0.082	-0.385	0.523	5	-0.035	0.732	100
Hybrid	Light	0.333	-0.892	0.000	15	-0.314	0.000	450
	Moderate	0.070	-0.770	0.001	15	-0.271	0.000	300
	Heavy	0.085	0.010	0.971	15	-0.064	0.274	298[†]

[†]. Two algorithm-iterations excluded due to zero degradation events (perfect thermal stability)