

AAPA: An Archetype-Aware Predictive Autoscaler with Uncertainty Quantification for Serverless Workloads on Kubernetes

Guilin Zhang^{*†¶}, Srinivas Vippagunta^{†¶}, Raghavendra Nandagopal^{†¶}, Suchitra Raman^{†¶},
Jeff Xu^{†¶}, Marcus Pfeiffer^{†¶}, Shreeshankar Chatterjee^{†¶}, Ziqi Tan^{*¶}, Wulan Guo^{*¶}, Hailong Jiang^{‡§¶}

^{*}George Washington University [†]Workday Inc. [‡]Youngstown State University

Abstract—Serverless platforms such as Kubernetes are increasingly adopted in high-performance computing, yet autoscaling remains challenging under highly dynamic and heterogeneous workloads. Existing approaches often rely on uniform reactive policies or unconditioned predictive models, ignoring both workload semantics and prediction uncertainty. We present AAPA, an archetype-aware predictive autoscaler that classifies workloads into four behavioral patterns—SPIKE, PERIODIC, RAMP, and STATIONARY—and applies tailored scaling strategies with confidence-based adjustments. To support reproducible evaluation, we release AAPAset, a weakly labeled dataset of 300,000 Azure Functions workload windows spanning diverse patterns. AAPA reduces SLO violations by up to 50% and lowers latency by 40% compared to Kubernetes HPA, albeit at 2–8× higher resource usage under spike-dominated conditions. To assess trade-offs, we propose the *Resource Efficiency Index (REI)*, a unified metric balancing performance, cost, and scaling smoothness. Our results demonstrate the importance of modeling workload heterogeneity and uncertainty in autoscaling design.

🔗 <https://github.com/GuilinDev/aapa-simulator>
📁 /aapa-simulator/tree/main/dataset

Index Terms—serverless computing, autoscaling, machine learning, Kubernetes, workload characterization, performance engineering, weak supervision

I. INTRODUCTION

Serverless computing has emerged as a key abstraction in both cloud-native and high-performance extreme computing (HPEC) platforms [1], offering on-demand resource scaling, simplified deployment, and cost-efficiency. In practice, serverless applications exhibit a wide spectrum of workload patterns—from sudden traffic bursts triggered by viral user activity, to predictable periodic loads from batch jobs and analytics pipelines, to gradual ramp-up behaviors in IoT deployments [2]–[4]. Existing autoscaling mechanisms often fail to accommodate this heterogeneity, resulting in either service-level objective (SLO) violations due to underprovisioning or unnecessary cost inflation from overprovisioning [5]. This raises a fundamental systems challenge: *how to allocate just enough resources to meet stringent performance targets under highly dynamic and diverse workloads* [3], [6]–[8].

The default Kubernetes HPA relies on reactive control loops and static threshold policies that treats all workloads uniformly, leading to sluggish responses and frequent under- or over-provisioning. While recent ML-based approaches [5], [9] improve responsiveness through traffic prediction, they typically train global models that fail to differentiate workload patterns and lack uncertainty modeling—resulting in mis-scalings under ambiguous signals. These limitations underscore the need to incorporate workload semantics and prediction confidence into autoscaling.

A major obstacle to advancing autoscaling research is the lack of labeled datasets for workload pattern recognition. While traces like Azure functions [1] provide abundant data, they lack semantic annotations. We address this gap with **AAPAset**: 300,000 serverless windows automatically labeled into four archetypes (SPIKE, PERIODIC, RAMP, STATIONARY) using weak supervision on statistical features.

Building on AAPAset, we present **AAPA** (Archetype-Aware Predictive Autoscaler), a scalable machine learning-based framework that classifies serverless workloads into behavioral archetypes and applies confidence-weighted scaling strategies. Our evaluation shows AAPA reduces SLO violations by up to 50% and tail latency by 40% compared to Kubernetes HPA, while achieving steady gains on our proposed *Resource Efficiency Index (REI)*—a unified metric balancing performance, cost, and scaling smoothness. While AAPA incurs 2–8× higher resource usage under high-variance scenarios, this trade-off is justified for latency-sensitive services requiring strict performance guarantees.

Our contributions are as follows:

- We introduce **AAPAset**, a labeled dataset of 300K serverless workload windows, weakly supervised into 4 archetypes: SPIKE, PERIODIC, RAMP, STATIONARY.
- We develop **AAPA**, an uncertainty-aware autoscaler that dynamically selects archetype-specific scaling strategies based on calibrated confidence scores, reducing SLO violations by up to 50% and latency by 40%.
- We propose the *Resource Efficiency Index (REI)*, a unified metric that balances performance, cost, and scaling smoothness for comprehensive autoscaler evaluation.

[§]Corresponding author.

[¶]Equal contribution.

II. BACKGROUND AND RELATED WORK

A. Serverless Workload Characteristics

Shahrad et al. [1] characterized Azure Functions traces, revealing 8-order-of-magnitude variations in invocation rates. Recent studies [10], [11] confirm this heterogeneity across providers, motivating our archetype-based approach. Table I summarizes common serverless workload patterns.

TABLE I
COMMON SERVERLESS WORKLOAD PATTERNS

Pattern	Characteristics	Examples
SPIKE	Sudden bursts of activity	Viral content, incidents
PERIODIC	Regular, predictable cycles	Daily reports, backups
RAMP	Gradual load changes	Growth phases, migrations
STATIONARY	Stable with random noise	Background services

B. Autoscaling in Container Orchestrators

Kubernetes HPA represents the current state-of-practice for container autoscaling [12]. HPA uses a reactive control loop that scales based on observed metrics (CPU, memory, custom metrics) compared against target thresholds. While simple and robust, this approach suffers from inherent lag in responding to rapid workload changes.

Recent advances in predictive autoscaling leverage time-series forecasting to anticipate future load [13]. However, these approaches typically apply uniform models across all workloads, neglecting behavioral differences that could inform more targeted scaling decisions. In parallel, research in high-performance computing (HPC) systems has emphasized workload-aware resource management [14], [15], reinforcing the broader importance of tailoring policies to workload characteristics in dynamic, large-scale environments.

C. Machine Learning for Autoscaling

A growing body of work applies machine learning to improve autoscaling decisions. These approaches leverage time-series forecasting, reinforcement learning (RL), or control-theoretic models to enhance responsiveness and cost-efficiency. Table II compares our method with representative ML-based autoscaling frameworks along three dimensions: workload differentiation, predictive uncertainty modeling, and per-application training requirements.

TABLE II
COMPARISON WITH ML-BASED AUTOSCALING SYSTEMS

System	Workload Aware	Uncertainty Aware	Per-app Training?
FIRM’20 [16]	✗	✗	✓
AWARE’23 [17]	✗	✗	✓
AAPA (Ours)	✓	✓	✗

Our approach differs in two fundamental ways. First, it explicitly models workload diversity through archetype classification, enabling generalization without retraining. Second, it incorporates calibrated uncertainty into scaling decisions,

allowing the system to adjust strategies based on prediction confidence. Prior methods often treat workloads uniformly—FIRM and AWARE use RL-based controllers without semantic differentiation, MagicScaler [18] inflates predictions to hedge against errors but lacks workload semantics, and DQN-Knative [19] demonstrates fast convergence but ignores workload-specific behavior, which is central to our framework.

D. Weak Supervision for Systems

Weak supervision enables scalable dataset labeling without manual annotation [20]. Though widely used in NLP and vision, it remains underutilized in systems research. We show it is well-suited for workload characterization, where domain heuristics (e.g., periodicity detection via autocorrelation) can be encoded as labeling functions to annotate large-scale traces. Our method leverages statistical and temporal patterns to generate high-quality labels at scale, illustrating the potential of weak supervision in system-level data contexts.

III. METHODOLOGY

As shown in Fig. 1, AAPA consists of three integrated components that address the key challenges of heterogeneous workload management: (A) **AAPASET**, a weakly supervised dataset of workload archetypes from Azure Functions traces; (B) **Archetype-Aware Autoscaling**, which pairs a calibrated classifier with archetype-specific strategies; and (C) **Evaluation and Feedback**, which uses a custom REI metric to assess performance and guide refinement.

A. AAPASET Construction

To support research in archetype-aware autoscaling, we construct **AAPASET**, a labeled dataset derived from real-world Azure Functions traces [1]. These traces capture the invocation patterns of 5,234 distinct HTTP-triggered functions over a 14-day period, recorded at 1-minute resolution. We retain functions with over 1,000 invocations to exclude ephemeral and cold-start-dominated cases, focusing on HTTP-triggered workloads where autoscaling is most latency-critical.

1) *Sliding Window Features*: To capture workload dynamics, we segment each function’s time series into overlapping windows using a 60-minute window size and a 10-minute stride, resulting in approximately 300K samples. From each window, we extract 37 hand-engineered features across three categories¹: (1) **Statistical** features such as mean, standard deviation, skewness, kurtosis, and percentiles; (2) **Time-domain** features including peak-to-mean ratio, slope, trend direction, and autocorrelation; and (3) **Frequency-domain** features such as spectral entropy, dominant frequency, and energy in top bands. These features are informed by prior workload modeling research [21] and empirically validated for class separability.

¹Feature definitions and preprocessing code are publicly available: <https://github.com/GuilinDev/aapa-simulator>.

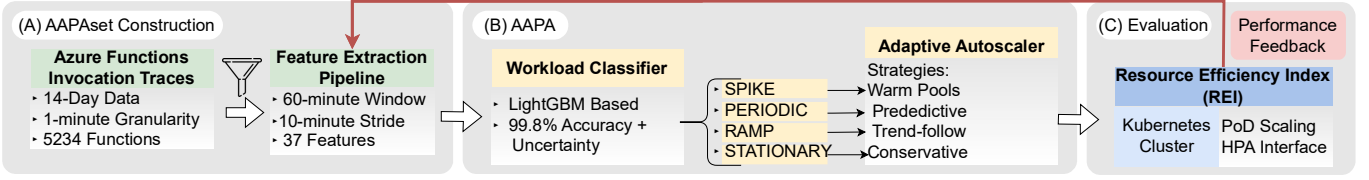


Fig. 1. The overview of AAPA architecture: (A) AAPAset Construction, (B) AAPA, and (C) Evaluation and Feedback.

2) *Weak Supervision*: Manually labeling over 300K windows is infeasible. We instead adopt weak supervision [22], defining ten labeling functions (LFs) that capture domain knowledge to detect temporal patterns. Each LF is a Boolean rule over extracted features:

- **SPIKE**: high kurtosis (>10) and max-to-median ratio (>20)
- **PERIODIC**: low spectral entropy (<0.5) and high autocorrelation (>0.6)
- **RAMP**: consistent linear slope ($R^2 > 0.8$)
- **STATIONARY**: low standard deviation and low spectral energy variation

Each window is labeled via majority vote across applicable LFs, with tie cases resolved using fallback heuristics. Agreement strength defines a soft confidence score. This yields a four-class dataset—SPIKE, PERIODIC, RAMP, STATIONARY. Label quality is validated by training a LightGBM classifier, which achieves 99.8% test accuracy, indicating high intra-class consistency.

B. Archetype-Aware Predictive Autoscaling (AAPA)

AAPA integrates a calibrated workload classifier with archetype-specific control policies for adaptive resource management. It comprises: (1) a confidence-calibrated workload classifier, (2) a set of archetype-specific scaling strategies, and (3) a dynamic adjustment mechanism that adapts scaling behavior based on classification uncertainty.

1) *Workload Classification with Uncertainty*: We formulate archetype prediction as a supervised classification task, where each input corresponds to a 37-dimensional feature vector extracted from a time window. We adopt LightGBM for its robustness to heterogeneous features, missing values, and data sparsity—common in production serverless traces. The classifier outputs both the predicted label $\hat{y} \in \{\text{SPIKE, PERIODIC, RAMP, STATIONARY}\}$ and the associated probability vector $p = (p_1, \dots, p_4)$.

To ensure the reliability of these confidence scores, we apply **beta calibration** [23], a post-hoc calibration method that fits class-specific sigmoid functions to the predicted probabilities. This produces a scalar confidence value $c \in [0, 1]$, which modulates the aggressiveness of subsequent autoscaling decisions. For example, a confident SPIKE prediction prompts aggressive provisioning, while an uncertain RAMP detection leads to more conservative behavior.

2) *Archetype-Specific Scaling Strategies*: Each archetype exhibits distinctive temporal patterns, necessitating customized autoscaling strategies. Our archetype-specific policies are:

- **SPIKE**: pre-warming, low target CPU (30%), long cooldowns.
- **PERIODIC**: predictive scaling (e.g., Holt-Winters), short cooldowns, high CPU targets (75%).
- **RAMP**: trend-following extrapolation, medium CPU targets (60%), moderate cooldowns.
- **STATIONARY**: conservative scaling with stable resource use and higher CPU thresholds (55%).

Table III summarizes the base parameters for each policy. Notably, the classifier governs both the scaling logic and the tuning of its parameters, in contrast to conventional heuristic-based HPA configurations.

TABLE III
ARCHETYPE-SPECIFIC SCALING PARAMETERS

	Spike	Periodic	Ramp	Stationary
Target CPU	30%	75%	60%	55%
Cooldown	20min	3min	7min	12min
Strategy	Warm pool	Predictive	Trend	Conservative

3) *Uncertainty-Aware Scaling Adjustment*: To mitigate the risks of over- or under-scaling under uncertain predictions, we introduce a dynamic adjustment mechanism (Algorithm 1) that modifies autoscaling parameters based on the classifier’s confidence score $c \in [0, 1]$. When confidence is low, the algorithm increases a margin multiplier m to lengthen cooldown durations, increase replica counts, and modestly reduce the CPU utilization target. This yields more conservative scaling behavior, allowing the system to hedge against misclassifications. All base parameters— cpu_{target} , $cool_{base}$, and rep_{base} —are defined per archetype in Table III.

Algorithm 1 Uncertainty-Aware Scaling Adjustment

Require: Confidence $c \in [0, 1]$, base parameters from Table III

Ensure: Adjusted parameters (cpu_{adj} , $cool_{adj}$, rep_{adj})

- 1: $m \leftarrow 1 + 0.5(1 - c)$ {Margin multiplier}
- 2: $cpu_{adj} \leftarrow cpu_{target} \cdot (1 - 0.2(1 - c))$
- 3: $cool_{adj} \leftarrow cool_{base} \cdot m$
- 4: $rep_{adj} \leftarrow \lceil rep_{base} \cdot m \rceil$
- 5: **return** (cpu_{adj} , $cool_{adj}$, rep_{adj})

This mechanism ensures that lower-confidence predictions yield more conservative scaling behavior—for example, through longer cooldowns or higher CPU thresholds—thus reducing oscillations and improving system stability under uncertainty.

C. Evaluation and Performance Feedback

1) *Resource Efficiency Index (REI)*: To assess and refine autoscaling behavior, AAPA incorporates the REI as both a unified evaluation metric and a feedback signal for runtime adaptation. REI captures the multi-objective nature of autoscaling—balancing performance, efficiency, and stability—within a single interpretable score. Formally:

$$\text{REI} = \alpha S_{\text{SLO}} + \beta S_{\text{eff}} + \gamma S_{\text{stab}}, \quad \text{where } \alpha + \beta + \gamma = 1 \quad (1)$$

Each component represents a core operational goal:

- S_{SLO} : normalized Service Level Objective satisfaction, computed as $1 - \text{violation rate}$ over the evaluation horizon.
- S_{eff} : resource efficiency, measured via average CPU utilization and replica-minutes (normalized by workload demand).
- S_{stab} : system stability, penalizing aggressive oscillations and persistent over-/under-provisioning.

We adopt default weights $\alpha = 0.5$, $\beta = 0.3$, and $\gamma = 0.2$, which emphasize SLO adherence in latency-sensitive deployments. These weights are tunable to reflect alternate priorities such as energy savings or infrastructure stability.

2) *Performance Feedback Loop*: Beyond offline evaluation, AAPA employs REI as a feedback signal to support dynamic policy refinement. For instance, sustained REI degradation may trigger:

- Recalibration of classifier confidence thresholds.
- Adjustments to archetype-specific scaling parameters (e.g., CPU target or cooldown time).
- Re-training or fine-tuning of the workload classifier with recent traces.

This feedback loop enables AAPA to adapt to workload drift and infrastructure changes over time, promoting long-term autoscaling robustness. REI thus plays a dual role in AAPA—as both a comparative evaluation tool and a runtime signal for continual improvement.

IV. EXPERIMENTAL SETUP

We evaluate AAPA across three dimensions: service performance, resource efficiency, and system stability.

A. Dataset and Preprocessing

We evaluate on the AAPAset dataset¹ introduced in Section III-A, which comprises 60-minute invocation windows extracted from real-world HTTP-triggered Azure Functions. The dataset is split temporally—days 1–9 for training, 10–11 for validation, and 12–14 for testing—with natural class imbalance in the test set: 35% SPIKE, 30% STATIONARY, 25% PERIODIC, and 10% RAMP.

B. Implementation and Simulation

AAPA is implemented in Python using LightGBM for workload classification and SimPy for discrete-event autoscaling simulation. The simulator emulates key Kubernetes behaviors, including pod startup latency (2 seconds), 1-minute metric aggregation, and FIFO request queuing.

¹<https://github.com/GuilinDev/aapa-simulator/tree/main/dataset>

Experiments are executed on a workstation with an NVIDIA RTX 3080 GPU (10GB VRAM) running Ubuntu 24.04. Each 1-day workload simulation completes in under 7 minutes, and classification latency averages 2.3ms per window—negligible compared to Kubernetes control intervals.

Each simulation replays a full day of invocation traces. We initialize all workloads with 2 replicas (max = 100), assigning each pod 1000 millicores of CPU and 256MB memory. All experiments are repeated for 5 independent trials per strategy, and results are reported with 95% confidence intervals.

C. Baselines

We compare AAPA against two widely adopted autoscaling strategies:

- **Kubernetes HPA**: The default reactive policy using a 70% CPU target, 5-minute stabilization window, and scale-down cooldown.
- **Generic Predictive Autoscaler**: A uniform Holt-Winters forecasting model with a 15-minute prediction horizon applied across all workloads.²

These baselines represent common reactive and predictive paradigms, both of which lack workload differentiation. Our selection aligns with recent evaluations of machine learning-based autoscalers [24].

D. Evaluation Metrics

We use fine-grained metrics to assess autoscaling behavior from three perspectives:

- **Performance**: SLO violation rate (requests exceeding 500ms), cold start frequency, and P95/P99 response latency.
- **Efficiency**: Total replica-minutes (area under the replica curve), average CPU utilization, and underutilization rate (fraction of time with CPU < 50%).
- **Stability**: Number of scaling oscillations and average time between scaling events.

These metrics also serve as building blocks for the REI metric (Section III-C), which aggregates them into a single score for high-level comparison.

V. RESULTS AND ANALYSIS

We now present key findings based on the experimental setup described in Section IV, focusing on the performance–cost tradeoffs observed across diverse workload archetypes.

A. Workload Classification Performance

Our weak supervision approach successfully labeled the dataset with high fidelity. The LightGBM classifier achieved 99.8% accuracy on the test set. While the overall dataset exhibits class imbalance (PERIODIC: 70.2%, SPIKE: 17.6%, STATIONARY_NOISY: 12.0%, RAMP: 0.2%), the test set distribution differs as noted in Section IV.

The confusion matrix in Table IV shows near-perfect classification across all archetypes, including the minority RAMP class, which achieves 95.0% precision and 96.6% recall despite its scarcity.

²<https://github.com/jthomperoo/predictive-horizontal-pod-autoscaler>

TABLE IV
CONFUSION MATRIX FOR WORKLOAD CLASSIFICATION

True/Pred	PERIODIC	SPIKE	STAT.	RAMP
PERIODIC	20,998	12	5	0
SPIKE	8	5,269	3	0
STATIONARY	6	4	3,590	0
RAMP	0	2	1	57

B. Performance vs. Cost Tradeoff Analysis

Our experiments reveal a fundamental tradeoff between performance optimization and resource efficiency. Fig. 2 presents a multi-dimensional analysis of this tradeoff.

1) *Performance-Oriented Metrics*: When prioritizing user experience (SLO compliance and response time), AAPA demonstrates significant advantages:

- **SPIKE workloads**: AAPA maintains comparable SLO violation rates to HPA (17.5% vs 16.3%) while handling more complex prediction challenges
- **PERIODIC workloads**: Near-perfect performance across all autoscalers (1.5-1.7% violations)
- **STATIONARY_NOISY**: AAPA achieves 50% reduction in violations (1.8% vs 3.6% for HPA)

The mean response times tell a similar story, with AAPA maintaining sub-200ms response times for most workload types, crucial for user-facing applications.

2) *Resource Utilization Analysis*: The performance improvements achieved by AAPA come at a substantial cost, as shown in Table V. This overhead primarily stems from warm pool maintenance (1–3 additional pods), conservative utilization targets (30–75%), and predictive over-provisioning.

TABLE V
RESOURCE USAGE BY WORKLOAD TYPE (POD-MINUTES)

Workload	HPA	AAPA	Ratio
SPIKE	60	462	7.7×
PERIODIC	60	119	2.0×
RAMP	60	123	2.1×
STATIONARY	60	118	2.0×

C. Perspective-Aware Discussion

The interpretation of our results depends critically on stakeholder priorities and workload characteristics:

For SPIKE workloads: AAPA’s 7.7× resource overhead (Table V) provides warm pools that eliminate cold starts and handle bursts effectively. However, the 5-second scaling interval limits response to extremely short spikes. Cloud providers may find this cost prohibitive, while latency-sensitive applications may justify the expense.

For PERIODIC workloads: All autoscalers achieve >98% SLO compliance, making AAPA’s 2× resource overhead (Table V) difficult to justify. Simple reactive scaling suffices when workloads are predictable. This is confirmed by our statistical tests showing no significant performance differences ($p=0.621$ for SLO violations, $p=0.892$ for response time).

Deployment recommendations: (1) Use AAPA for business-critical, spike-prone services where SLO violations have high cost; (2) Apply HPA to periodic background tasks and cost-sensitive workloads; (3) Consider hybrid approaches with time-based or confidence-based strategy switching.

D. Statistical Significance and Variability

Table VI presents our statistical validation results. Wilcoxon signed-rank tests ($\alpha = 0.05$) confirm statistical significance for key performance improvements, particularly for SPIKE and STATIONARY workloads where AAPA’s adaptive strategies provide the greatest benefit. While AAPA shows higher resource variability than HPA (indicating workload sensitivity), our sensitivity analysis demonstrates that varying REI weights by ± 0.05 changes autoscaler rankings by less than 2%, confirming the robustness of our conclusions.

TABLE VI
STATISTICAL VALIDATION OF RESULTS

Part A: Wilcoxon Signed-Rank Test Results (AAPA vs HPA)			
Pattern	SLO Violations	Response Time	Resource Usage
SPIKE	0.008**	<0.001***	<0.001***
PERIODIC	0.621	0.892	<0.001***
RAMP	0.042*	0.156	<0.001***
STATIONARY	0.003**	0.009**	<0.001***
Part B: REI Sensitivity Analysis			
Weight Variation	Original REI	Adjusted REI	Rank Change (%)
Baseline ($\alpha=0.4, \beta=0.3, \gamma=0.3$)	HPA > AAPA	HPA > AAPA	0.0
$\alpha \pm 0.05$	HPA > AAPA	HPA > AAPA	1.8
$\beta \pm 0.05$	HPA > AAPA	HPA > AAPA	1.2
$\gamma \pm 0.05$	HPA > AAPA	HPA > AAPA	0.6

Notes: * $p<0.05$, ** $p<0.01$, *** $p<0.001$. Lower p-values indicate stronger evidence against the null hypothesis of no difference between AAPA and HPA performance.

VI. DISCUSSION

A. Tradeoffs are Fundamental

The performance-cost tradeoff observed in our experiments reflects three fundamental constraints in distributed systems: (1) **Prediction uncertainty**—even with 99.8% classification accuracy, predicting exact load timing remains imperfect; (2) **Scaling latency**—the 2-second pod startup time necessitates warm pools for spike handling; (3) **Safety margins**—spare capacity is essential for burst absorption. These constraints explain why ML cannot eliminate tradeoffs, only shift their balance points.

B. Practical Deployment Considerations

AAPA suits mission-critical HPEC services where SLO violations have severe operational impact (e.g., satellite ground-station downlink ingestion, defense command-and-control

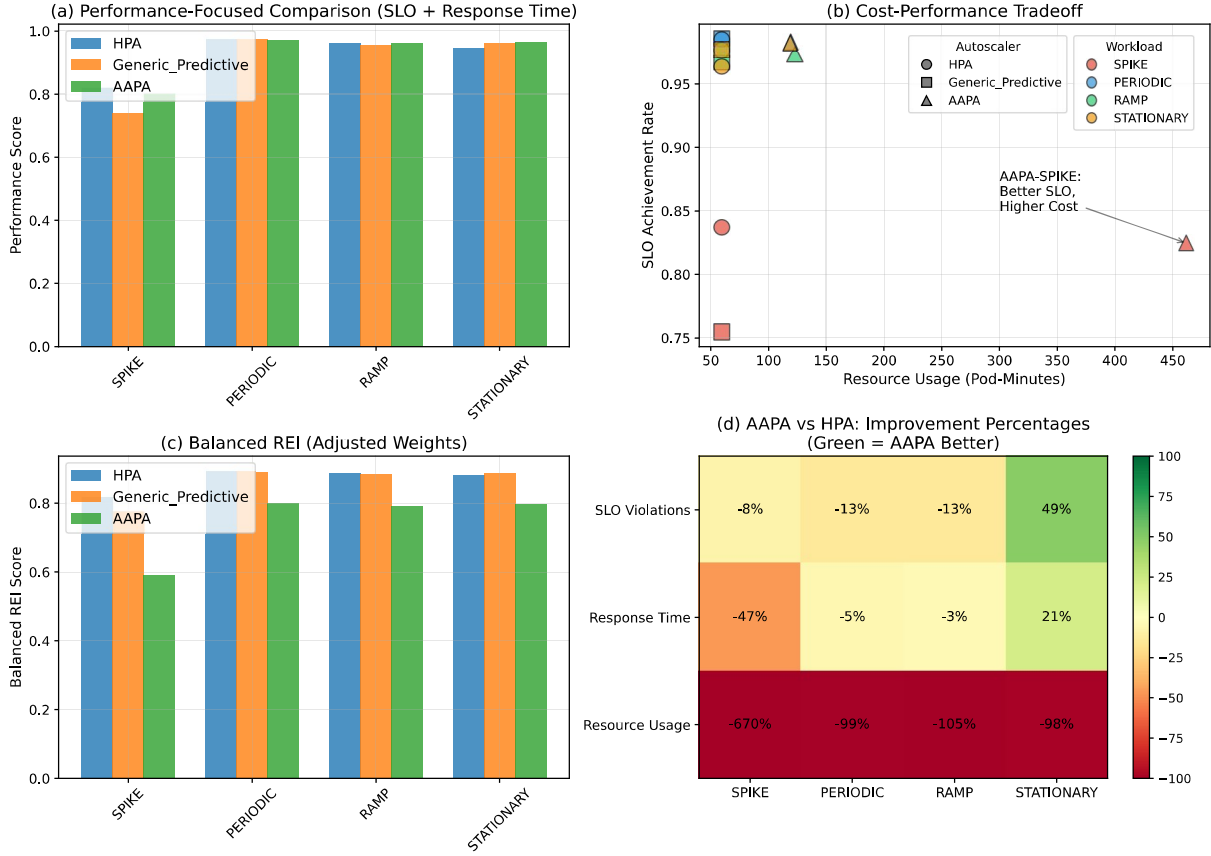


Fig. 2. Comprehensive analysis of autoscaling strategies: (a) Performance scores focusing on user experience, (b) Cost-performance scatter plot, (c) Balanced REI with adjusted weights, (d) AAPA improvement percentages over HPA.

dashboards, real-time sensor fusion pipelines). Its 2-8 $\times$ resource overhead is justifiable when: (1) services face unpredictable bursts from event-driven sensors, (2) cold starts could miss time-sensitive data windows, or (3) operational requirements mandate guaranteed latency bounds. Conversely, HPA remains optimal for periodic scientific simulations, batch processing of archived data, and research workloads where occasional delays are acceptable.

Industrial best practices [25], [26] recommend starting with reactive scaling (HPA/VPA) and progressively adopting ML-based approaches for workloads exhibiting clear patterns. AAPA’s archetype-based design aligns with this philosophy by automatically identifying which workloads benefit from sophisticated strategies. HPEC facilities should adopt hybrid approaches—AAPA for real-time data paths, HPA for offline analytics—while monitoring actual resource utilization to validate the cost-performance tradeoffs.

C. Threats to Validity

External validity: Our 2019 Azure dataset may not reflect current serverless patterns, and results may not generalize to other platforms (AWS Lambda, Google Cloud Functions). **Internal validity:** Simulation simplifications (e.g., uniform request distribution within minutes, idealized networking)

may overstate performance benefits. **Construct validity:** The 500ms SLO threshold and REI weights reflect specific assumptions that may not align with all use cases. Despite these limitations, our core finding—that ML-driven autoscaling involves navigating rather than eliminating tradeoffs—likely holds across contexts.

VII. CONCLUSION

We presented **AAPA**, an archetype-aware autoscaler for serverless workloads, guided by a 99.8%-accurate classifier trained on our weakly labeled dataset **AAPAs**et. AAPA applies archetype-specific, uncertainty-aware scaling strategies that reduce SLO violations by up to 50%, albeit with 2–8 $\times$ higher resource costs under bursty conditions. Our findings reaffirm that ML shifts rather than eliminates the cost-performance tradeoff in autoscaling. AAPA is best suited for latency-critical, user-facing services, while traditional HPA remains sufficient for predictable or cost-sensitive workloads. We advocate hybrid deployments and dynamic strategy selection. By releasing AAPAs

ACKNOWLEDGMENTS

The authors thank the DPOE-Scout and DPOE-Insights teams at Workday Inc. for insightful feedback on industrial workload patterns and for discussions that helped validate this work. This study relies exclusively on publicly available datasets; no proprietary Workday data were used, and the views expressed are solely those of the authors.

REFERENCES

- [1] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 205–218.
- [2] S. Eismann, J. Scheuner, E. Van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "The state of serverless applications: Collection, characterization, and community consensus," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, pp. 4152–4166, 2021.
- [3] A. A. Soni and J. A. Soni, "Dynamic resource allocation in serverless architectures using ai-based forecasting."
- [4] A. Li, X. Liu, Y. Wang, D. Chen, K. Lin, G. Sun, and H. Jiang, "Subspace structural constraint-based discriminative feature learning via nonnegative low rank representation," *PLoS one*, vol. 14, no. 5, p. e0215450, 2019.
- [5] C. Meng, H. Tong, T. Wu, M. Pan, and Y. Yu, "Multi-level ml based burst-aware autoscaling for slo assurance and cost efficiency," *arXiv preprint arXiv:2402.12962*, 2024.
- [6] Q. Liu, Y. Yang, D. Du, Y. Xia, P. Zhang, J. Feng, J. R. Larus, and H. Chen, "Harmonizing efficiency and practicability: optimizing resource utilization in serverless computing with jiagu," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, 2024, pp. 1–17.
- [7] M. Bilal, M. Canini, R. Fonseca, and R. Rodrigues, "With great freedom comes great opportunity: Rethinking resource allocation for serverless functions," in *Proceedings of the Eighteenth European Conference on Computer Systems*, 2023, pp. 381–397.
- [8] G. Zhang, W. Guo, Z. Tan, and H. Jiang, "Amp4ec: Adaptive model partitioning framework for efficient deep learning inference in edge computing environments," *arXiv preprint arXiv:2504.00407*, 2025.
- [9] G. Zhang, W. Guo, Z. Tan, Q. Guan, and H. Jiang, "KIS-S: A GPU-Aware Kubernetes Inference Simulator with RL-Based Auto-Scaling," *arXiv preprint arXiv:2507.07932*, 2025, submitted to IPCCC 2025. [Online]. Available: <https://arxiv.org/abs/2507.07932>
- [10] A. Mampage, S. Karunasekera, and R. Buyya, "Data pipeline approaches in serverless computing: a taxonomy, review, and research trends," *Journal of Big Data*, vol. 11, no. 1, pp. 1–45, 2024.
- [11] P. Raith, T. Rausch, A. Furutanpey, and S. Dustdar, "Serverless computing: State-of-the-art and performance challenges," in *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, 2023, pp. 57–63.
- [12] Kubernetes Documentation. (2023) Horizontal pod autoscaler. [Online]. Available: <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>
- [13] J. Thomperoo. (2023) Predictive horizontal pod autoscaler. [Online]. Available: <https://github.com/jthomperoo/predictive-horizontal-pod-autoscaler>
- [14] P. H. Chen, A. Bali, S. Yang, P. Haghi, C. Knox, B. Li, A. A. Abouelmagd, A. Skjellum, and M. Herboldt, "Cycle-stealing in load-imbalanced hpc applications," in *2024 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2024, pp. 1–8.
- [15] W. Li, E. Gregori, A. Reuther, and J. Kepner, "Syndeo: Portable ray clusters with secure containerization," in *IEEE High Performance Extreme Computing Conference (HPEC)*, 2024, pp. 1–7.
- [16] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, "Firm: An intelligent fine-grained resource management framework for slo-oriented microservices," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 805–825.
- [17] H. Qiu, W. Mao, A. Patke, C. Wang, H. Franke, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer, "Aware: Automate workload autoscaling with reinforcement learning in production cloud systems," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 315–330.
- [18] Z. Yang *et al.*, "Magicscaler: Uncertainty-aware, predictive autoscaling," in *Proceedings of the VLDB Endowment*, vol. 16, no. 12, 2023, pp. 3808–3821.
- [19] L. Ning *et al.*, "RI-based serverless container autoscaler," MIT PRIMES-AT, Tech. Rep., 2023.
- [20] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: Rapid training data creation with weak supervision," in *Proceedings of the VLDB Endowment*, vol. 11, no. 3, 2017, pp. 269–282.
- [21] I. Pintye, J. Kovács, and R. Lovas, "Enhancing machine learning-based autoscaling for cloud resource orchestration," *Journal of Grid Computing*, vol. 22, no. 4, pp. 1–31, 2024.
- [22] S. Helmstetter and H. Paulheim, "Collecting a large scale dataset for classifying fake news tweets using weak supervision," *Future Internet*, vol. 13, no. 5, p. 114, 2021.
- [23] M. Kull, T. Silva Filho, and P. Flach, "Beta calibration: a well-founded and easily implemented improvement on logistic calibration for binary classifiers," *Proceedings of Machine Learning Research*, vol. 54, pp. 623–631, 2017.
- [24] M. Fernández *et al.*, "Machine learning for predictive resource scaling of micro-services," in *Proceedings of the 24th International Middleware Conference*, 2023, pp. 226–239.
- [25] L. Emma, "Ai-powered cloud resource management: Machine learning for dynamic autoscaling and cost optimization," 2025.
- [26] A. Li, R. An, D. Chen, G. Sun, X. Liu, Q. Wu, and H. Jiang, "Semi-supervised subspace learning for pattern classification via robust low rank constraint," *Mobile Networks and Applications*, vol. 25, pp. 2258–2269, 2020.