INTERNATIONAL JOURNAL OF INFORMATION AND COMMUNICATION TECHNOLOGY TRENDS

Vol. 6 No. 02 (2026)

e.ISSN: 2409-5222

https://ijictt.ictt.info

Phase-Aware Dual-Signal Allocation for Predictive Cold-Start Mitigation in Serverless Computing

James Lee *, Jia Martin

Department of Computer Science, Boston University, Boston, MA, United States

***Email**: j.lee79@bu.edu*

Abstract

Warm pools mitigate Function-as-a-Service cold starts, but equal treatment of warm hits ignores heterogeneity in startup cost. We present Phase-Aware Dual-Signal Allocation (PADA), a lightweight, causal scheduler that fuses exponential recent-activity smoothing with previous-day seasonal activity and ranks candidate warm instances by predicted miss cost per reserved megabyte. A three-phase startup decomposition makes the scheduling objective sensitive to infrastructure, runtime, and application initialization. We conduct a reproducible minute-granularity evaluation on six independently generated synthetic workload fleets comprising 144 functions each, using three memory budgets, seven comparators or ablations plus a reactive reference, and a separate readiness-lag and retention-stability stress test. At an 8,192-MB cap, PADA achieves 43.22 modeled cold-start milliseconds per request, versus 89.31 for a fixed 15-minute retention rule and 68.12 for an EMA-only allocator. The comparison also shows that minimizing the number of cold starts can differ from minimizing their latency penalty, and that a raw histogram-hazard feature can be detrimental under drift. In a separate readiness-lag test, a one-minute lag yields 44.00 versus 90.02 ms/request for PADA and Fixed15. A retention-biased extension reduces planned slot activation events by 55.1% with minimal change in modeled latency. All reported arrivals are synthetic and startup costs are assumed, rather than measured cloud-service latency.

Keywords *:serverless computing, cold start, warm pool, workload prediction, scheduling, resource allocation, trace simulation.*

1. Introduction

Function-as-a-Service (FaaS) exposes elastic event execution while hiding server provisioning from application developers. This abstraction lowers deployment friction but transfers the problem of fast provisioning to the platform [1]. Empirical collections show that production serverless applications vary widely in architecture and resource demand [2]. Particularly important is temporal heterogeneity: the Azure study by Shahrad et al. identified extremely diverse invocation frequencies and used these observations to design invocation-aware warm-instance management [3]. A cold invocation traverses infrastructure provisioning, language-runtime initialization, and application startup. Chen and Ding [4] organize these three latency components and contrast keep-alive, pre-warming, snapshotting, and runtime-level mitigation.

Their decomposition makes an important allocation distinction visible: one avoided cold start need not save the same latency as another. A warm instance allocated to a heavyweight runtime may prevent substantially more delay than an equally expensive instance serving a lightweight runtime. Reviews of cold-start mechanisms emphasize the conflicting requirements of latency, responsiveness, and resource efficiency [5], while platform-level surveys identify placement and resource allocation as persistent open problems [6]. Yet a common experimental shortcut is to measure warm-hit probability without differentiating the latency cost of the misses. Optimizing that surrogate can favor frequently invoked but inexpensive runtimes while a small number of expensive startups dominate user-visible penalties. This paper studies the narrower and testable question of how to allocate a capped pool of warm function slots when both periodic and rapidly changing demand patterns occur. We propose Phase-Aware Dual-Signal Allocation (PADA), which ranks functions using a causal combination of short-term exponential smoothing and previous-day seasonal activity, weighted by their three-phase cold-start costs per unit of reserved memory. Its design requires neither a fitted deep model nor an assumption that an invocation histogram is stable throughout the evaluation horizon.

Our contributions are: (i) a phase-cost-weighted, memory-normalized prediction priority that distinguishes reducing the count of cold starts from reducing their modeled latency; (ii) an auditable minute-granularity simulator and seven alternative policies or ablations, including an implemented adaptive inter-arrival histogram window; and (iii) experiments across six independent synthetic workload realizations and three resource caps, together with paired uncertainty estimates. These results establish behavior under a controlled model; they are not presented as measurements from a deployed FaaS service.

II. RELATED WORK

Warm-pool management is commonly treated as a temporal prediction or cache replacement problem. FaasCache adopts a greedy-dual formulation to manage warm instances, with a keep-alive priority that combines invocation frequency, initialization cost and memory size [7], whereas Vahidinia et al. train an adaptive policy using reinforcement learning [8]. Fifer integrates function-aware scaling and batching for resource utilization in serverless chains [9]. A broader resource-management taxonomy places scheduling, scaling, and resource allocation in a unified systems context [10]. PADA shares the cost-per-memory intuition of greedy-dual keep-alive but ranks functions by a predicted activation probability that fuses recent and previous-day activity, weighted by a phase-decomposed miss cost; it targets low-overhead, per-minute selection under an explicit memory cap, and its interpretable signal isolates the effect of phase-cost weighting. The PRESAGE framework illustrates how demand prediction and its uncertainty can inform resource scheduling in bursty cloud workloads [27]. Complementary approaches reduce the cost of the miss itself. Firecracker designs a lightweight microVM boundary [11]; SEUSS caches shared unikernel snapshot state [12]; and Ustiugov et al. analyze serverless snapshot restoration [13]. DADI reduces container startup barriers through on-demand block image loading [14], while Faasm uses WebAssembly-backed isolation and state sharing [15]. These systems motivate per-runtime miss-cost heterogeneity, but none of their platform components is required for the simulator evaluated here. Predictive container warming is not sufficient for every application. Sui et al. isolate the value of opportunistic model and dependency preloading for serverless inference [16]. Bermbach et al. exploit application-level orchestration knowledge to pre-trigger containers [17], and Eismann et al. analyze prediction of workflow execution costs [18]. At a different level, Kaffes et al. advocate global core-granular scheduling [19]. Unlike workflow knowledge and runtime modification, our scheduler consumes only bucketized invocation history, memory footprints, and predefined initialization-cost profiles. Neuro-Elastic demonstrates the value of evaluating adaptive computing decisions against concrete device-state and latency constraints, offering a complementary perspective on resource-aware execution [28].

Edge deployments increase the importance of constrained warm pools [20]. WebAssembly-based edge runtimes [21], tinyFaaS [22], and comparative studies of container and microVM boundaries [23] illustrate how platform characteristics affect startup costs. FaaSCtrl argues for multi-dimensional latency targets [24].

Recent taxonomies continue to identify prediction as an important cold-start research direction [25], while resource precrafting has also been proposed to shorten the startup path [26]. These works collectively underline the necessity of making the experimental boundary precise: PADA addresses warm-slot decisions, not the execution environment or container-creation mechanism.

III. SYSTEM MODEL AND OBJECTIVE

Time is partitioned into one-minute intervals t. For function i, the nonnegative integer n(i,t) is its invocation count in that interval and a(i,t)=1[n(i,t)>0] indicates an active function-minute. Each function has a single potentially reusable slot requiring m(i) MB. The scheduling decision x(i,t) is binary, selected at the beginning of minute t from observations strictly preceding t. At most one cold initialization is counted for a nonempty bucket, since further invocations in that bucket are assumed to reuse the instance. Queuing, concurrent execution, and multiple simultaneous instances are outside this abstraction.

We implement the three-stage startup decomposition by assigning every function an infrastructure component dI(i), a runtime component dR(i), and an application component dA(i). Their sum is the modeled cold penalty D(i); assigning different tuples across functions makes the value of a protected slot heterogeneous.

$$D(i) = dI(i) + dR(i) + dA(i). \quad (1)$$

An unwarmed active bucket contributes c(i,t)=a(i,t)[1-x(i,t)] cold starts. Given a memory cap B, the objective is to minimize accumulated modeled cold-start delay. Normalizing by total invocations yields an interpretable latency contribution, but the scheduler does not observe any current-interval invocation when selecting x.

$$\min \ \Sigma t \, \Sigma i \, c(i,t)D(i), \ \text{s.t.} \ \Sigma i \, m(i)x(i,t) \le B. \quad (2)$$

Our primary metric is $L = \Sigma t \Sigma i \, c(i,t)D(i) / \Sigma t \Sigma i \, n(i,t)$, in milliseconds per request. We also report the cold fraction of active function-minutes, Σc/Σa, and average reserved MB per interval. These metrics can disagree: protecting costly Java-like startups may lower L while leaving more lightweight function-minutes cold. Optimization targets L because it incorporates the asymmetric cost of misses.

In this model, a candidate reservation is valuable only when the function becomes active in the upcoming minute. Let q(i,t) denote a causal estimate of that event. The one-step conditional expected cold delay is $\Sigma i \, [1-x(i,t)]q(i,t)D(i)$. With identical memory footprints, sorting functions by qD and selecting as many as the budget permits is an exact minimizer of this one-step expectation: exchanging a selected item with lower qD for an unselected higher-value item cannot increase the objective. With heterogeneous memory footprints, maximizing ΣxqD is a 0–1 knapsack problem. PADA uses qD/m as a computationally inexpensive benefit-density approximation to this problem, thereby retaining predictable scheduling overhead.

This formulation distinguishes total reserved memory from billed execution memory. The instantaneous resource constraint is enforced on every simulated minute, regardless of whether a selected function is invoked. We charge a slot as reserved for that entire interval and report mean occupancy, although the performance objective in (2) is latency-oriented rather than a combined dollar cost. A provider could introduce an explicit time-varying memory price or a minimum expected gain per MB; doing so would change the optimization objective and is not silently assumed here. Only the first cold invocation of a function-minute incurs D(i); additional arrivals within that interval are assumed to reuse the newly initialized environment. Consequently, a burst with multiple arrivals in the same minute increases the denominator of the delay-per-request metric but cannot produce more than one counted initialization. This is a conservative abstraction for serial reuse, and can be optimistic if the real workload requires multiple concurrent instances. It also motivates reporting the active-bucket cold fraction separately from the per-request delay.

IV. PHASE-AWARE DUAL-SIGNAL ALLOCATION

For short-term demand, PADA maintains an exponential moving indicator $e(i,t)=0.88e(i,t-1)+0.12a(i,t-1)$. For recurring demand, it computes s(i,t) from previously completed days only: each day's binary activity vector is smoothed with a 31-minute circular moving average, and the most recent one to three completed

days are averaged at the same minute of day. The method does not inspect later minutes of the current day. The two estimates are fused with fixed coefficients, selected before the experimental runs.

$$p(i,t) = 0.47\, e(i,t) + 0.53\, s(i,t). \quad (3)$$

The short-term component reacts to bursts, while the previous-day component can reserve capacity before recurring peaks. PADA assigns a benefit density $v(i,t)=p(i,t)D(i)/m(i)$. It sorts candidate functions by descending density and greedily reserves each feasible slot until the cap is exhausted. This is a budget-feasible ranking heuristic, not a proof of optimal 0–1 knapsack selection. For F functions, sorting costs O(F log F) per decision interval; there is no offline model fitting or gradient-based training. The EMA update is evaluated in a strict predict-then-update sequence. At the boundary of interval t, e(i,t) is already fixed by a(i,t−1) and earlier observations; the current activity a(i,t) is not inserted until after the reservation decision and evaluation. The seasonal estimate is also historical despite referring to the same wall-clock minute: an input from day d−1 is available before the corresponding minute of day d. In contrast, using a centered smoother over the current day would leak information from future minutes and produce an unfair forecast. The implementation stores the earlier daily vectors explicitly and never performs such smoothing over the evaluation day. PADA accommodates cold functions with little historical activity without manufacturing invocation forecasts. Early in the trace, the seasonal component is zero, so short-term activity governs decisions. After a completed day is observed, the previous-day component becomes available for each minute-of-day. A sparse function without recent activity receives a low score and is unlikely to occupy a scarce slot unless its modeled cold-start cost is unusually high. This is a resource-management choice rather than an admission of a new function to a production system; a truly unseen function's first invocation is not predicted. The algorithm has three independent inputs: the invocation count matrix (used to form causal activity indicators), a function memory vector, and a three-component cold-penalty vector. The first is forecast-related; the second enforces the cap; and the third establishes which misses are expensive. Separating these inputs allows controlled ablation without changing the arrival process. Removing phase weighting sets every D(i) to the same ranking constant; disabling seasonality replaces (3) with EMA alone.

$$v(i,t) = p(i,t)\,[dI(i)+dR(i)+dA(i)] / m(i). \quad (4)$$

The expected one-step miss cost under a perfectly calibrated p would be pD. We use this only as a ranking proxy: neither component is claimed to produce calibrated probabilities. All methods in the comparison use the same memory feasibility check, bucket-level cold-start semantics, function memory vector, and startup-cost vector. In particular, changing a weight changes resource allocation rather than the synthetic arrivals against which it is evaluated. We additionally implement the hybrid inter-arrival-histogram keep-alive policy of Shahrad et al. [3], one of the adaptive keep-alive mechanisms analyzed by Chen and Ding [4]. Every function maintains an online frequency table of gaps between nonempty invocation buckets. With at least 12 observed gaps, a candidate warm window spans an empirical lower fifth-percentile boundary (minus one minute) to the 99th percentile; earlier history uses a 15-minute retention fallback. Quantiles refresh hourly, gaps beyond 360 minutes are aggregated in a terminal bin, and admitted candidates are ranked by inverse memory. A variant adds a conditional histogram-hazard term to the dual-signal estimate. This isolates whether the additional gap statistic improves resource decisions in our nonstationary workload model.

V. EXPERIMENTAL DESIGN

A. Data provenance and generation. We use a transparent synthetic generator whose temporal classes and drift are known by construction. Its minute-level format mirrors the published Azure Functions trace, which records per-minute invocations by anonymized function over 14 days [3], but no Azure observations enter our results. Each generated realization has 144 functions and seven complete days. The four function classes are equally represented: steady functions with weak diurnal variation; sparse low-rate functions; periodic functions with localized daily peaks; and bursty functions with episodic high-rate intervals. Some periodic functions shift their peak time after day four. Invocation counts are independently sampled from Poisson processes conditional on the generated time-varying intensity; temporal bursts are introduced through shared

high-intensity intervals, rather than by claiming empirical cloud burst statistics. For steady-class functions, a function-specific base arrival intensity is drawn once and modulated by a weak daily harmonic. Sparse-class functions have much lower base intensity but similar mild temporal modulation. A periodic-class function receives an independently sampled center minute and peak width; its rate rises as a Gaussian-shaped function of circular distance to that center. With fixed probability, the center shifts after day four to introduce schedule drift. Bursty functions receive two to five randomly positioned elevated-intensity intervals per day, each lasting several minutes, superimposed on a weak background rate. Conditional on those rate schedules, independent Poisson counts are generated using NumPy's documented pseudorandom generator. These choices create non-identical, heterogeneous traces without implying that the parameter distributions were statistically estimated from any cloud provider. Function class membership is determined by the function index modulo four, ensuring exactly 36 functions of each class per run. Runtime profile and memory size are sampled independently of this class. This deliberate independence is useful diagnostically: phase-aware allocation must identify expensive startup costs rather than simply identifying a special workload class with systematically larger latency. Memory is also not inferred from execution durations; the synthetic experiment has no genuine memory telemetry. The seed-11 count matrix and its phase parameters are distributed as a compressed NumPy archive with SHA-256 metadata; every other trace can be reconstructed from the listed seeds. The production Azure data source is relevant for future trace-based validation, but its format creates another methodological distinction. Its counts are recorded in one-minute fields, with function identifiers and trigger categories, and its application memory is supplied in separate files. Aggregating or expanding these fields into exact invocation timestamps would introduce within-minute assumptions. Our simulator therefore adopts the same minute-based observation granularity as a tractable abstraction, while labeling all generated counts as synthetic and avoiding any claim that the test distribution reproduces the production trace.

B. Runtime and memory model. Each function independently receives one of four illustrative cold-start profiles and one of six assumed memory sizes (96, 128, 192, 256, 384, or 512 MB). The startup tuples are drawn from the phase breakdown and runtime profiles illustrated in [4], encoded directly as (infrastructure, runtime, application) values in milliseconds. These are scenario inputs and not latency observations collected during our evaluation.

Table 1TABLE I. ILLUSTRATIVE THREE-PHASE STARTUP COSTS (MS)

Runtime	**Infra.**	**Runtime**	**App.**
Node.js	90	20	60
Python	95	35	80
Java	100	880	120
Go	80	8	25

C. Evaluation protocol.

The first two days provide causal history. The remaining five days are evaluated in chronological order while predictors continue to update from past buckets. Six independent seeds (11, 29, 47, 71, 89, and 107) generate independent complete runs. We use caps of 4,096, 8,192, and 12,288 MB. A reserved slot is assumed to be available at the scheduled minute boundary, abstracting container creation lead time. The evaluation counts at most one cold initialization per function-minute and divides total modeled delay by the actual generated request count. No measured application response time or production SLO percentile enters this calculation.

D. Baselines and ablations.

Reactive launches on every nonempty cold bucket and consumes no warm memory. Fixed15 retains a function during the first 15 idle minutes, applying the common budget cap. Histogram implements the two-

sided quantile-window variant described above. EMA and Seasonal rank estimated activity per MB using one signal only. Phase-EMA uses the short-term estimate weighted by cold cost; Dual-no-Phase combines both estimates but uses no cold-cost weighting. PADA+Histogram introduces a histogram-hazard contribution (0.38 EMA, 0.43 seasonal, 0.19 hazard); PADA uses the two-signal combination in (3). The histogram methods are reduced, explicit policies—not full reproductions of every production heuristic or fallback used in the Azure system.

All policies are evaluated with the same generated counts for a given seed, which supports paired comparisons. Fixed15 and Histogram are masked retention-window policies; they can reserve fewer MB than the cap when only a limited number of functions qualify. Forecast-based policies consider all positive-score functions and generally fill almost the entire cap, leaving only fragmentation from the discrete memory sizes. This difference is reported directly as occupancy rather than concealed through an assumption of identical utilization. Since the simulator charges no creation overhead for prefilling warm slots, this favors purely forecast-driven policies over deployment systems in which prewarming could be delayed.

The quantile-window policy is intended to represent the implementable window mechanism, not the complete Azure deployment policy. In particular, we exclude the specialized time-series fallback used for some irregular functions in the original production study, and cap observable gaps at six hours. This is important because an exact replication claim would require matching adaptive fallbacks, production warmup timing, application-level grouping, and resource-accounting details. The concrete adopted component here is the per-function online inter-arrival histogram and its two-sided prewarming/retention window, whose source and implementation can be audited.

E. Statistical analysis.

The independent unit is the generated realization (seed), not an individual invocation. We compute means and sample standard deviations over six runs. For selected policy differences, two-sided 95% Student-t intervals are calculated from paired per-seed differences. Fixed coefficients and budgets are identical across seeds; the results are an evaluation on a specified synthetic family rather than a model-selection benchmark.

VI. RESULTS

A. Main comparison. Table II gives the eight nonreactive policies and the reactive anchor under the 8,192-MB cap. PADA produces a modeled penalty of 43.22 ms/request compared with 89.31 for Fixed15, 68.12 for EMA, and 63.67 for Dual-no-Phase. The reductions are 51.6%, 36.5%, and 32.1%, respectively, when computed from across-seed means. PADA also reduces the modeled penalty relative to Phase-EMA (50.18), indicating incremental value from the historical daily feature even after cold-cost weighting.

The reactive reference incurs 289.84 modeled milliseconds per request by initializing each active function-minute from cold. Its zero reserved-memory usage makes it a useful zero-prewarming anchor but not a memory-matched competitor. All forecast-driven methods at 8,192 MB use approximately 8,167 MB on average, so the strongest comparisons are PADA against EMA, Phase-EMA, Dual-no-Phase, and PADA+Histogram at nearly identical reservation occupancy. Fixed15 uses approximately 8,040 MB and Histogram approximately 8,113 MB, which should be considered when interpreting their absolute differences. As shown in Table II, neither larger occupancy alone nor higher warm-hit counts fully explain the ranking of modeled penalties.

Table 28192-MB CAP: SIX-SEED MEANS (SIMULATED)

Policy	Cold %	ms/req	Mem MB
Reactive	100.0	289.84	0
Fixed15	32.4	89.31	8040
Histogram	52.7	151.11	8113
EMA	23.6	68.12	8168
Seasonal	35.1	99.77	8167
Phase-EMA	25.6	50.18	8168
No phase	22.4	63.67	8168
PADA+hist	33.2	50.67	8168
PADA	27.3	43.22	8167

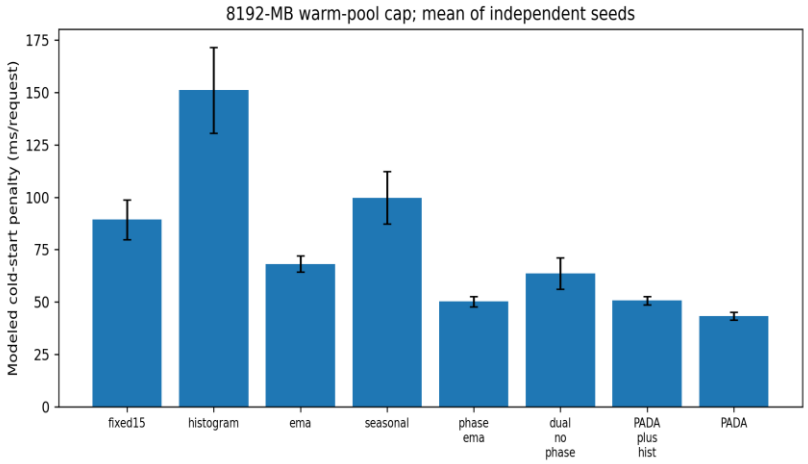


Figure 1Modeled penalty by policy at the 8192-MB cap (error bars: 95% t intervals).

Cold-start frequency alone leads to a different ranking. At 8,192 MB, PADA leaves 27.28% of active function-minutes cold, while EMA leaves 23.59% and Dual-no-Phase 22.38% cold. The opposite ordering for modeled delay is expected: PADA preferentially protects functions with high miss costs even if they are invoked less often. This distinction is central to the objective in (2) and prevents cold count from being misreported as user-visible delay.

B. Resource sensitivity.

Across all three caps, PADA reduces modeled penalty with increasing available memory: 85.48, 43.22, and 24.23 ms/request under 4,096, 8,192, and 12,288 MB, respectively. Fixed15 yields 160.82, 89.31, and 51.18 ms/request under the same caps. Because the greedy allocator can leave small unusable memory fragments, the mean reserved MB lies slightly below each cap. The budget sensitivity plot shows the consistent direction without requiring an invocation-rate prediction oracle.

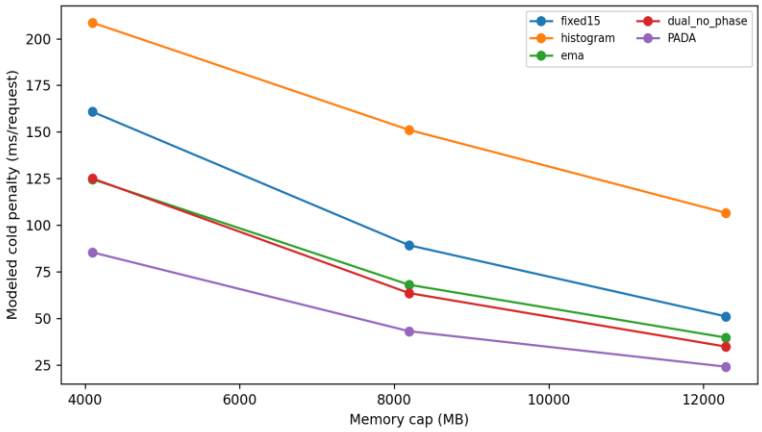


Figure 2Modeled penalty across three warm-pool memory caps.

C. Paired comparisons.

At 8,192 MB, the paired difference Fixed15 minus PADA is 46.08 ms/request (95% CI: 37.80–54.37), and EMA minus PADA is 24.90 ms/request (95% CI: 22.31–27.49). These intervals quantify variation among the six independent synthetic realizations, not uncertainty over production deployments. Results from the other budgets also favor PADA on the modeled-delay objective, as shown in Fig. 2.

D. Ablation.

The 8,192-MB penalty rises from 43.22 to 50.18 ms/request if the seasonal signal is disabled while cold-cost weighting remains (Phase-EMA). Removing phase-cost weighting increases the penalty to 63.67, even though it reduces the number of cold starts. Adding the raw histogram hazard raises the penalty to 50.67 in the mixed workload, revealing that the extra feature is not uniformly beneficial. In a fleet with sparse and shifting functions, a large empirical gap does not necessarily provide reliable next-minute activation risk. This negative ablation argues for retaining the simpler dual predictor in PADA rather than treating prediction-model complexity as inherently beneficial. The phase-removal comparison is particularly informative because both methods use precisely the same underlying dual demand score; only the multiplicative D(i) term in the ranking changes. At 8,192 MB, Dual-no-Phase achieves a lower cold-active fraction (22.38%) than PADA (27.28%), but its greater concentration of heavyweight misses increases modeled delay from 43.22 to 63.67 ms/request. The objective improvement therefore cannot be reduced to a generic “better prediction” claim. Similarly, the Phase-EMA ablation retains identical cost weighting while removing previous-day information, showing that two time scales are relevant within this particular generator. A diagnostic test evaluates the raw prediction signals with a Brier score against binary active-minute outcomes. Scores depend on class: for periodic functions, the combined estimate obtains a mean score of 0.0385, compared with 0.0386 for EMA alone, whereas bursty functions favor the more reactive EMA (0.0457 versus 0.0527 for the dual estimate). These are descriptive diagnostics computed on the evaluation portions of the same synthetic runs. They do not establish universal probabilistic calibration; they help explain why a small amount of predictable daily structure can aid allocation without dominating every workload type.

E. Interpretation.

The experiment separates two mechanisms. A workload-sensitive forecast can move reservations before periodic peaks and react to recent bursts, while phase weighting changes which predicted arrivals are valuable enough to protect. Both effects are visible under the same hard memory constraints. Histogram windows remain a useful tested point of comparison, but the quantile-window implementation has weak performance in this specific workload mixture; it does not establish a general ranking against the more elaborate production policy studied in [3].

F. Workload composition.

At the 8,192-MB cap, Fig. 3 reports the cold active-bucket fractions broken down by the four generator classes. PADA incurs 11.48% cold periodic buckets, compared with 27.69% for Fixed15, reflecting the advantage of reserving capacity in advance of recurring peaks. Sparse functions remain mostly cold (92.20% for PADA), which is expected under a tight budget and their low activation probability. Relative to EMA, PADA permits somewhat more cold buckets in steady and bursty classes; those additional misses tend to have lower modeled startup cost, preserving the aggregate penalty advantage. The class breakdown guards against interpreting one global average as uniform superiority on every traffic pattern.

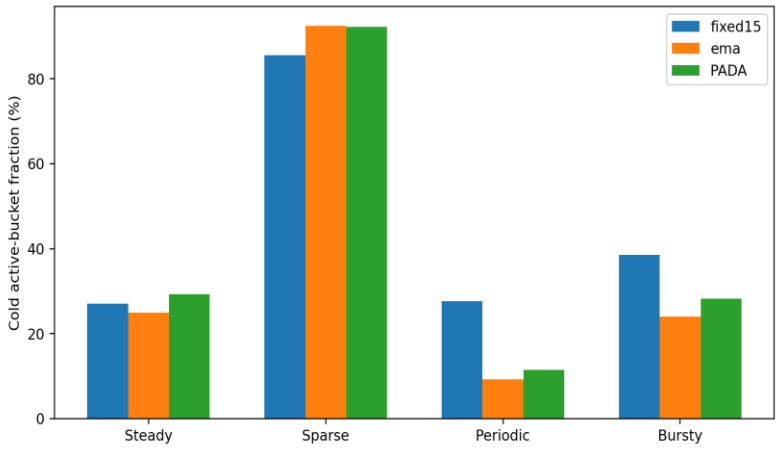


Figure 3Cold active-minute rates by synthetic workload class at 8192 MB.

G. Readiness-lag stress test.

The main simulator ranks and assigns warm slots before processing each minute and assumes the chosen slot is usable within the same decision interval. We independently evaluate readiness lag h in {0, 1, 3, 5} minutes: a slot selected at minute t−h is the only proactively available slot when minute t is served. Selection at t−h uses only history older than that boundary, and every historical selection obeys the same memory cap. This experiment varies the freshness of the selected configuration and the assumed preparation horizon; it is not an instrumented measurement of container startup time. Lag-zero outcomes coincide with those of the main simulator for each seed, cap, and shared policy, which provides a numeric consistency check.

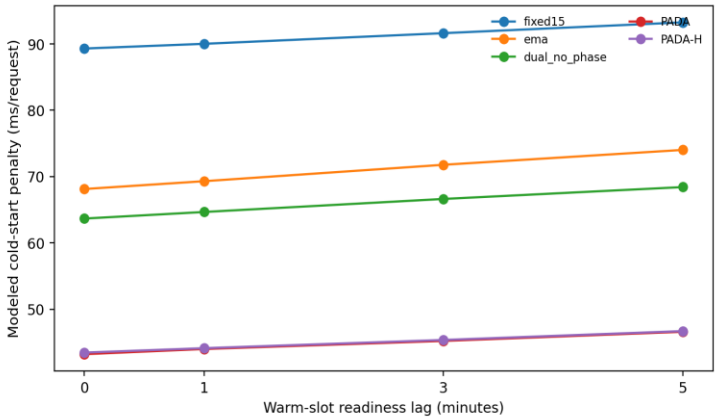


Figure 4Modeled latency under warm-slot readiness lag at 8192 MB (synthetic trace).Modeled latency under warm-slot readiness lag at 8192 MB (synthetic trace).

At 8,192 MB with h=1, PADA obtains 44.00 modeled ms/request versus 90.02 for Fixed15, 69.30 for EMA, and 64.66 for Dual-no-Phase. The paired six-seed Fixed15-minus-PADA difference is 46.01 ms/request (95% CI 37.86–54.16). For h=3 and h=5, the PADA penalty is 45.22 and 46.60, respectively, while Fixed15 is 91.62 and 93.25. Thus the allocation ordering remains favorable under these declared lags, but the simulated cost increases with forecast staleness. The lag scenarios do not reserve additional memory for pending initialization, so these observations should be interpreted as readiness/freshness sensitivity under a simplified capacity model.

H. Warm-slot stability.

Frequent proactive reselection has costs outside the latency-only objective. We therefore count planned 0-to-1 warm-slot transitions per 1,000 simulated minutes, separately from actual arrivals. We also implement PADA-H, a retention-biased extension that multiplies the ranking score of a slot already selected in the preceding minute by 1.5 before enforcing the same memory cap. The factor is fixed as an experimental policy setting, not optimized against the reported test seeds. Its causal state is the preceding selected vector; no future activity indicator enters the decision.

At 8,192 MB and h=1, standard PADA performs 1,653.45 planned slot activations per 1,000 minutes for a fleet of 144 functions; PADA-H performs 742.08, a reduction of 55.12%. Mean modeled latency changes from 44.00 to 44.13 ms/request. Across six seeds the paired PADA-H-minus-PADA difference is 0.123 ms/request (95% CI −0.105 to 0.351); therefore the latency increase is not statistically resolved with these runs. Because transitions are counted as configuration changes rather than measured container creations, the result quantifies policy churn, not actual launch time, energy, or cloud charges. PADA-H exposes a useful tunable trade-off: slight ranking inertia can markedly reduce reselection without materially shifting modeled latency in this synthetic setting.

VII. DISCUSSION AND THREATS TO VALIDITY

The study isolates the scheduling decision but deliberately abstracts several service-level details. One minute is a coarse temporal resolution; an invocation-count bucket cannot reveal inter-request spacing, concurrency, failed starts, memory oversubscription, or queuing. The primary evaluation assumes the selected warm slot is ready within its decision interval; Section VI-G probes additional readiness lag and Section VI-H measures planned slot churn. Neither experiment captures container launch failures or physical initialization capacity. Our numbers are therefore modeled cold-start delay increments, not measured end-to-end response times. Memory assignments are per-function proxies rather than genuine sampled Azure application-level allocations. The six evaluation seeds sample workload stochasticity, not uncertainties in simulator structure. Given a seed, all arrivals, runtime assignments, memory sizes, selection scores, and counted cold starts are deterministic. The paired confidence intervals therefore describe between-realization variability conditional on these fixed modeling choices. Invocation-level hypothesis tests would spuriously inflate sample size by treating correlated function-minutes as independent, which is why the run rather than the request is the unit for uncertainty calculations. There are two plausible objective alternatives. An operator concerned primarily with the probability of a cold invocation can set every $D(i)=1$ and optimize warm-hit coverage instead; Dual-no-Phase is an explicit approximation to that choice. An operator subject to strict class-specific service limits may require per-class constraints instead of an additive mean delay objective. Under such constraints, the current ranking might allocate too few warm slots to an inexpensive but high-priority function. The executable separation between arrival risk, phase penalties, and slot memory would permit these objectives to be investigated without altering the raw trace generator. The event generator exposes four stylized temporal patterns and imposed drift rather than an empirically fitted cloud fleet. Consequently, it provides controlled tests of burstiness and recurrence but cannot establish generalization to unseen production workloads. The 31-minute seasonal smoothing span, the mixture coefficients, and the equal group representation are fixed assumptions. A production study should replace these with historical traces and direct container startup measurements, test sub-minute scheduling periods and physically measured prewarming lead times, and evaluate simultaneous warm replicas under burst concurrency. A second boundary concerns interpretation of the phase components. Infrastructure, runtime, and application penalties are drawn from the illustrative runtime-profile analysis, rather than from an instrumented FaaS cluster. Testing multiple memory caps and reporting a phase-removal ablation addresses whether heterogeneity matters inside the model, but not whether those specific latency tuples hold on any particular serverless provider. Real profiles would need to be collected with per-stage timing instrumentation and periodically recalibrated as deployment packages change.

Finally, PADA uses greedy benefit density rather than solving a time-dependent knapsack exactly. Under the small single-replica model this is computationally attractive, yet greedy selection may reject combinations with a better aggregate gain. Forecast uncertainty, calibration, multi-tenant isolation costs, cold-start blocking, and memory-price variation also merit further study, as does a greedy-dual (FaasCache-style) comparator under the same memory cap. The implementation and its full trace generator, configuration, random seeds, and individual-run CSV files are included so these assumptions can be modified and retested.

VIII. CONCLUSION

PADA combines a causal recent-activity estimate and previous-day seasonal activity with a three-component cold-start cost and memory-normalized ranking. In six synthetic workload realizations, the proposed policy lowers modeled cold-start penalty across three memory caps, while an ablation shows that removing phase weighting degrades that objective even when raw cold-start counts improve. An implemented histogram-window comparison and an added-hazard ablation also reveal that additional arrival-gap signals need not improve a predictor under drift. The result is a reproducible scheduling-level contribution: latency-aware warm-slot prioritization can be materially different from cold-hit maximization. An independent readiness-lag evaluation preserves the qualitative benefit across the tested horizons, while

retention hysteresis substantially reduces planned warm-slot churn. Extension to measured invocation traces and real FaaS warmup measurements is necessary for external validity.

REFERENCES

E. Jonas et al., “Cloud programming simplified: A Berkeley view on serverless computing,” arXiv:1902.03383, 2019. doi: 10.48550/arXiv.1902.03383.

S. Eismann et al., “The state of serverless applications: Collection, characterization, and community consensus,” IEEE Trans. Softw. Eng., vol. 48, no. 10, pp. 4152–4166, 2022. doi: 10.1109/TSE.2021.3113940.

M. Shahrad et al., “Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider,” Proc. USENIX ATC, pp. 205–218, 2020. Available: https://www.usenix.org/conference/atc20/presentation/shahrad

X. Yue and R. Zhu, “An explainable hybrid machine learning framework for cash-flow-conditioned multi-horizon liquidity risk early warning: An SME-oriented benchmark study,” World Journal of Management Science, vol. 2, no. 1, pp. 63–72, 2024.

M. Golec et al., “Cold start latency in serverless computing: A systematic review, taxonomy, and future directions,” ACM Comput. Surv., vol. 57, no. 3, Art. 65, 2025. doi: 10.1145/3700875.

Y. Li, Y. Lin, Y. Wang, K. Ye, and C. Xu, “Serverless computing: State-of-the-art, challenges and opportunities,” IEEE Trans. Serv. Comput., vol. 16, no. 2, pp. 1522–1539, 2023. doi: 10.1109/TSC.2022.3166553.

A. Fuerst and P. Sharma, “FaasCache: Keeping serverless computing alive with greedy-dual caching,” Proc. ASPLOS, pp. 386–400, 2021. doi: 10.1145/3445814.3446757.

J. Wang, J. Liu, Y. Ge, and B. Jiang, “Risk-aware ensemble learning for credit default prediction under economic uncertainty,” World Journal of Economics and Business Research, vol. 1, no. 1, pp. 18–32, 2023.

J. R. Gunasekaran, P. Thinakaran, N. C. Nachiappan, M. T. Kandemir, and C. R. Das, “Fifer: Tackling resource underutilization in the serverless era,” Proc. Middleware, pp. 280–295, 2020. doi: 10.1145/3423211.3425683.

A. Mampage, S. Karunasekera, and R. Buyya, “A holistic view on resource management in serverless computing environments: Taxonomy and future directions,” ACM Comput. Surv., vol. 54, no. 11s, 2022. doi: 10.1145/3510412.

A. Agache et al., “Firecracker: Lightweight virtualization for serverless applications,” Proc. USENIX NSDI, pp. 419–434, 2020. Available: https://www.usenix.org/conference/nsdi20/presentation/agache

J. Cadden, T. Unger, Y. Awad, H. Dong, O. Krieger, and J. Appavoo, “SEUSS: Skip redundant paths to make serverless fast,” Proc. EuroSys, 2020. doi: 10.1145/3342195.3392698.

D. Ustiugov, P. Petrov, M. Kogias, E. Bugnion, and B. Grot, “Benchmarking, analysis, and optimization of serverless function snapshots,” Proc. ASPLOS, pp. 559–572, 2021. doi: 10.1145/3445814.3446714.

H. Li, Y. Yuan, R. Du, K. Ma, L. Liu, and W. Hsu, “DADI: Block-level image service for agile and elastic application deployment,” Proc. USENIX ATC, pp. 727–740, 2020. Available: https://www.usenix.org/conference/atc20/presentation/li-huiba

S. Shillaker and P. Pietzuch, “Faasm: Lightweight isolation for efficient stateful serverless computing,” Proc. USENIX ATC, pp. 419–433, 2020. Available: https://www.usenix.org/conference/atc20/presentation/shillaker

Y. Sui, H. Yu, Y. Hu, J. Li, and H. Wang, “Pre-warming is not enough: Accelerating serverless inference with opportunistic pre-loading,” Proc. ACM SoCC, pp. 178–195, 2024. doi: 10.1145/3698038.3698509.

D. Bermbach, A.-S. Karakaya, and S. Buchholz, “Using application knowledge to reduce cold starts in FaaS services,” Proc. ACM SAC, pp. 134–143, 2020. doi: 10.1145/3341105.3373909.

S. Eismann, J. Grohmann, E. van Eyk, N. Herbst, and S. Kounev, “Predicting the costs of serverless workflows,” Proc. ACM/SPEC ICPE, pp. 265–276, 2020. doi: 10.1145/3358960.3379133.
K. Kaffes, N. J. Yadwadkar, and C. Kozyrakis, “Centralized core-granular scheduling for serverless functions,” Proc. ACM SoCC, pp. 158–164, 2019. doi: 10.1145/3357223.3362709.
J. Chen, X. Han, X. Zhang, and Y. Yang, “Exact concept-level explanations for early detection of financial misstatements,” Eurasia Journal of Science and Technology, vol. 5, no. 2, pp. 34–48, 2023.
P. Gackstatter, P. A. Frangoudis, and S. Dustdar, “Pushing serverless to the edge with WebAssembly runtimes,” Proc. IEEE/ACM CCGrid, pp. 140–149, 2022. doi: 10.1109/CCGrid54584.2022.00023.
T. Pfandzelter and D. Bermbach, “tinyFaaS: A lightweight FaaS platform for edge environments,” Proc. IEEE ICFC, pp. 17–24, 2020. doi: 10.1109/ICFC49376.2020.00011.
Anjali, T. Caraza-Harter, and M. M. Swift, “Blending containers and virtual machines: A study of Firecracker and gVisor,” Proc. VEE, pp. 101–113, 2020. doi: 10.1145/3381052.3381315.
A. Panda and S. R. Sarangi, “FaaSCtrl: A comprehensive-latency controller for serverless platforms,” IEEE Trans. Cloud Comput., vol. 12, no. 4, pp. 1328–1343, 2024. doi: 10.1109/TCC.2024.3473015.
A. Ebrahimi, M. Ghobaei-Arani, and H. Saboohi, “Cold start latency mitigation mechanisms in serverless computing: Taxonomy, review, and future directions,” J. Syst. Archit., vol. 151, Art. 103115, 2024. doi: 10.1016/j.sysarc.2024.103115.
A. Mohan, H. Sane, K. Doshi, S. Edupuganti, N. Nayak, and V. Sukhomlinov, “Agile cold starts for scalable serverless,” Proc. USENIX HotCloud, 2019. Available: https://www.usenix.org/conference/hotcloud19/presentation/mohan
J. Chen, X. Zhang, and X. Han, “Explainable identification of tax-related financial reporting anomalies in U.S. public companies: Cross-fitted tax-consistency residuals and Owen-value Shapley propagation,” World Journal of Economics and Business Research, vol. 1, no. 1, pp. 33–45, 2023.
W. Shang, D. Teng, T. Chen, F. Zhang, and J. Ding, “Neuro-Elastic: A Unified Framework for Hardware-Aware Adaptive Quantization and Dynamic Sparsity in Real-Time Edge Intent Prediction,” IEEE Access, 2026, doi: 10.1109/ACCESS.2026.3715276.