**Journal of Computer Science and Electrical Engineering**
**Print ISSN: 2663-1938**
**Online ISSN: 2663-1946**
**DOI: https://doi.org/10.61784/jcsee4150**

# CAPA: A PREDICTION-DRIVEN AUTOSCALING FRAMEWORK FOR SLO-AWARE CLOUD-NATIVE MACHINE LEARNING INFERENCE

WenYu Zhao[1], BoYuan Wang[2*]
[1]*Microsoft, Beijing 100080, China.*
[2]*University of Southern California, Los Angeles 90089, CA, USA.*
[*]*Corresponding Author: BoYuan Wang*

**Abstract:** Cloud-native machine-learning inference must reconcile sub-second demand changes with non-negligible replica startup delay. Reactive horizontal scaling therefore tends to provision after a burst has already accumulated a queue, whereas aggressive prediction wastes resources when a workload is stable. This paper presents CAPA, a confidence-aware predictive autoscaler that couples a lightweight rolling ridge forecaster with an online one-sided residual-quantile bound and an SLO-debt feedback signal. The upper demand estimate covers startup delay; the debt signal lowers the utilization target when recent requests approach or exceed a latency objective; and an explicit queue-drain term converts backlog into temporary capacity. A reproducible evaluation combines 250 measured CPU inference latencies and seven measured process cold starts for MobileNetV3-Small with three fixed-seed, trace-calibrated stress profiles. Across eight paired runs, CAPA reduces the estimated SLO violation rate relative to a reactive HPA-style baseline by 17.83% on bursty traffic and 11.37% under regime shifts, while increasing mean replicas by 2.59% and 1.79%, respectively. The improvement is statistically significant in both nonstationary profiles. On a regular periodic workload, CAPA uses 3.82% fewer replicas but raises violations by 14.55%, exposing a clear boundary condition rather than hiding a negative result. The code, raw measurements, generated traces, and statistical outputs are packaged with the paper.
**Keywords:** Cloud-native systems; Autoscaling; Machine-learning inference; Service-level objectives; Workload forecasting; Uncertainty calibration

## 1 INTRODUCTION

Online inference is now a latency-sensitive cloud service rather than an offline model invocation. A single request may be inexpensive, yet the aggregate service is difficult to operate because arrivals are bursty, model replicas take time to initialize, and tail latency—not average latency—determines user-visible quality. The tail-at-scale effect makes even a small probability of a slow component consequential in distributed applications [1]. Modern cluster managers provide declarative replication and isolation, but their generic autoscalers usually react to utilization after demand has changed [2-3]. For inference, that reaction can arrive too late: a queue can form during image loading, framework import, model construction, or accelerator allocation.

Prior inference systems optimize complementary layers. Clipper and TensorFlow Serving provide flexible serving paths [4-5]; Swayam and MArk explicitly connect scaling to inference SLAs or SLOs [6-7]; InferLine plans and tunes multi-stage prediction pipelines [8]; and Clockwork, INFaaS, Cocktail, Orca, AlpaServe, SHEPHERD, and vLLM improve predictability, model selection, batching, placement, or memory management [9-15]. These systems establish that serving efficiency depends on workload shape and model behavior. However, a deployable autoscaling controller still faces a practical gap between purely reactive thresholds and heavyweight predictors that may be brittle under distribution shift.

CAPA addresses this gap with a deliberately small control model. It forecasts demand only far enough to cover replica startup, constructs a one-sided upper estimate from recent residuals, and closes the loop with an exponentially weighted SLO-debt signal. This separation is important: prediction anticipates demand, while debt corrects prediction and queue-model error using observed service quality. The controller is intended to sit beside a cloud-native inference deployment and emit replica targets; it does not require changes to the model server.

The work makes three contributions. First, it introduces a confidence-aware capacity rule that combines a rolling ridge forecast, an online residual-quantile margin, and a queue-drain term. Second, it incorporates SLO debt into the target utilization, yielding asymmetric behavior: uncertainty and recent violations trigger headroom, whereas scale-down remains conservative. Third, it supplies a fully reproducible prototype-driven simulation in which service latency and startup delay are measured locally rather than invented. The evaluation also reports a negative periodic-workload result and identifies why prediction is not universally beneficial.

## 2 BACKGROUND AND RELATED WORK

### 2.1 Cloud Autoscaling

Autoscaling methods span thresholds, control theory, queueing, time-series prediction, and learning-based policies [16]. Production cluster managers such as Borg demonstrate the value of centralized resource accounting, admission control, and overcommitment, while the Borg–Omega–Kubernetes lineage made reconciliation loops and declarative desired state common cloud-native abstractions [2-3]. Resource Central showed that large-cloud workloads are heterogeneous and that historical signals can improve resource decisions [17]. Autopilot combines historical learning with operational heuristics at Google scale [18]. These systems motivate prediction, but they also show that prediction must be guarded by robust control because workloads and software versions change.

### 2.2 Inference Serving Systems

Clipper introduced a low-latency abstraction across ML frameworks and used batching, caching, and adaptive model selection [4]. TensorFlow Serving emphasized extensibility, version management, and optimized lookup paths [5]. Swayam provided distributed model-based autoscaling for production inference services, and MArk combined heterogeneous cloud resources with predictive autoscaling for cost-effective SLO-aware serving [6-7]. InferLine separated low-frequency planning from a high-frequency tuning loop for prediction pipelines [8]. Clockwork exploited deterministic execution characteristics to obtain predictable serving, whereas INFaaS selected model variants and hardware without requiring callers to choose a concrete model configuration [9-10].

Recent systems broaden the optimization space. FIRM and Sinan use telemetry and learning to manage SLOs for microservices [19-20]. Cocktail jointly considers accuracy, latency, and deployment cost [11]. Orca and PagedAttention target autoregressive transformer serving through iteration-level scheduling and efficient KV-cache management [12, 15]. AlpaServe uses model parallelism for statistical multiplexing, and SHEPHERD studies DNN serving under realistic variability [13-14]. CAPA is narrower: it does not redesign batching or placement. Instead, it targets the replica-control loop that can surround these serving engines.

Runtime and cluster schedulers further shape the capacity seen by an autoscaler. PRETZEL exposes cross-model, white-box prediction-serving optimizations, while Nexus coordinates multiple DNN applications on GPU clusters under latency constraints [21-22]. Gandiva exploits deep-learning job structure for introspective GPU scheduling, and Quasar classifies workloads to improve resource efficiency subject to QoS goals [23-24]. These systems reinforce a key interface assumption in CAPA: the replica controller should consume a measured effective service profile rather than treat nominal hardware count as fixed capacity.

### 2.3 Forecasting and Calibration

Exponential smoothing remains a strong lightweight baseline for level and trend forecasting, and modern forecasting practice stresses rolling evaluation and transparent baselines [25-26]. Ridge regression is attractive in a controller because it is stable with correlated lag features and inexpensive to refit [27]. Online conformal methods provide a principled view of coverage under distribution shift [28]. CAPA does not claim finite-sample conformal validity: its windowed residuals are not exchangeable and the target changes over time. Instead, it adopts the operational idea of recalibrating a one-sided residual quantile online and reports empirical coverage explicitly.

## 3 PROBLEM FORMULATION

Consider a stateless inference service sampled every second and controlled every $\Delta$ seconds. Let $\lambda\_t$ denote arrivals, $n\_t$ the number of active replicas, $b\_t$ the queued requests, $\mu$ the measured mean service rate of one replica, and $d\_s$ the startup delay. A latency SLO $L$ defines violation probability $v\_t = \Pr\{R\_t > L\}$, where $R\_t$ is response time. The controller chooses a desired replica target $a\_k$ at decision epoch $k$. New replicas become active after $d\_s$; scale-down is immediate but constrained by cooldown and a maximum removal fraction.

A useful operating objective balances service quality, capacity, and churn rather than optimizing one metric in isolation: $\min \quad \Sigma_t [ w_v \lambda_t v_t + w_r n_t + w_s |n_t - n_{t-1}| ], 1 \le n_t \le n\_max$ .

The first term weights violations by requests so that a brief high-volume burst matters more than an idle interval. The second represents replica-seconds, and the third discourages oscillation. CAPA does not solve this mixed-integer problem globally. It implements a receding, interpretable approximation whose parameters map to observable quantities: forecast uncertainty, accumulated SLO debt, queue size, service rate, and startup delay.

## 4 CAPA FRAMEWORK

### 4.1 Startup-Horizon Forecast

At each decision epoch, CAPA averages arrivals over the most recent control interval and constructs a feature vector containing four lags, five- and fifteen-window means, a local trend, and sine/cosine terms for short and long periods. The periodic terms are optional domain features; they do not prevent operation on nonperiodic traffic because ridge regularization can shrink them. The model is refit every six decisions over at most $W = 180$ labeled windows. With regularization $\alpha$, the coefficient vector is

$$\hat{\beta}_k = (X_k^T X_k + \alpha I)^{-1} X_k^T y_k , \quad \hat{\lambda}_{k+h} = x_k^T \hat{\beta}_k \tag{1}$$

The horizon h is the smallest number of control intervals covering startup delay. Refitting every 30 s in the default configuration reduces controller overhead and avoids responding to each Poisson fluctuation. During warm-up, CAPA uses the larger of the last observation and a three-window mean (Figure 1).

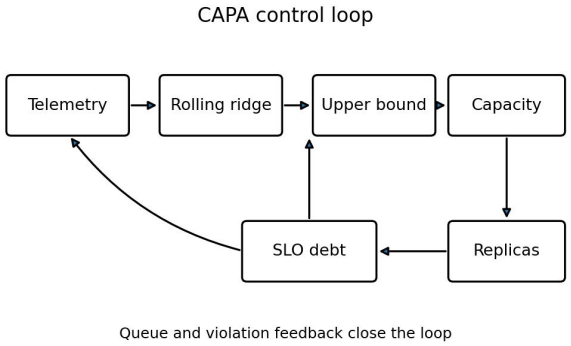


**Figure 1** CAPA Combines Forecast Uncertainty with SLO-Debt Feedback and Emits a Replica Target

### 4.2 Online One-Sided Uncertainty

A point forecast is insufficient when underprediction is more harmful than overprediction. After each fit, CAPA computes residuals $e_i = y_i - x_i^T\beta$ and takes the empirical 0.90 quantile $q_{0.90}$ using the conservative higher order statistic. The protected demand is

$U_k = \max(\hat{\lambda}_{k+h} + q_{0.90}, \bar{\lambda}_{\text{recent-peak}})$.

The recent-peak guard catches a sharp rise inside the observation window. The uncertainty ratio $z_k = \max(0, U_k - \hat{\lambda}_k)/\max(U_k,1)$ converts the absolute margin into a dimensionless headroom signal. This mechanism is intentionally auditable: operators can inspect the point forecast, upper estimate, and empirical coverage rather than relying on an opaque uncertainty score.

### 4.3 SLO Debt and Capacity Rule

Prediction cannot represent all latency sources. CAPA therefore maintains an exponentially weighted debt $D_k$ from the estimated request-level violation probability. A high debt indicates that recent capacity or queue assumptions were optimistic. The target utilization becomes

$u_k = \text{clip}(0.72 - 0.20\, z_k - 0.22\, D_k, 0.46, 0.72)$.

The constants define an operating envelope rather than a learned optimum. Their effect is monotone and interpretable: higher uncertainty or debt lowers utilization and creates headroom. To clear existing work, CAPA converts backlog to a temporary arrival-equivalent term $b_k/T_d$, where $T_d = \max(5\text{ s}, 2d_s)$. The desired target is

$a_k = \text{ceil}[\,(U_k + b_k/T_d)\,/\,(\mu\, u_k)\,]$.

The target is clipped to 1–12 replicas in the experiment. Scale-up adds all required pending replicas; scale-down removes at most 25% of active replicas and observes a 30 s cooldown. This asymmetric hysteresis treats missing capacity during a burst as more expensive than briefly retaining excess capacity.

**Algorithm 1** One CAPA Control Decision

| Step | CAPA decision procedure |
|---|---|
| Input | arrival history, backlog $b_k$, active/pending replicas, measured $\mu$ and $d_s$ |
| 1 | Aggregate the last $\Delta$ seconds and build the 12-feature vector $x_k$. |
| 2 | Every six decisions, refit rolling ridge on observable horizon labels. |
| 3 | Compute point forecast $\lambda$-hat and one-sided residual quantile q. |
| 4 | Set $U_k$ to the maximum of $\lambda$-hat + q and the recent-peak guard. |
| 5 | Update SLO debt $D_k$ and derive uncertainty/debt-aware utilization $u_k$. |
| 6 | Add the queue-drain term $b_k/T_d$ and compute the clipped replica target. |
| 7 | Scale up immediately; scale down conservatively under cooldown. |

### 4.4 Safety and Failure Containment

CAPA preserves three operational invariants. First, every decision is clipped to configured replica bounds, so a forecast cannot request unbounded capacity. Second, pending replicas are counted before additional scale-up, preventing duplicate provisioning while startup is in progress. Third, prediction affects only the desired target: if the forecaster is unavailable or lacks sufficient history, the warm-up rule falls back to recent observations. A production adapter can additionally reject stale telemetry and cap step changes. These safeguards make forecast failure degrade toward reactive behavior rather than bypassing the cluster's resource policy.

### 4.5 Deployment Path and Complexity

In Kubernetes, the telemetry inputs can be produced by Prometheus-compatible counters, request queues, and latency histograms, while the output can update a scale subresource or an external autoscaling adapter. The controller stores O(Wp) feature data for p = 12 features and solves a p × p linear system every refit. With the default window, the computation is negligible compared with model inference. CAPA is compatible with CPU or accelerator-backed serving provided a per-replica service profile and an effective startup delay are available.

## 5 EXPERIMENTAL METHODOLOGY

### 5.1 Reproducibility Boundary

The evaluation deliberately separates measured quantities from synthesized stress input. We attempted to use the public Azure Functions 2019 data associated with Serverless in the Wild [29]. The official release establishes realistic heterogeneity and burstiness, but the large compressed asset was not reliably retrievable in the execution environment. We therefore do not claim trace replay. Instead, we use fixed-seed synthetic profiles inspired by the qualitative properties reported in the public study, and we package the generator so every arrival can be regenerated. This choice is preferable to silently substituting unverifiable numbers.

### 5.2 Measured Inference Worker

The service worker is torchvision MobileNetV3-Small with batch size one and 224 × 224 input on CPU. Random weights avoid an external model download; they preserve the architecture and execution cost, and no accuracy claim is made. PyTorch is restricted to one intra-operation thread. After 30 warm-up iterations, 250 forward passes are timed with a monotonic nanosecond clock. Cold start is measured seven times by launching a fresh Python subprocess, importing PyTorch and torchvision, constructing the model, creating an input tensor, and completing the first inference. Table 1-2 reports the resulting measurements rather than assumed values.

**Table 1** Measured Model Parameters

| Metric | Measured value |
|---|---|
| Mean forward latency | 4.899 ms |
| Median / p95 / p99 | 4.830 / 5.457 / 5.875 ms |
| Mean-derived service rate | 204.11 req/s |
| Cold-start median / p95 | 2509.7 / 2588.0 ms |
| Software | Python 3.13.5; PyTorch 2.10.0 CPU |
| Samples | 250 warm inferences; 7 cold starts |

**Table 2** Simulation and Control Settings

| Setting | Value |
|---|---|
| Run length / decision interval | 7,200 s / 5 s |
| Latency SLO / replica bounds | 20 ms / 1–12 |
| Startup delay / efficiency | 3 s / 0.92 |
| Scale-down cooldown / limit | 30 s / 25% per action |
| Forecast window / refit | 180 decisions / 30 s |
| Main / ablation seeds | 100–107 / 200–207 |

### 5.3 Workloads and Queue Model

Each run lasts 7,200 s. A base rate combines a 1,800 s sinusoid, a 300 s sinusoid, and Gaussian perturbation, then draws one-second Poisson arrivals. The periodic profile contains only this structure. The bursty profile adds a three-state Markov multiplicative process with factors 1.0, 1.55, and 2.65 plus eight Gaussian flash crowds with randomized centers, widths, and amplitudes. The regime profile adds a 1.55× demand interval and a later 0.72× shift. Rates are clipped to 20–1,500 requests/s. Main experiments use seeds 100–107; ablations use 200–207.

The three profiles are stress tests rather than estimates of a particular provider's request volume. Their role is to isolate controller behavior under smooth recurrence, transient bursts, and persistent distribution shifts while retaining stochastic arrivals. Every random draw is generated from NumPy's documented generator with an explicit seed. This design enables exact workload regeneration and paired comparisons, but external validity must come from future replay or deployment studies.

The simulator uses a fluid backlog update and an M/M/c waiting-time tail. Effective per-replica capacity is 0.92 times the inverse measured mean latency, accounting for non-inference overhead. Crucially, violation probability is not assigned a fixed percentile floor. The empirical service-time distribution is represented by 25 quantile points and combined with the Erlang-C waiting probability. For each service sample s and deterministic backlog delay $d_b$, the conditional violation contribution is one if $d_b + s \geq L$; otherwise it is $C(c,\rho)\exp[-(c\mu-\lambda)(L-d_b-s)]$. The reported request-weighted mean is computed from all one-second intervals (Figure 2).

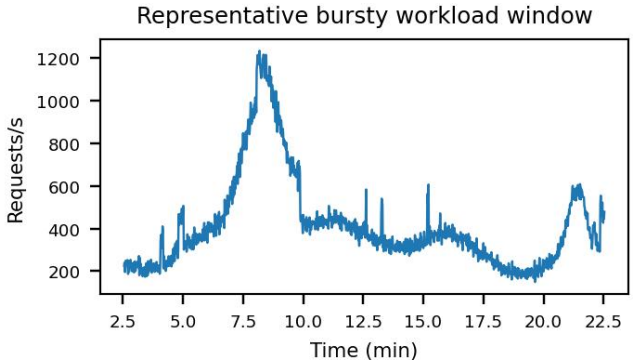


**Figure 2** The Highest-Volume 20-Min Window of the Representative Bursty Trace (Seed 100)

### 5.4 Baselines and Metrics

Reactive-HPA provisions from the last 5 s mean rate at 65% target utilization. Queue-KEDA adds a backlog-drain term and uses a 70% target, approximating queue-triggered event scaling. DES-Predictive applies Holt-style double exponential smoothing with a trend forecast and queue term [25]. All methods share replica bounds, startup delay, and scale-down constraints. Metrics include request-weighted SLO violation rate, SLO attainment, active replica-seconds, request-weighted estimated p95 latency, maximum backlog, scale actions, forecast MAE, and empirical upper-bound coverage.

Eight paired seeds allow each autoscaler to see identical arrivals. We apply a one-sided Wilcoxon signed-rank test to the hypothesis that CAPA has lower violation rate than Reactive-HPA. Because n = 8 is modest, we also report 10,000-sample paired bootstrap intervals for relative effects. The resampling unit is the seed-level paired difference, not an individual second, which avoids treating temporally correlated observations as independent evidence. No multiple-comparison correction is claimed because the two nonstationary profiles test the primary directional hypothesis and the periodic profile is reported as a boundary case. These tests quantify repeatability across generated profiles; they do not substitute for a production-cluster study.

## 6 RESULTS

### 6.1 Bursty and Regime-Shifting Traffic

On bursty traffic, CAPA lowers the violation rate from 3.255% for Reactive-HPA to 2.674%, a relative reduction of 17.83%. The paired bootstrap interval is [13.63%, 21.94%], and the one-sided Wilcoxon p-value is 0.0039. Mean replicas increase from 3.175 to 3.257 (+2.59%), while request-weighted estimated p95 latency decreases from 59.25 ms to 48.48 ms (18.18% reduction). CAPA also cuts violations by 47.12% relative to Queue-KEDA. The trade is not uniformly favorable: mean maximum backlog is 9.74% higher than HPA because CAPA occasionally delays scale-down recovery after a sharp forecast miss. The request-weighted violation metric still improves because the larger backlog peaks occur in fewer intervals (Table 3).

**Table 3** Main Results: Mean of Eight Paired Runs

| P | Method | Viol. % | Replicas | p95 ms |
|---|---|---|---|---|
| B | CAPA | 2.67 | 3.26 | 48.48 |
| B | HPA | 3.25 | 3.17 | 59.25 |
| B | DES | 3.44 | 3.19 | 60.59 |
| B | KEDA | 5.06 | 2.96 | 93.87 |
| R | CAPA | 2.63 | 3.42 | 51.18 |
| R | HPA | 2.97 | 3.36 | 55.90 |
| R | DES | 3.25 | 3.36 | 62.94 |
| R | KEDA | 4.65 | 3.14 | 92.25 |
| P | CAPA | 2.74 | 2.50 | 20.56 |
| P | HPA | 2.39 | 2.60 | 19.67 |
| P | DES | 2.43 | 2.59 | 19.74 |
| P | KEDA | 3.67 | 2.42 | 23.09 |

Under regime shifts, CAPA reduces violations from 2.965% to 2.628%, or 11.37%, with a bootstrap interval of [4.83%, 17.85%] and p = 0.0078. It uses 1.79% more replicas. Estimated p95 latency falls by 8.45% on average, although its bootstrap interval [−1.99%, 19.70%] crosses zero, so that secondary effect is not conclusive. These results support the controller's central design premise: startup-horizon headroom is most useful when the recent past remains informative but the workload is not stationary (Figure 3).

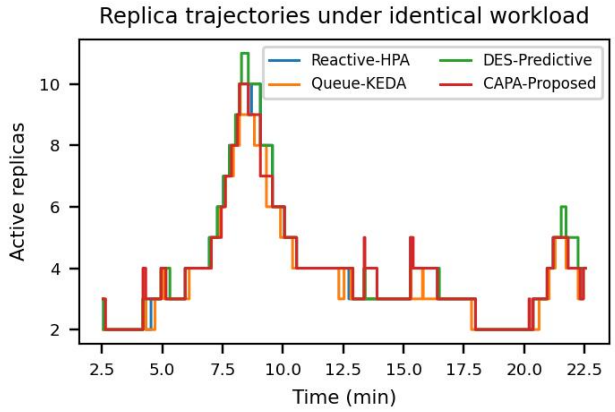


**Figure 3** Active Replicas for Identical Bursty Arrivals; CAPA Scales Earlier Around Several Peaks

### 6.2 Stable Periodic Traffic: A Negative Result

CAPA is not the best controller on the periodic profile. It uses 2.503 replicas on average, 3.82% fewer than HPA, but raises violation rate from 2.388% to 2.736% (+14.55%) and raises estimated p95 latency by 4.54%. The regular workload favors a direct reactive estimate because its short-term rate is already smooth, while CAPA's residual margin and lower replica count can leave insufficient headroom near deterministic crests. The Wilcoxon p-value for CAPA being better is 1.0. This failure case is operationally useful: confidence-aware prediction should be enabled selectively or paired with an online policy gate when forecast protection costs more than reactive control (Figure 4).

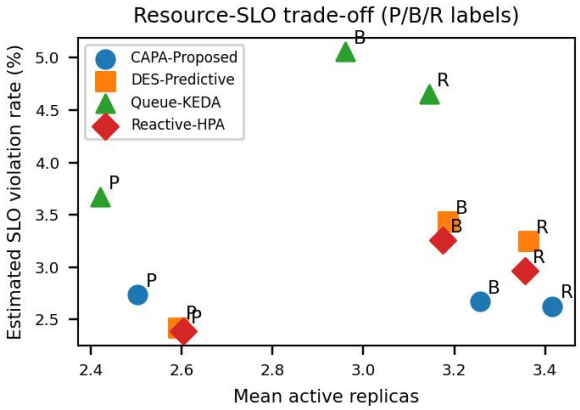


**Figure 4** Resource–SLO Points for Periodic (P), Bursty (B), and Regime (R) Profiles

### 6.3 Forecast Coverage and Control Behavior

CAPA's one-sided upper estimate achieves 88.72% empirical coverage on bursty traffic, 89.19% on regime shifts, and 88.97% on the periodic profile, close to but below the nominal 90% residual quantile. DES upper coverage remains near 45% because it has no explicit uncertainty margin, while the reactive and queue baselines are near 50%. CAPA's forecast MAE is not always the smallest; its advantage arises from asymmetric protection, not point-error minimization. It also performs approximately as many scale actions as HPA on bursty and regime traffic, indicating that the improvement does not come from extreme oscillation (Table 4).

**Table 4** CAPA Relative to Reactive-HPA (Positive Reduction Is Better)

| Profile | Viol. Δ% [95% CI] | Rep. Δ% | p95 Δ% | p |
|---|---|---|---|---|
| Bursty | 17.83 [13.63,21.94] | 2.59 | 18.18 | 0.0039 |
| Regime | 11.37 [4.83,17.85] | 1.79 | 8.45 | 0.0078 |
| Periodic | -14.55 [-16.82,-12.31] | -3.82 | -4.54 | 1.0000 |

### 6.4 Ablation Study

**Table 5** Ablation on Bursty Traffic

| Variant | Viol. % | Replicas | p95 ms | Cov. % |
|---|---|---|---|---|
| CAPA | 2.94 | 3.26 | 54.84 | 88.4 |
| CAPA-no-Debt | 3.16 | 3.22 | 59.46 | 88.4 |
| CAPA-no-UQ | 4.63 | 3.06 | 85.17 | 67.6 |

Removing SLO debt raises the mean violation rate from 2.943% to 3.164% and p95 latency from 54.84 ms to 59.46 ms while saving only 1.38% of replicas (Table 5). Removing the uncertainty margin is more damaging: violation rate reaches 4.629%, p95 latency reaches 85.17 ms, and empirical coverage falls from 88.4% to 67.6%. Relative to CAPA-no-UQ, the complete controller reduces violations by 36.42% with 6.81% more replicas. Thus, prediction alone is not the main contribution; the useful behavior comes from combining an asymmetric upper bound with feedback from service quality (Figure 5).

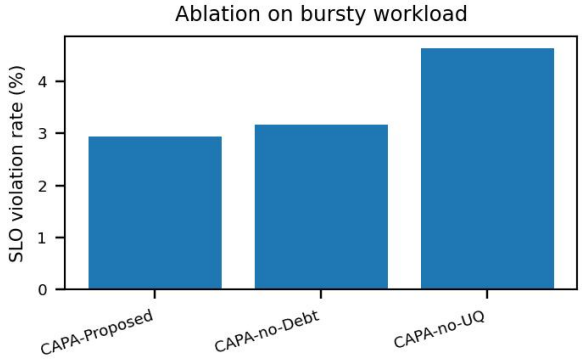


**Figure 5** Both Uncertainty Protection and SLO Debt Contribute to Burst Resilience

### 6.5 Residual-Quantile Sensitivity

**Table 6** CAPA Sensitivity to the One-Sided Residual Quantile

| Profile | q | Viol. % | Replicas | Cov. % |
|---|---|---|---|---|
| Bursty | 0.75 | 3.81 | 3.22 | 78.2 |
| Bursty | 0.90 | 2.73 | 3.36 | 88.5 |
| Bursty | 0.97 | 1.42 | 3.73 | 95.0 |
| Regime | 0.75 | 3.54 | 3.39 | 78.6 |
| Regime | 0.90 | 2.64 | 3.54 | 88.6 |
| Regime | 0.97 | 1.37 | 3.90 | 95.1 |
| Periodic | 0.75 | 3.05 | 2.47 | 78.7 |
| Periodic | 0.90 | 2.66 | 2.51 | 89.3 |
| Periodic | 0.97 | 2.13 | 2.59 | 95.7 |

A separate eight-seed sweep (seeds 300–307) changes only the residual quantile (Table 6). Raising q from 0.75 to 0.97 reduces mean violation rate from 3.81% to 1.42% on bursty traffic and from 3.54% to 1.37% under regime shifts, while mean replicas rise from 3.22 to 3.73 and from 3.39 to 3.90. Coverage increases from approximately 78% to 95%. The periodic profile shows the same monotone protection–capacity trade-off. Thus, q is an operator-facing risk knob rather than a universally optimal constant; q = 0.90 was selected before the main comparison as a moderate operating point, not chosen retrospectively to maximize one profile.

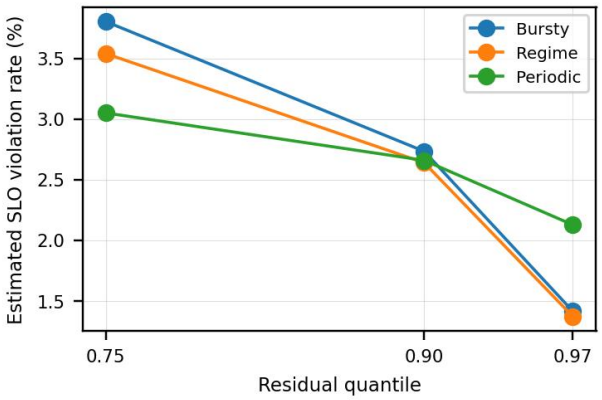


**Figure 6** Higher One-Sided Residual Quantiles Reduce Violations at the Cost of Additional Replicas

## 7 DISCUSSION

### 7.1 Why CAPA Helps

Reactive controllers observe demand after it consumes capacity. CAPA shifts part of the decision earlier by forecasting over startup delay, and the upper margin makes underprediction more expensive than overprediction. Debt then compensates for misspecified service capacity, imperfect queue assumptions, and temporally clustered slow requests.

The mechanism is analogous to integral feedback: persistent quality error lowers the utilization target until the service recovers. The resulting policy is more interpretable than directly learning replica counts because every term has an operational meaning.

### 7.2 When CAPA Should Be Disabled or Adapted

The periodic result shows that prediction is not automatically superior. A deployment should track the marginal value of the upper bound: if forecast protection repeatedly increases violations or cost relative to a shadow reactive policy, the controller can reduce its quantile, shorten its window, or fall back to HPA. Very long startup delays may require richer models, while sub-second serverless activation may make prediction unnecessary. Large language models add token-dependent service times, batching, and KV-cache constraints [12, 15]; in that setting CAPA's arrival forecast would need to predict work units such as tokens or batch-seconds rather than requests.

### 7.3 Interaction with the Serving Backend

CAPA is deliberately backend-agnostic, but its service-rate input is not immutable. Dynamic batching, GPU sharing, model colocation, and memory pressure can change the effective request capacity of a replica. A practical deployment should periodically refresh $\mu$ from completed-request telemetry and maintain separate profiles for material configuration changes such as batch size, device type, or model version. The debt loop provides short-term correction while a new profile is learned, but it cannot replace admission control when a node is saturated by unrelated tenants. Integrating backend signals from systems such as PRETZEL, Nexus, or a GPU scheduler would therefore improve capacity estimation without changing CAPA's forecast-and-feedback structure [21-23].

### 7.4 Threats to Validity

First, this is a prototype-driven simulation, not a live Kubernetes cluster. Network overhead, container image pulling, node scheduling, CPU throttling, and model-server batching are summarized by measured service time, startup delay, and a 0.92 efficiency factor. Second, the workload profiles are synthetic and trace-calibrated in shape, not direct Azure trace replays. Third, the queue model assumes Poisson arrivals within each second and an M/M/c waiting tail, although service-time violation is integrated over the measured empirical distribution. Fourth, only one CPU vision model is benchmarked. Fifth, eight seeds provide paired evidence but limited power for small effects. These restrictions narrow the claim to the evaluated control setting; they do not invalidate the reported generated data.

### 7.5 Reproducibility

The artifact contains the benchmark script, simulator, workload generator, sensitivity, plot and statistics scripts, raw per-seed outputs, representative one-second traces, configuration JSON, measured latency samples, and reference-verification table. Random seeds are explicit. Running the benchmark again can change service parameters because it measures the new host; rerunning only the simulator with the included benchmark JSON reproduces the packaged results. Summary tables and figures are generated from CSV outputs, and no result in the paper is hard-coded in the experiment scripts. A syntax check and a one-seed smoke run are included in the final validation log.

## 8 CONCLUSION

CAPA demonstrates that a small, auditable predictor can improve SLO-aware replica control when it is paired with uncertainty protection and closed-loop quality feedback. In the measured MobileNetV3-Small setting, CAPA reduces estimated request violations on bursty and regime-shifting traffic with a low single-digit replica premium, but it underperforms a reactive baseline on smooth periodic demand. The mixed result is the main design lesson: prediction should be treated as conditional operational leverage, not as a universal replacement for feedback control. Future work should validate the controller in Kubernetes, learn the policy gate online, and extend the capacity model to dynamic batching, GPUs, and token-based generative inference.

## COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

## REFERENCES

[1] Dean J, Barroso L A. The tail at scale. Communications of the ACM, 2013, 56(2): 74-80. DOI: 10.1145/2408776.2408794.

[2] Verma A, Pedrosa L, Korupolu M R, et al. Large-scale cluster management at Google with Borg. In: Proceedings of EuroSys, 2015, 18: 1-17. DOI: 10.1145/2741948.2741964.

[3] Burns B, Grant B, Oppenheimer D, et al. Borg, Omega, and Kubernetes. ACM Queue, 2016, 14(1): 70-93. DOI: 10.1145/2890784.

[4] Crankshaw D, Wang X, Zhou G, et al. Clipper: A low-latency online prediction serving system. In: Proceedings of NSDI, 2017: 613-627.
[5] Olston C, Li F, Harmsen J, et al. TensorFlow-Serving: Flexible, high-performance ML serving. In: Workshop on ML Systems at NIPS, 2017.
[6] Gujarati A, Elnikety S, He Y, et al. Swayam: Distributed autoscaling to meet SLAs of machine learning inference services with resource efficiency. In: Proceedings of Middleware, 2017: 109-120. DOI: 10.1145/3135974.3135993.
[7] Zhang C, Yu M, Wang W, et al. MArk: Exploiting cloud services for cost-effective, SLO-aware machine learning inference serving. In: Proceedings of USENIX ATC, 2019: 1049-1062.
[8] Crankshaw D, Sela G E, Mo X, et al. InferLine: Latency-aware provisioning and scaling for prediction serving pipelines. In: Proceedings of ACM SoCC, 2020: 477-491. DOI: 10.1145/3419111.3421285.
[9] Gujarati A, Karimi R, Alzayat S, et al. Serving DNNs like Clockwork: Performance predictability from the bottom up. In: Proceedings of OSDI, 2020: 443-462.
[10] Romero F, Li Q, Yadwadkar N J, et al. INFaaS: Automated model-less inference serving. In: Proceedings of USENIX ATC, 2021: 397-411.
[11] Gunasekaran J R, Mishra C S, Thinakaran P, et al. Cocktail: A multidimensional optimization for model serving in cloud. In: Proceedings of NSDI, 2022: 1041-1057.
[12] Yu G I, Jeong J S, Kim G W, et al. Orca: A distributed serving system for transformer-based generative models. In: Proceedings of OSDI, 2022: 521-538.
[13] Li Z. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In: Proceedings of OSDI, 2023: 663-679.
[14] Zhang H, Tang Y, Khandelwal A, et al. SHEPHERD: Serving DNNs in the wild. In: Proceedings of NSDI, 2023: 787-808.
[15] Kwon W, Li Z, Zhuang S, et al. Efficient memory management for large language model serving with PagedAttention. In: Proceedings of SOSP, 2023: 611-626. DOI: 10.1145/3600006.3613165.
[16] Lorido-Botran T, Miguel-Alonso J, Lozano J A. A review of auto-scaling techniques for elastic applications in cloud environments. Journal of Grid Computing, 2014, 12(4): 559-592. DOI: 10.1007/s10723-014-9314-7.
[17] Cortez E, Bonde A, Muzio A, et al. Resource Central: Understanding and predicting workloads for improved resource management in large cloud platforms. In: Proceedings of SOSP, 2017: 153-167. DOI: 10.1145/3132747.3132772.
[18] Rzadca K. Autopilot: Workload autoscaling at Google. In: Proceedings of EuroSys, 2020, 16: 1-16. DOI: 10.1145/3342195.3387524.
[19] Qiu H, Banerjee S S, Jha S, et al. FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices. In: Proceedings of OSDI, 2020: 805-825.
[20] Zhang Y, Hua W, Zhou Z, et al. Sinan: ML-based and QoS-aware resource management for cloud microservices. In: Proceedings of ASPLOS, 2021: 167-181. DOI: 10.1145/3445814.3446693.
[21] Lee Y, Scolari A, Chun B G, et al. PRETZEL: Opening the black box of machine learning prediction serving systems. In: Proceedings of OSDI, 2018: 611-626.
[22] Shen H, Chen L, Jin Y, et al. Nexus: A GPU cluster engine for accelerating DNN-based video analysis. In: Proceedings of SOSP, 2019: 322-337. DOI: 10.1145/3341301.3359658.
[23] Xiao W. Gandiva: Introspective cluster scheduling for deep learning. In: Proceedings of OSDI, 2018: 595-610.
[24] Delimitrou C, Kozyrakis C. Quasar: Resource-efficient and QoS-aware cluster management. In: Proceedings of ASPLOS, 2014: 127-144. DOI: 10.1145/2541940.2541941.
[25] Holt C C. Forecasting seasonals and trends by exponentially weighted moving averages. International Journal of Forecasting, 2004, 20(1): 5-10. DOI: 10.1016/j.ijforecast.2003.09.015.
[26] Hyndman R J, Athanasopoulos G. Forecasting: Principles and Practice. 3rd ed. Melbourne, Australia: OTexts, 2021.
[27] Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning. 2nd ed. New York, NY, USA: Springer, 2009. DOI: 10.1007/978-0-387-84858-7.
[28] Gibbs I, Candès E. Adaptive conformal inference under distribution shift. Advances in Neural Information Processing Systems, 2021, 34: 1660-1672.
[29] Shahrad M. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In: Proceedings of USENIX ATC, 2020: 205-218.