# Constrained Hybrid Soft Actor–Critic (CH-SAC) with a Mobility- and Load-Aware Fusion Predictor

by

Sima BEIGALI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR A MASTER'S DEGREE
WITH THESIS IN ELECTRICAL ENGINEERING
M.A.Sc.

MONTREAL, JUNE 20, 2026

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

**BOARD OF EXAMINERS**

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Dr. Michel Kadoch, Thesis Supervisor
Department of Electrical Engineering, École de technologie supérieure

Dr. Rami Langar, President of the Board of Examiners

Department of Software and IT Engineering, École de technologie supérieure

Dr. David Bensoussan, Member of the jury

Department of Software Engineering and IT, École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC

ON APRIL 28, 2026

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

## ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis supervisor, Professor Michel Kadoch, for his guidance, expertise, and continuous support throughout the course of this research. His insightful feedback, academic rigor, and encouragement were essential to the development and successful completion of this thesis.

I would also like to acknowledge the support provided by École de technologie supérieure (ÉTS), which offered a stimulating academic environment and the necessary resources to carry out this research.

I am deeply grateful to my parents for their unconditional support, patience, and encouragement throughout my academic journey. Their constant belief in me provided the motivation and strength needed to overcome challenges and complete this work.

Finally, I would like to thank my friends for their understanding, encouragement, and moral support during this process. Their presence and support made this journey more meaningful and manageable.

**CH-SAC (soft actor–critic hybride contraint) avec un prédicteur de fusion tenant compte de la mobilité et de la charge**

Sima BEIGALI

## RÉSUMÉ

L'informatique en brouillard (Fog computing) fournit une couche d'exécution distribuée pour les applications sensibles à la latence et pilotées par la mobilité. Cependant, son efficacité est limitée par une forte variabilité spatio-temporelle de la demande, des arrivées de trafic en rafales et des rétroactions de congestion. Dans des environnements urbains réalistes, les contrôleurs réactifs et les prédicteurs basés uniquement sur la mobilité réagissent souvent seulement après la formation des files d'attente, ce qui entraîne une congestion persistante, une augmentation de la latence de queue (tail latency), des pertes de paquets et une dégradation de la Qualité de Service (QoS). Ces limitations deviennent particulièrement sévères en présence de goulots d'étranglement de calcul, de capacités de tampon limitées et de points chauds de demande se déplaçant rapidement.

Cette thèse propose CH-SAC avec un prédicteur de fusion sensible à la mobilité et à la charge pour un provisionnement proactif des ressources du brouillard. Le cadre combine une tête de mobilité Flow-GRU avec une tête de demande de charge offerte de type Holt qui opère sur la somme des requêtes acceptées et rejetées, puis fusionne ces informations au moyen d'un pondération adaptative sensible à la congestion avec des bornes de sécurité explicites (plancher et plafond). Les signaux prédictifs obtenus sont intégrés dans un contrôleur Constrained Hybrid Soft Actor–Critic (CH-SAC) soutenu par un mécanisme de garde qui surveille la pression des files d'attente, l'utilisation des ressources, les pics de latence et le taux de rejet. Le cadre est évalué à l'aide de traces réelles de mobilité issues du jeu de données Beijing T-Drive, à la fois dans un environnement YAFS piloté par traces et dans un banc d'émulation de brouillard basé sur Mininet, sous des conditions de stress CPU, de stress de file d'attente et de situations de foule soudaine / déplacement de point chaud (flash-crowd / hotspot-shift).

Le cadre proposé forme une boucle de contrôle en ligne fermée dans laquelle la charge de travail induite par la mobilité est prévue à court horizon, les nœuds de point chaud sont priorisés de manière proactive et des actions continues d'ajustement des ressources sont émises avant que la congestion ne devienne dominante. Dans l'analyse comportementale sous YAFS, le prédicteur de fusion atteint une forte précision à court horizon (MAE = 3,45 taxis, corrélation de Pearson $r \approx 0,95$). Selon une définition d'action proactive basée sur un redimensionnement effectué avant l'apparition d'une surcharge, le contrôleur atteint une proportion d'actions proactives de 78,2 %, un temps d'anticipation médian de 17,0 s et un score F1 de prédiction

de surcharge de 0,890. Dans l'évaluation sous contrainte avec Mininet, CH-SAC avec prédicteur de fusion atteint une latence moyenne de 79,3 ms et 241 454 requêtes complétées sous stress de file d'attente, et une latence moyenne de 41,65 ms avec 162 285 requêtes complétées sous stress de foule soudaine / déplacement de point chaud. À travers ces deux niveaux d'évaluation, le cadre améliore le provisionnement proactif, la latence, le contrôle de la congestion, la fiabilité du service et le comportement de récupération par rapport aux approches basées sur des seuils, aux méthodes métaheuristiques, aux méthodes RL de base et au SAC non prédictif. Les résultats mettent également en évidence les rôles complémentaires des deux principales améliorations du cadre : le prédicteur de fusion est particulièrement important lorsque l'anticipation précoce est nécessaire, notamment dans les conditions de stress CPU et de file d'attente, tandis que le mécanisme de garde est essentiel pour la robustesse et la survie du système face aux situations soudaines de foule massive.

Globalement, cette thèse montre que l'intégration d'une prévision fusionnée à court horizon avec un apprentissage par renforcement à contrôle continu et conscient des contraintes transforme le provisionnement des ressources du brouillard, passant d'une réponse réactive à la surcharge vers un contrôle proactif, résilient et orienté QoS dans des environnements urbains pilotés par la mobilité.

**Mots-clés :** informatique en périphérie (fog computing) ; provisionnement proactif des ressources ; contrôle sensible à la mobilité ; Soft Actor–Critic (SAC) ; apprentissage profond par renforcement ; prédiction de la demande à court horizon ; prédicteur de fusion ; détection de hotspot ; qualité de service (QoS) ; latence en queue de distribution (p95) ; simulation YAFS ; émulation Mininet ; jeu de données taxis Beijing T-Drive.

**Constrained Hybrid Soft Actor–Critic (CH-SAC) with a Mobility- and Load-Aware Fusion Predictor**

SIMA BEIGALI

## ABSTRACT

Fog computing provides a distributed execution layer for latency-sensitive and mobility-driven applications, but its effectiveness is constrained by strong spatiotemporal demand variability, bursty arrivals, and congestion feedback. In realistic urban environments, reactive controllers and mobility-only predictors often respond only after queues have formed, causing sustained congestion, inflated tail latency, packet drops, and Quality-of-Service (QoS) degradation. These limitations are especially severe under compute bottlenecks, constrained buffering, and rapidly shifting demand hotspots.

This thesis proposes CH-SAC with a Mobility- and Load-Aware Fusion Predictor for proactive fog-resource provisioning. The framework combines a Flow-GRU mobility head with a Holt-style offered-load demand head that operates on accepted plus dropped requests, and fuses them through adaptive congestion-aware weighting with explicit floor and cap safety bounds. The resulting predictive signals are integrated into a Constrained Hybrid Soft Actor–Critic (CH-SAC) controller supported by a guard mechanism that monitors queue pressure, utilisation, latency spikes, and drop ratio. The framework is evaluated using real mobility traces from the Beijing T-Drive dataset in both a trace-driven YAFS environment and a Mininet-based fog emulation testbed under CPU-stress, queue-stress, and flash-crowd / hotspot-shift conditions.

The proposed framework forms a closed online control loop in which mobility-driven workload is forecast at a short horizon, hotspot nodes are prioritised proactively, and continuous resource-scaling actions are issued before congestion becomes dominant. In the YAFS behavioural analysis, the Fusion Predictor achieves strong short-horizon accuracy (MAE = 3.45 taxis, Pearson correlation $r \approx 0.95$). Under a proactive-action definition based on scaling before overload onset, the controller attains a proactive-action share of 78.2%, a median lead time of 17.0 s, and an overload-prediction F1-score of 0.890. In the Mininet stress evaluation, CH-SAC with Fusion Predictor achieves 79.3 ms mean latency and 241,454 completed requests under queue stress, and 41.65 ms mean latency with 162,285 completed requests under flash-crowd / hotspot-shift stress. Across the two evaluation layers, the framework improves proactive provisioning, latency, congestion control, service reliability, and recovery behaviour relative to threshold-based, metaheuristic, baseline RL, and non-predictive SAC baselines. The results also clarify the complementary roles of the framework's two main enhancements: the Fusion Predictor is especially important when early

anticipation is required, particularly under CPU- and queue-stress conditions, while the guard is essential for robustness and survivability under abrupt flash-crowd stress.

Overall, the thesis shows that integrating fused short-horizon forecasting with constraint-aware continuous-control reinforcement learning transforms fog-resource provisioning from reactive overload response into proactive, resilient, and QoS-oriented control in mobility-driven urban environments.

**Keywords:** Fog computing; proactive resource provisioning; mobility-aware control; Soft Actor–Critic (SAC); deep reinforcement learning; short-horizon demand prediction; fusion predictor; hotspot detection; Quality of Service (QoS); tail latency (p95); YAFS simulation; Mininet emulation; Beijing T-Drive taxi dataset.

# Table of Contents

Page

## LIST OF FIGURES

## LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| ARIMA | AutoRegressive Integrated Moving Average |
| CH-SAC | Constrained Hybrid Soft Actor–Critic |
| CI | Confidence Interval |
| CPU | Central Processing Unit |
| CV | Coefficient of Variation |
| F1 Score | Harmonic Mean of Precision and Recall |
| GA | Genetic Algorithm |
| GRU | Gated Recurrent Unit |
| IoT | Internet of Things |
| IPT | Instructions Per Tick |
| MAE | Mean Absolute Error |
| MAPE | Mean Absolute Percentage Error |
| MDP | Markov Decision Process |
| Mininet | Network Emulation Framework for Software-Defined Networks |
| ML | Machine Learning |
| P95 | 95th Percentile |
| P99 | 99th Percentile |
| PR Curve | Precision–Recall Curve |
| PSO | Particle Swarm Optimization |
| QoS | Quality of Service |
| RL | Reinforcement Learning |
| SAC | Soft Actor–Critic |
| SLA | Service Level Agreement |
| T-Drive | Beijing Taxi Mobility Dataset |

| | |
|---|---|
| TTR | Time to Recovery |
| YAFS | Yet Another Fog Simulator |
| ΔCPU | CPU Scaling Adjustment |

## LIST OF SYMBOLS AND UNITS OF MEASUREMENT

| $Symbol$ | Meaning |
|---|---|
| $A_{exc}$ | Excess violation area |
| $a_{k,t}^{raw}$ | Raw action for focus slot $k$ |
| $a_t^{raw}$ | Full raw action vector |
| $a_t^{exec}$ | Executed action after guard enforcement |
| $\alpha_0$ | Baseline blending coefficient |
| $\alpha_q$ | Queue-pressure sensitivity of the blending factor |
| $\alpha_d$ | Level-update smoothing coefficient |
| $\beta$ | Baseline turnover coefficient |
| $\beta_d$ | Trend-update smoothing coefficient |
| $\beta_{min}$ | Baseline mobility-scaling factor |
| $\beta_s$ | Demand-driven mobility amplification factor |
| $c_{k,t}$ | Current resource vector for focus slot $k$ |
| $c_{min}$ | Minimum confidence floor |
| $CV$ | Coefficient of variation |
| $Composite$ | Composite fairness and resilience score |
| $d_{k,t}^{med}$ | Median queue delay at focus slot $k$ |
| $Drop\ Rate$ | Ratio of dropped requests to total requests |
| $Delivery\ Ratio$ | Ratio of successfully completed requests to total requests |
| $\Delta c_{k,t}$ | Resource-change vector applied at step $t$ |
| $\Delta CPU$ | CPU scaling adjustment |
| $F_t$ | Selected focus-node set at time $t$ |

| $Symbol$ | Meaning |
|---|---|
| $F1$ | F1-score |
| $G(\cdot)$ | Overall guard operator |
| $G_{activity}$ | Activity-gating operator |
| $G_{bind}$ | Focus-node binding operator |
| $G_{bounds}$ | Bounds-enforcement operator |
| $G_{forecast}$ | Forecast-aware scaling operator |
| $G_{queue}$ | Emergency queue-override operator |
| $G_{sla}$ | SLA guardrail operator |
| $\gamma(d_{k,t}^{med})$ | Queue-emergency scale factor |
| $K$ | Number of focus nodes |
| $\kappa_0$ | Baseline upper-reference coefficient |
| $\kappa_f$ | Upper cap relative to the clipped mobility signal |
| $\kappa_q$ | Queue-pressure sensitivity of the upper reference |
| $L(t)$ | Smoothed average end-to-end latency |
| $L_{SLA}$ | SLA latency threshold |
| $MAE$ | Mean absolute error |
| $MAPE$ | Mean absolute percentage error |
| $P95$ | 95th-percentile latency |
| $P99$ | 99th-percentile latency |
| $\psi_{i,t}$ | Node-priority score |
| $Queue\_Pressure$ | Normalized queue occupancy / queue stress |
| $r$ | Pearson correlation coefficient |

| $Symbol$ | Meaning |
|---|---|
| $r_t$ | Reward at time $t$ |
| $S_i$ | Node-specific calibration factor |
| $s_t$ | Aggregated CH-SAC state vector |
| $\rho$ | Flow-decay factor in the mobility graph |
| $SLA\ Risk$ | SLA-related violation score / risk metric |
| $Throughput$ | Completed requests received at sink |
| $t_{rec}$ | Recovery time |
| $TTR$ | Time to recovery |
| $\nu_p, \nu_q, \nu_u$ | Priority-score weights for prediction, queue pressure, and utilization |
| $V$ | Set of all fog nodes |
| $w_{min}, w_{max}$ | Lower and upper bounds on fusion weights |
| $x_{k,t}^{focus}$ | Slot-level focus-node descriptor |
| $\xi_t$ | Runtime state inspected by the guard |
| $\eta_0$ | Baseline lower-bound coefficient |
| $\eta_q$ | Queue-pressure sensitivity of the lower bound |
| $\eta_q^{load}$ | Queue-pressure amplification coefficient in the load-aware forecast |
| $\theta_0$ | Baseline demand-floor coefficient |
| $\theta_q$ | Queue-pressure sensitivity of the demand floor |
| $\phi$ | Flow-normalization constant |
| $\chi$ | Mobility-clipping multiplier |
| $\omega_0$ | Base demand-weight coefficient in fusion |
| $\omega_a$ | Offered-load influence weight in the anchor forecast |
| $\omega_q$ | Queue-pressure sensitivity in fusion weighting |
| $\omega_s$ | Demand-strength sensitivity in fusion weighting |

# INTRODUCTION

Fog and edge computing extend centralized cloud services toward end users by distributing computation, storage, and networking resources across gateways, roadside units, and micro–data centers near where data are generated and consumed. This architectural shift is especially important for latency-sensitive, location-aware, and mobility-driven applications such as intelligent transportation systems, connected vehicles, real-time navigation, and large-scale urban sensing. In such environments, service quality depends not only on total available capacity but also on how quickly and accurately resources can be placed, scaled, and adapted across heterogeneous and resource-constrained fog nodes.

Urban mobility makes this control problem particularly difficult. As vehicles and users move through the city, the spatial distribution of requests changes continuously, creating short-lived hotspots, shifting bottlenecks, and strong non-stationarity in both computational demand and network traffic. Under these conditions, static placement, periodic re-optimization, and purely reactive controllers often respond only after congestion has already formed. The result is rapid queue growth, heavy latency tails, packet drops, and degraded Quality of Service (QoS). Effective fog-resource management therefore requires controllers that can anticipate near-future demand and act before overload becomes visible.

This thesis addresses that challenge through a predictive and constraint-aware fog-control framework that combines a Mobility- and Load-Aware Fusion Predictor with Constrained Hybrid Soft Actor–Critic (CH-SAC). Unlike mobility-only forecasting approaches, the proposed predictor combines a flow-aware recurrent mobility component with a demand component that models offered load, including both accepted and dropped requests, so that hidden demand remains visible even during congestion. An adaptive fusion mechanism then balances mobility and demand evidence according to real-time system stress and applies explicit safety bounds

to prevent both under-prediction and over-reaction. The resulting forecasts are integrated into CH-SAC, a safety-aware extension of Soft Actor–Critic that augments continuous reinforcement learning with guard mechanisms based on queue pressure, utilization, latency spikes, and drop signals, enabling proactive yet stable resource provisioning under highly dynamic conditions.

The proposed framework is developed in a trace-driven environment built from the Beijing T-Drive dataset and the YAFS simulator, providing a realistic platform for studying mobility-aware fog provisioning under real urban movement patterns. To move beyond simulation-only evidence, the thesis also validates the framework in a Mininet-based emulation testbed that maps logical fog nodes to emulated network and compute endpoints and supports controlled stress conditions, including compute bottlenecks, queue-limited overload, and flash-crowd or hotspot-shift scenarios. By combining YAFS-based design with Mininet-based validation, this thesis argues that effective fog control requires both foresight and safety: foresight from accurate short-horizon demand prediction, and safety from reinforcement-learning control that remains robust when overload emerges.

# CHAPTER 1

# RESEARCH FRAMEWORK AND SCOPE

## 1.1 Research Overview

Fog computing has emerged as a practical distributed-computing paradigm for applications that require low latency, locality awareness, and rapid service response. By placing computation, storage, and networking resources closer to end users through gateways, roadside units, and micro data centres, fog architectures reduce dependence on distant cloud infrastructure and improve service delivery for mobility-driven systems such as intelligent transportation, connected vehicles, and large-scale urban sensing. At the same time, fog nodes operate with limited and heterogeneous resources, so capacity must be allocated carefully across space and time to maintain acceptable Quality of Service (QoS).

This problem becomes significantly more difficult in urban environments, where demand is highly dynamic and non-stationary. User mobility continuously changes the spatial distribution of workload, creating moving hotspots, shifting bottlenecks, and rapidly changing communication and processing conditions. In such settings, reactive control is often insufficient: once queue growth, latency spikes, or service degradation become visible, the system may already be operating in overload. Moreover, admitted traffic alone does not always represent the true system pressure, because congestion can hide demand through queue overflow and dropped requests. Effective fog-resource management therefore requires control mechanisms that can interpret both mobility evolution and actual workload pressure before instability forms.

Conventional strategies such as static provisioning, threshold-based scaling, and periodic optimization with metaheuristics such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) provide useful baselines, but they remain limited under fast-changing fog workloads. Reinforcement learning offers a more adaptive alternative, especially in continuous

control settings, yet unconstrained reactive controllers can still respond too late, generate unsafe actions, or exhibit unstable behavior under bursty demand. This thesis addresses that challenge through a unified predictive and constraint-aware control framework in which future workload estimation and online control are designed together rather than treated as separate components.

The proposed framework consists of two tightly coupled methodological components. The first is a Mobility- and Load-Aware Fusion Predictor, whose novelty lies in combining a flow-aware recurrent mobility model with an explicit offered-load estimator through adaptive congestion-aware weighting and safety bounds. The predictor is designed to preserve visibility of true workload pressure even under packet drops and queue overflow, while also tracking spatial demand migration across fog nodes. The second component is Constrained Hybrid Soft Actor–Critic (CH-SAC), a domain-specific reinforcement-learning controller that extends standard SAC into a safety-aware and continuously adapting control policy for fog environments. To improve robustness, CH-SAC is supported by a guard mechanism that monitors queue pressure, utilization, latency spikes, and drop ratio, and injects corrective scaling actions when safety thresholds are exceeded. Together, these components enable proactive, stable, and resource-efficient fog provisioning under highly dynamic conditions.

The thesis evaluates this framework using two complementary experimental platforms. First, it uses a trace-driven fog environment built from the Beijing T-Drive dataset and the YAFS simulator, where real taxi mobility is mapped onto a geographically coherent tiled fog topology and converted into application-level workload. Second, it validates the same control framework in a Mininet-based emulation testbed, where each logical fog node is mapped to a switch–host pair and network behavior emerges through TCLink-based bandwidth and delay control. This dual-platform methodology allows the thesis to study the framework under realistic mobility-driven demand while also testing its behavior under controlled stress conditions such as compute bottlenecks, queue buildup, and hotspot-shift or flash-crowd scenarios.

This thesis investigates how mobility- and load-aware prediction, constraint-aware reinforcement learning, and cross-platform evaluation can be combined to transform fog-resource provisioning from reactive response into proactive, resilient, and QoS-oriented control in real urban mobility settings.

## 1.2 Research Objectives and Questions

The aim of this thesis is to develop a unified framework for proactive and constraint-aware fog-resource provisioning under mobility-driven demand. The framework combines a trace-driven urban-mobility environment, a Mobility- and Load-Aware Fusion Predictor, a Constrained Hybrid Soft Actor–Critic controller, and cross-platform evaluation in both YAFS and Mininet. In this way, the thesis addresses modeling, prediction, control, and validation as parts of one integrated research problem rather than as separate tasks.

To achieve this aim, the thesis pursues the following objectives:

- **Develop a trace-driven, mobility-aware fog experimentation framework.**
  Construct an end-to-end environment in which real mobility traces from the Beijing T-Drive dataset are mapped to geographically distributed fog workload in YAFS, while preserving a consistent logical mapping for Mininet-based emulation. This provides a reproducible basis for studying mobility-driven provisioning under realistic urban demand conditions.
- **Design a Mobility- and Load-Aware Fusion Predictor.**
  Develop a predictive model that combines a Flow-GRU mobility component with a Holt-style offered-load component, then fuses both through congestion-aware weighting and explicit safety bounds. The objective is to ensure that future demand estimates remain informative even under queue buildup, packet drops, hotspot migration, and flash-crowd behavior.
- **Develop a constraint-aware reinforcement-learning controller for fog provisioning.**
  Design CH-SAC as a safety-aware continuous-control model that consumes fused predictive signals, performs proactive resource scaling, and is supported by a guard mechanism that monitors queue pressure, utilization, latency spikes, and drop ratio. The controller should remain stable and effective even when the learned policy alone would react too late under overload.
- **Evaluate and validate the framework across multiple operating regimes.**

Assess the proposed method against threshold-based, heuristic, and non-predictive learning baselines in both YAFS and Mininet, with particular attention to CPU-stress, queue-stress, and flash-crowd or hotspot-shift scenarios. The evaluation should capture not only latency improvement, but also queue stability, request delivery, throughput, robustness, and recovery behavior.

These objectives lead to the following research questions:

- **RQ1 – Predictive modeling:**
  How can mobility evolution, offered-load pressure, queue feedback, and burst behavior be fused into a forecasting signal that remains informative under congestion and spatial demand migration?
- **RQ2 – Predictive and safe control:**
  How can predictive signals be integrated into CH-SAC, together with guard-based constraint enforcement, so that fog resources are scaled proactively without destabilizing the system?
- **RQ3 – Performance under dynamic stress:**
  To what extent does the proposed framework improve latency, congestion control, request delivery, and service stability compared with threshold, GA, PSO, baseline RL, and non-predictive SAC under CPU-stress, queue-stress, and flash-crowd or hotspot-shift conditions?
- **RQ4 – Cross-platform validity and component roles:**
  Do the gains observed in trace-driven YAFS experiments remain visible in Mininet emulation, and what distinct roles are played by the fusion predictor and the guard in robustness, safety, and anticipatory scaling?

## 1.3 Overview of the Proposed Framework and Methodology

To answer the research questions of this thesis, a dual-platform, trace-driven methodology is adopted that combines simulation in YAFS with emulation in Mininet. Real mobility traces from the Beijing T-Drive dataset are preprocessed into time-aligned events and mapped onto a

geographically coherent fog environment. This provides a common workload basis for both behavioural analysis and stress-based validation.

The proposed framework integrates two tightly coupled components. The first is a Mobility- and Load-Aware Fusion Predictor that combines short-horizon mobility evolution with offered-load pressure so that congestion-hidden demand remains visible even during overload. The second is CH-SAC, a safety-aware continuous-control reinforcement-learning controller that uses these predictive signals to prioritise high-pressure regions and adjust fog-side service capacity proactively. A guard mechanism monitors runtime safety indicators and can enforce corrective action when overload escalates too quickly.

These components operate in a closed online loop. Mobility-driven workload generates telemetry, the predictor estimates near-future node pressure, CH-SAC converts current and predicted system conditions into resource-scaling decisions, and the guard preserves stable operation under critical conditions. The same logical control design is then evaluated across both platforms: YAFS is used to study forecast–action coupling, proactive behaviour, and hotspot dynamics, while Mininet is used to validate end-to-end effectiveness under explicit compute, buffering, and network constraints.

The evaluation covers normal mobility-driven operation together with CPU-stress, queue-stress, and flash-crowd / hotspot-shift scenarios. Performance is assessed using latency, queue pressure, drop rate, throughput, delivery ratio, recovery behaviour, and ablation analysis of the individual and combined roles of prediction and guarding. Full platform details, control design, state and action formulation, reward structure, and runtime execution are presented later in Chapters 3 and 4.

## 1.4 Problem Definition and Scope

Fog-resource management in mobility-driven environments is fundamentally an online constrained-control problem. Fog nodes are geographically distributed, resource-limited, and often heterogeneous, while the workloads they serve evolve rapidly over time and space. In urban settings, mobility generates moving hotspots, changing communication paths, and sudden demand surges that make provisioning decisions obsolete very quickly. Under these conditions, static placement, threshold-based scaling, and repeated large-scale optimization are

difficult to apply effectively, not only because the underlying placement problem is computationally hard, but also because the system must react within strict time budgets while maintaining acceptable Quality of Service (QoS).

The problem addressed in this thesis is more demanding than classical one-shot service placement. In the considered setting, application modules are already deployed, and the control task is to continuously adjust fog-tier resource allocations under real mobility-driven demand. The workload is generated from the Beijing T-Drive dataset and mapped to a tiled fog topology over central Beijing, so every control decision is tied directly to realistic spatial demand evolution rather than synthetic traffic assumptions. The controller operates under a limited control domain: it can influence fog-tier resource allocation and local service capacity, but end-to-end QoS also depends on infrastructure elements such as backhaul segments and sink-tier processing that lie outside its direct authority. The decision process must therefore be fine-grained, time-constrained, and explicitly multi-objective, balancing latency, queue stability, SLA compliance, resource efficiency, smoothness of control, and resilience under mobility-induced disturbances.

A central difficulty in this problem is that visible admitted traffic does not always reflect true demand. During congestion, queue overflow and dropped requests can hide workload pressure, causing mobility-only or admission-only controllers to underestimate the actual stress imposed on the system. For this reason, the thesis treats prediction and control as a coupled problem: effective fog provisioning must reason jointly about mobility evolution, offered load, queue pressure, and hotspot migration. This motivates the use of a Mobility- and Load-Aware Fusion Predictor, in which a flow-aware recurrent mobility model is combined with an offered-load demand model through adaptive congestion-aware fusion and explicit safety bounds. The aim is to preserve predictive visibility even during overload, flash crowds, and rapid spatial demand shifts.

Accordingly, the core research problem of this thesis can be stated as follows: how can a predictive and constraint-aware controller be designed to scale fog resources proactively and safely under mobility-driven, non-stationary demand, such that latency, queue buildup, and QoS degradation are reduced relative to reactive, heuristic, and non-predictive learning baselines? To answer this question, the thesis develops a unified framework built around the

Fusion Predictor and Constrained Hybrid Soft Actor–Critic (CH-SAC), supported by a guard mechanism that monitors queue pressure, utilization, latency spikes, and drop ratio. The framework is studied in a trace-driven YAFS environment and validated in a Mininet-based emulation testbed under CPU-stress, queue-stress, and flash-crowd or hotspot-shift conditions, so that both broad behavioral analysis and stress-oriented validation are included within the same research design.

In scope, this thesis focuses on a representative urban intelligent-transportation workload derived from real taxi mobility, with the primary emphasis placed on fog-tier decision-making and its impact on service quality. It does not attempt to redesign the entire network stack or solve global cloud–network optimization jointly. Instead, it investigates whether mobility- and load-aware prediction, constraint-aware reinforcement learning, and cross-platform evaluation can together transform fog-resource provisioning from reactive overload response into proactive, resilient, and practically credible control.

## 1.5 Contributions

This thesis makes the following main contributions:

1. **A unified mobility-driven fog experimentation framework spanning simulation and emulation.**

The thesis develops an end-to-end research framework in which real Beijing T-Drive mobility traces are mapped to a geographically coherent fog environment in YAFS, with workload generation tied directly to realistic application messages and hotspot evolution. The same logical fog setting is also represented in a Mininet testbed through a one-to-one mapping of logical fog nodes to switch–host pairs with TCLink-based link control, enabling consistent cross-platform evaluation rather than isolated simulator-only analysis.

2. **A Mobility- and Load-Aware Fusion Predictor for proactive fog control.**

The thesis introduces a new predictor that combines a Flow-GRU mobility head with a Holt-style offered-load demand head and fuses them through adaptive congestion-aware weighting, scaling, clipping, and explicit safety bounds. By modeling offered load as accepted plus dropped requests, the predictor remains informative even when congestion hides true demand, and by blending spatial mobility intelligence with demand pressure it can track hotspot

migration, flash-crowd behavior, and overload escalation more reliably than mobility-only prediction.

3. **A Constrained Hybrid Soft Actor–Critic (CH-SAC) controller with safety guarding.**

The thesis proposes CH-SAC as a domain-specific extension of Soft Actor–Critic for mobility-driven fog provisioning. The controller is designed for continuous resource control under SLA-critical conditions and is supported by a constraint-aware guard mechanism that monitors queue pressure, utilization, latency spikes, and drop ratio, injecting corrective actions when safety thresholds are violated. This makes the controller not only adaptive but also robust under extreme overload, preventing passive or delayed reactions when congestion escalates rapidly.

4. **An integrated deployment and evaluation methodology for predictive fog control.**

The work embeds the control loop directly into the experimental platforms: in YAFS, the controller is coupled with the simulator's state–action–reward loop and placement/routing process, enabling continuous online adaptation; in Mininet, the same control logic is examined under explicit compute and buffering constraints. The evaluation is organized around both general trace-driven operation and dedicated CPU-stress, queue-stress, and flash-crowd/hotspot-shift scenarios, with ablations that isolate the roles of baseline SAC, guard-only control, and guard-plus-prediction control.

5. **Cross-platform empirical evidence and methodological insight.**

Across YAFS and Mininet, the thesis shows that the proposed framework improves proactive provisioning, latency, congestion control, service reliability, and recovery behavior relative to threshold-based, metaheuristic, baseline RL, and non-predictive SAC baselines. Just as importantly, the experiments clarify the distinct roles of the two main enhancements: the guard is necessary for robustness and survivability under stress, while the Fusion Predictor is essential for strong performance when early anticipation is needed, especially in CPU- and queue-stress regimes. Under flash-crowd stress, the guard dominates safety while prediction contributes additional stabilization and more anticipatory hotspot-aware scaling. This provides not only performance evidence, but also a clearer control interpretation of when and why predictive reinforcement learning helps in fog computing.

## 1.6 Constraints

This thesis is conducted within a set of deliberate study boundaries designed to keep the research problem focused and the evaluation fair. These constraints define the intended scope of the work rather than post hoc weaknesses. They clarify what the proposed framework is designed to address, under what conditions it is evaluated, and how the reported findings should be interpreted.

First, the study focuses on a representative mobility-driven urban workload derived from the Beijing T-Drive dataset. This provides realistic spatiotemporal variation, moving hotspots, and non-stationary demand, but it still reflects a single-city mobility source and one broad application context. The conclusions of the thesis should therefore be interpreted as applying primarily to mobility-aware fog provisioning under urban transport-like demand rather than to every possible fog workload.

Second, the evaluation is performed on a fixed and controlled Mininet–YAFS environment. This design is intentional: it ensures repeatable comparison across controllers and makes it possible to isolate the effect of prediction-guided and constraint-aware control. At the same time, it does not attempt to represent the full heterogeneity of real fog deployments, such as highly asymmetric node capacities, partially disconnected infrastructures, dynamic failures, or full wireless-network variability.

Third, the thesis concentrates on fog-tier resource control rather than full end-to-end network redesign. The controller adjusts node-level service capacity at selected focus nodes through bounded provisioning actions, but it does not redesign routing paths, change physical link capacities, or solve full cloud–network co-optimisation. As a result, the framework can strongly influence the controllable fog layer, while some end-to-end performance components remain outside its direct action space.

Fourth, the proposed predictive-control design is intentionally short-horizon and operational. The Fusion Predictor is designed to support near-term congestion prevention rather than long-range forecasting, and CH-SAC operates on aggregated telemetry together with a focus-node abstraction rather than complete per-flow visibility over the entire system. These design choices are necessary for tractable online control, but they also mean that very fine-grained

anomalies, rare microbursts, and long-range dependencies may be represented only approximately.

Finally, the absolute numerical values reported in later chapters depend on the selected experimental settings. CPU-stress, queue-stress, and flash-crowd / hotspot-shift scenarios are defined using specific queue sizes, worker limits, processing times, and SLA thresholds. For this reason, the thesis places greater emphasis on comparative controller behaviour, cross-scenario robustness, and mechanistic interpretation than on claiming universal absolute values for latency or other performance metrics.

Within these boundaries, the thesis investigates whether mobility- and load-aware forecasting, constraint-aware reinforcement learning, and guard-based safety can be combined to transform fog-resource provisioning from reactive overload response into proactive, resilient, and QoS-oriented control. A fuller discussion of generalisability, deployment realism, and remaining limitations is presented later in Chapter 7.

## 1.7 Thesis Organization

The remainder of this thesis is organised as follows:

Chapter 2 reviews the background and related work on fog and edge computing, mobility-aware resource management, reinforcement learning for systems control, safe RL, and short-horizon demand prediction.

Chapter 3 describes the experimental environment, including the Beijing T-Drive dataset, tiled spatial mapping, workload generation, hotspot induction, and the Mininet-YAFS testbed used to emulate network, compute, and queueing behaviour.

Chapter 4 presents the proposed framework, detailing the Mobility- and Load-Aware Fusion Predictor, the Constrained Hybrid Soft Actor–Critic controller, the observation and action design, the reward structure, and the safety guard that couples prediction with online control.

Chapter 5 analyses the controller’s operational behaviour, with emphasis on hotspot prioritisation, forecast-driven scaling, temporal queue dynamics, and the complementary roles of prediction and constraint guarding.

Chapter 6 provides the quantitative evaluation across CPU-stress, queue-stress, and flash-crowd/hotspot-shift scenarios, and reports latency, queue pressure, throughput, drop rate, SLA compliance, recovery time, and ablation results against the selected baselines.

Chapter 7 discusses the implications and limitations of the proposed approach, outlines directions for future work, and concludes the thesis.

# CHAPTER 2

# LITERATURE REVIEW

## 2.1 Literature Review

Optimal resource management in fog computing is computationally difficult. Many placement, scheduling, and allocation problems can be reduced to NP-hard formulations such as bin packing, partitioning, or scheduling on unrelated machines. As the numbers of tasks and nodes increase, the search space grows combinatorially, making exact methods such as brute-force search or mixed-integer programming impractical except for small or offline cases. Consequently, fog resource management has largely relied on approximate methods, including heuristics, metaheuristics, and, more recently, AI-based approaches. These limitations motivate the study of learning-based and prediction-aware controllers for dynamic urban fog environments.

## 2.2 Heuristic and Metaheuristic Approaches

Heuristic methods use simple rules to construct feasible allocations quickly, for example by assigning tasks to the nearest node, the least-loaded node, or nodes selected through clustering. Their main advantage is low overhead, but their performance often degrades when workloads or topology differ from the assumptions under which they were designed. Greedy decisions may therefore produce locally efficient but globally suboptimal allocations(Goswami et al., 2025).

Metaheuristics improve search quality by exploring a broader solution space (Bachiega et al., 2023). Canali and Lancellotti used a genetic algorithm (GA) for fog service placement and reported faster convergence to near-optimal solutions through chromosome-based encoding and fitness evaluation (Canali & Lancellotti, 2019). Sarrafzadeh proposed a penalty-based GA

that improved response time by balancing exploration with constraint handling (Sarrafzade et al., 2022). Other methods include ant colony optimization, particle swarm optimization, and simulated annealing. Although metaheuristics are flexible and suitable for multi-objective optimisation, they usually require careful hyperparameter tuning and are often too slow for real-time adaptation, especially under mobility. For this reason, heuristics and metaheuristics remain useful baselines, but their speed-quality trade-off has encouraged more adaptive methods (Ghobaei-Arani et al., 2020).

## 2.3 Machine Learning and Reinforcement Learning Approaches

To overcome the limitations of rule-based and search-based methods, fog resource management increasingly relies on machine learning (ML). Supervised models are commonly used to predict latency, workload, or placement quality, while clustering can simplify allocation by grouping similar tasks or nodes. Fuzzy and neuro-fuzzy systems are also used to handle uncertainty, and deep learning models have been applied to recognise workload patterns and support scheduling decisions (Alsadie, 2024).Among these approaches, reinforcement learning (RL) has received particular attention because it learns allocation policies directly through interaction with the environment rather than from labelled optimal decisions (Ghafari & Mansouri, 2025). RL is attractive in fog settings for three reasons: it adapts to changing workloads and network conditions, it does not require labelled training data, and it can optimise long-term performance rather than only immediate outcomes. Qi et al. used a deep Q-network for task offloading and showed improvements over simple heuristics under bursty traffic (Tang & Wong, 2022). Nieto et al. proposed a distributed actor-critic method that improved quality of experience while considering battery level and latency (Nieto et al., 2024).Related work has also applied multi-agent RL to service deployment and deep RL to vehicular fog scheduling. Lera and Guerrero (Lera & Guerrero, 2024) reported millisecond-scale placement decisions, in contrast to the far greater cost of exact optimisation or exhaustive search (Abdelghany, 2025).

Despite these advantages, RL in fog computing still faces several challenges, including large state spaces, reward design, training cost, and non-stationarity in multi-agent settings. Even so,

recent surveys identify ML, and especially deep RL, as promising directions for fog resource management (Ghobaei-Arani et al., 2020).

## 2.4 Graph Neural Networks and Other Advanced Technologies

Recent work has further strengthened fog resource management through graph-based learning and time-series forecasting. Fog systems are naturally represented as graphs, where nodes correspond to computing devices and edges to communication links. Graph neural networks (GNNs) can therefore encode topology, connectivity, and service dependencies more effectively than flat state representations. Lera and Guerrero (Lera & Guerrero, 2024).incorporated a GNN into a deep RL framework for application placement, enabling the model to capture both service structure and fog network state.

Prediction has also become important for proactive resource management. Time-series models such as Holt-Winters, ARIMA, and LSTM are used to forecast workload or resource availability before allocation decisions are made. Behera (Behera et al., 2023) combined Holt-Winters and vector autoregression to predict CPU availability on edge virtual machines, achieving better results than non-predictive approaches. Hybrid methods that combine RL with forecasting, fuzzy logic, or metaheuristics therefore represent a clear trend in the literature. As summarised in Table 2.1, recent research increasingly integrates learning, graph-aware modelling, and prediction to address the dynamic and multi-objective nature of fog resource management.

## 2.5 Hybrid AI Approaches:

Combining these techniques leads to hybrid solutions, such as:

- Fuzzy logic and RL hybrids.
- Learning and metaheuristic combinations.
- DRL augmented with predictions from models like Holt-Winters or LSTMs.

The current trend in fog resource management research is towards intelligent, hybrid solutions that integrate various AI techniques to address the complexity of dynamic, distributed, and multi-metric fog environments.

Table 2.1 provides a high-level comparison of the main model categories discussed in this chapter. The literature shows a clear evolution from static optimisation-based methods toward adaptive learning-based, predictive, and hybrid approaches that are better suited to mobility-driven fog environments.

Table 2.1 Comparison of models

| **Approach Category** | **Techniques** | **Handles Mobility/Dynamics** | **Advantages** | **Limitations** |
|---|---|---|---|---|
| Deterministic | ILP/MIP formulations and exact optimisation methods | No (typically static, small scale) | Optimal or near-optimal solutions for small problems | Computationally infeasible at scale; requires re-solving on changes |
| Heuristics | Greedy algorithms, rule-based assignment, clustering first | Limited (often static assumptions) | Very fast runtime; simple implementation | Suboptimal results; get stuck in local optima; not adaptive |
| Metaheuristics | Genetic Algorithms, PSO, ACO, Simulated Annealing | Partial (can re-run on changes, but slow) | Can find near-optimal solutions; general applicability | High computational cost; many hyperparameters; typically offline use |
| Supervised ML/DL | Neural network–based prediction models | Possibly (via retraining or online learning) | Learns from data; can predict complex patterns | Needs large training data; may not generalize to unseen conditions |

Table 2.1 Comparison of models (continued)

| Approach Category | Techniques | Handles Mobility/Dynamics | Advantages | Limitations |
|---|---|---|---|---|
| Predictive Models | Time-series forecasting models (e.g., Holt–Winters and LSTM) | Yes (anticipates future changes) | Proactive resource allocation; reduces reactive overhead | Forecast errors can mislead decisions; adds extra component |
| Hybrid Approaches | RL + GNN, Forecast+ Metaheuristic, Fuzzy + RL | Yes (designed for dynamic scenarios) | Combines strengths of multiple methods | Increased system complexity; careful integration needed |

## 2.6 Gap Analysis

Despite extensive research, several gaps remain in fog resource management. First, prediction and control are still rarely integrated into a single decision loop; many studies treat forecasting and allocation as separate stages (Goswami et al., 2025), (Alsadie, 2024). Second, maintaining stability and efficiency under large-scale, highly dynamic fog environments remains difficult, especially in mobility-driven scenarios (Goswami et al., 2025), (Ghobaei-Arani et al., 2020). Third, seamless service adaptation under mobility and handoff is still an open problem, since frequent topology changes often require costly re-optimisation or migration (Goswami et al., 2025).

## 2.7 Challenges and open issues in fog computing

Fog computing moves computation and storage closer to the network edge, enabling low-latency and locality-aware services. However, this shift also makes resource management more difficult than in centralised cloud systems. Fog environments are heterogeneous, dynamic,

geographically distributed, and subject to strict QoS constraints. These characteristics make direct application of classical optimisation and scheduling methods difficult, especially at realistic scales (Goswami et al., 2025).

## 2.8 Resource Constraints and Heterogeneity

Fog nodes range from small embedded devices to micro-data centres, and typically provide limited CPU, memory, storage, and bandwidth compared with cloud servers. Resource allocation is therefore constrained not only by node capacity but also by the combined demands of multi-service IoT applications (Goswami et al., 2025) Heterogeneity further complicates management, since nodes differ in hardware, software, and communication capabilities, making uniform placement policies ineffective (Goswami et al., 2025), (Ghobaei-Arani et al., 2020). Although heterogeneity is a central issue in practical fog systems, the present thesis focuses instead on mobility-driven non-stationarity within a controlled baseline topology, so that the effects of the proposed controller can be isolated more clearly.

## 2.9 Dynamism, Mobility, and Uncertainty

Fog environments are inherently dynamic. Nodes may join or leave, wireless link quality may vary, and user or vehicle mobility continuously changes both topology and demand distribution. In transportation scenarios, for example, workload hotspots may appear and disappear within seconds. Under such conditions, static placement or infrequent re-optimisation quickly becomes outdated. Effective resource management must therefore adapt in near real time and, ideally, anticipate short-term changes rather than reacting only after congestion occurs (Goswami et al., 2025), (Alsadie, 2024).

## 2.10 Scalability in geographically distributed fog–cloud systems

Fog systems often span large geographical areas and interact closely with cloud infrastructure. As the numbers of devices, nodes, and applications increase, centralised control becomes difficult to scale. Collecting global state and solving large optimisation problems can exceed the available decision time, resulting in stale or coarse decisions. Scalable management therefore requires distributed or hierarchical control, but designing such architectures while

maintaining coherent global behaviour remains an open challenge (Goswami et al., 2025), (Ghobaei-Arani et al., 2020).

### 2.11 Diverse QoS requirements, metrics, and SLAs

Fog platforms support applications with very different QoS requirements, including real-time control, healthcare monitoring, navigation, AR/VR, and batch analytics. These workloads may prioritise latency, throughput, reliability, locality, or energy efficiency, and these objectives often conflict. Traditional cloud SLAs are therefore insufficient for fog environments, where guarantees must often be made over fine time scales and within specific regions. Fog schedulers must consequently optimise multiple objectives simultaneously rather than a single performance metric (Goswami et al., 2025), (Alsadie, 2024).

### 2.12 Security, trust, and reliability in decentralized fog networks

The decentralised and heterogeneous nature of fog infrastructures increases their attack surface. Edge nodes can be physically accessible, poorly protected, or even malicious. Ensuring secure data processing, protecting privacy, and establishing trust between loosely coupled entities is therefore more difficult than in a controlled cloud data centre.

From a resource-management perspective, this implies that placement decisions cannot be based solely on performance metrics: the “best” node may be unacceptable if it cannot be trusted with sensitive workloads. Schedulers must incorporate security attributes (e.g. node trust level, hardware isolation support) into their decision process and, in some cases, retain redundant replicas or migration strategies to tolerate node failures or compromise (Goswami et al., 2025).

### 2.13 Algorithmic complexity of service placement and scheduling

The optimal placement of services and tasks in a heterogeneous fog–cloud environment has been shown to be NP-hard. Even simplified formulations of the IoT Service Placement Problem (SPP), where each service must be assigned to a node under CPU, memory, and bandwidth constraints, can be reduced to multi-dimensional bin packing, knapsack, or

scheduling on unrelated parallel machine problems that are NP-complete in general (Goswami et al., 2025), (Ghobaei-Arani et al., 2020).

In realistic applications, additional structure further increases complexity. Many IoT services form workflows with explicit dependencies, naturally represented as Directed Acyclic Graphs (DAGs). For such applications, the scheduler must not only decide *where* to place each task but also *in which order* to execute dependent tasks and *how* to route intermediate data. Ignoring these dependencies leads to sub-optimal utilisation and unnecessary delays; modelling them increases the size and dimensionality of the optimisation problem (Goswami et al., 2025).

Because exact Mixed-Integer Programming (MIP) or exhaustive search methods do not scale to large fog deployments or online decision-making, most works resort to heuristics and metaheuristics such as greedy placement, genetic algorithms, particle swarm optimisation, and ant-colony optimisation (Goswami et al., 2025), (Tang & Wong, 2022). These techniques can approximate near-optimal solutions offline but often require long convergence times and extensive parameter tuning. Avoiding local optima and premature convergence, while keeping computation within strict per-decision budgets, is a persistent challenge, especially in highly dynamic scenarios.

## 2.14 Learning-Based Control, GNNs, and the need for Prediction

In response to these limitations, the fog-computing community has increasingly turned to machine learning and, in particular, reinforcement learning (RL) to tackle resource management. RL-based controllers learn policies directly from interaction with the environment, sidestepping the need to solve an explicit optimisation problem at every step. Deep RL methods (DQN, actor–critic, SAC, PPO) can handle high-dimensional state spaces and continuous control, making them natural candidates for fine-grained scaling and placement (Alsadie, 2024), (Ghafari & Mansouri, 2025).

More recently, Graph Neural Networks (GNNs) have been proposed to encode the graph-structured nature of fog topologies and service dependencies. By learning node and edge embeddings, GNN-augmented RL agents can exploit structural information about the service graph and physical network, often achieving better placement decisions than purely tabular or MLP-based approaches. However, these techniques remain computationally heavy; training

GNN-based controllers in real time on resource-constrained nodes is still challenging(Alsadie, 2024).

A recurring insight across surveys is that prediction is a missing but critical ingredient in many existing controllers (Goswami et al., 2025), (Alsadie, 2024). Most current solutions remain largely reactive: they scale or migrate services only after congestion, SLA violations, or QoS degradation are observed.

Given the strong spatio-temporal dynamics of mobility-driven workloads, relying solely on current load leads to oscillations, long tails in latency, and poor utilisation. Accurate short-term forecasting of demand and resource availability, whether via statistical methods (e.g. Holt–Winters), time-series models, or learned predictors, can provide the foresight needed to allocate capacity before hotspots form. Yet fully integrated "forecast + control" solutions are still relatively rare, and many works treat prediction and scheduling as separate modules rather than as a unified control loop (Goswami et al., 2025), (Alsadie, 2024).

In summary, fog-computing resource management faces a combination of constraints: limited and heterogeneous resources; strong dynamism and mobility; large-scale, geographically distributed deployments; diverse and sometimes conflicting QoS requirements; security and trust concerns; and NP-hard optimisation problems that are difficult to solve under tight time budgets. Heuristics and metaheuristics offer partial remedies but struggle with online adaptation and extreme dynamics. Learning-based methods, including RL and GNN-augmented policies, provide adaptivity but are often deployed in reactive, non-predictive settings and evaluated under simplified workloads.

These challenges directly motivate the research problem addressed in this thesis. The strong spatio-temporal variability of urban mobility calls for mobility-aware prediction rather than purely reactive control. The NP-hardness of service placement under realistic constraints argues for learning-based policies instead of repeated combinatorial optimisation. The need for timely decisions in a dynamic fog environment favours continuous-control RL algorithms that can operate within strict per-step time budgets. By embedding a short-horizon mobility predictor into a Soft Actor–Critic controller, and by evaluating this architecture in a trace-driven, city-scale environment, this thesis aims to address precisely these open issues: moving

fog resource management from static, reactive scheduling toward proactive, resilient control that is explicitly aware of mobility, prediction uncertainty, and multi-objective QoS trade-offs.

# CHAPTER 3

# EXPERIMENTAL SETUP AND EVALUATION

## 3.1 Experimental Scope

This chapter describes the experimental environment used to evaluate the proposed predictive and constraint-aware fog-resource management framework. The evaluation pipeline begins with real mobility traces from the Beijing T-Drive dataset, which capture large-scale urban vehicle movements across central Beijing. These traces are spatially mapped onto a tiled fog region and transformed into timestamped service requests representing mobility-driven application activity.

The generated workload is then executed within a fog-computing testbed where network links, compute limits, and queue capacities are explicitly enforced. This controlled environment enables systematic experimentation under repeatable conditions while preserving the temporal and spatial characteristics of the original mobility data. Within this environment, the proposed framework integrates a Mobility- and Load-Aware Fusion Predictor with a Constrained Hybrid Soft Actor–Critic (CH-SAC) controller to dynamically provision fog resources in response to predicted and observed demand.

The purpose of this chapter is to define the workload source, preprocessing pipeline, network environment, and evaluation protocol that are shared across all controller experiments. The internal design of the predictor and the reinforcement-learning controller is described separately in Chapter 4.

## 3.2 Dataset and Preprocessing

The workload used throughout this thesis is derived from the Beijing T-Drive taxi dataset, which provides timestamped GPS trajectories describing large-scale urban vehicle movement across central Beijing. This dataset is used as the common mobility source for both the YAFS simulation layer and the Mininet emulation layer, ensuring that all evaluated controllers are exposed to the same underlying spatiotemporal demand process. In this way, differences in

performance can be attributed to the control policy rather than to differences in workload generation.

Before each experiment, the raw trajectory data are preprocessed into a replayable mobility stream. The Beijing T-Drive records are parsed into timestamped taxi trajectories and converted into discrete mobility events representing client activation, movement, and deactivation. These processed traces are then cached and reused across runs to define a fixed replay horizon for all compared methods. This preprocessing step transforms raw GPS logs into an event-driven workload source that can be replayed consistently across seeds, baselines, and stress scenarios.

To enable controlled experimentation, a one-hour slice of the trace is selected and temporally compressed so that mobility dynamics unfold on an experimental timescale while preserving the relative timing and burst structure of movement events. After preprocessing and temporal compression, the resulting replay stream produces an active mobility period, followed by a short drain period to capture the workload tail and allow in-flight requests and queues to clear. This design preserves moving hotspots and bursty non-stationary arrivals while keeping the experiments tractable and repeatable.

The selected trace corresponds to a geographic window covering central Beijing, approximately 39.85°–40.05° N and 116.25°–116.55° E. This region is later discretised into the tiled fog topology described in the following sections, allowing each taxi position to be mapped consistently to a nearby fog node. The alignment between the mobility trace and the tiled fog fabric preserves the spatial relationship between taxi density, hotspot formation, and computational demand.

In the main evaluation scenarios, workload generation remains entirely trace-driven: As a result, the workload used in this thesis reflects authentic urban demand variation This is important because the aim of the study is not only to measure average performance, but also to evaluate whether the proposed framework can respond proactively and safely to realistic mobility-driven non-stationarity.

## 3.3 Evaluation Runs and Random Seeds

All controllers are evaluated under identical workload traces, network topology, and system configurations. Differences in performance therefore arise solely from the behaviour of the control policy rather than from environmental variability.

To account for stochastic elements of the reinforcement-learning process and the runtime environment, experiments are executed using multiple random seeds. Unless otherwise specified, reported scalar metrics are calculated as the mean across three independent runs using the seeds 1337, 777, and 42. This multi-seed evaluation approach reduces sensitivity to random initialisation and provides a more reliable estimate of controller performance.

For visualisations in which overlaying several runs would reduce clarity, such as time-series plots of system behaviour or spatial workload snapshots, a single representative run may be shown. In such cases the selected run corresponds closely to the multi-seed mean, and the figure caption explicitly indicates that the plot represents one seed.

Prediction diagnostics, when reported separately, may be generated using an additional seed dedicated to forecasting analysis.

## 3.4 Workload Horizon and Experiment Duration

The workload used in this study is derived from the Beijing T-Drive taxi trace. After preprocessing and temporal compression, the resulting replayed workload produces a continuous period of simulated mobility-driven activity.

This horizon represents the interval during which vehicles generate mobility events that trigger application requests. After the mobility phase concludes, the simulation continues for a short additional period to allow in-flight requests and queues to drain, ensuring that system behaviour during the workload tail is also captured.

All controller policies are evaluated under the same replay duration, workload stream, and simulation parameters. This ensures that each controller experiences identical demand conditions, enabling direct comparison of latency, congestion behaviour, throughput, and recovery dynamics.

## 3.5 Stress Scenarios and Reporting Conventions

To evaluate the robustness of fog-resource management under different types of operational pressure, the experimental design includes several controlled stress scenarios. Each scenario emphasizes a different potential bottleneck within the fog system.

CPU stress is introduced by increasing service processing time while restricting parallel processing capacity. This configuration tests the controller's ability to anticipate growing demand and provision sufficient compute resources before queues accumulate.

Queue stress is created by limiting buffer capacity while maintaining bursty arrival patterns. Under this condition, the system must react quickly to prevent queue growth from exceeding available capacity.

Flash-crowd or hotspot-shift stress concentrates a large fraction of requests within a small set of nodes and periodically shifts the spatial location of this concentration. This scenario evaluates how effectively the controller adapts to rapidly changing spatial demand patterns.

Across all scenarios, the underlying workload trace, network topology, routing logic, and service placement remain fixed. Only the stress parameters are varied, ensuring that the comparative performance of different control policies can be evaluated under consistent experimental conditions.

The primary performance metrics reported throughout the thesis include:

mean and tail latency (e.g., P95 latency), queue pressure and congestion indicators, throughput or successfully completed requests, request drop rate and delivery ratio, service-level agreement (SLA) violations, and recovery behaviour following overload conditions.

## 3.6 Data Flow

The experimental framework follows a continuous pipeline linking the preprocessed mobility stream, workload generation, fog-network execution, and adaptive resource control. The process begins with configuration loading and platform initialization and ends with metric extraction and visualisation.

At the start of each experiment, simulation parameters such as topology size, link properties, processing limits, queue capacities, and experiment duration are loaded from configuration

files. The corresponding fog topology and service environment are then initialised, previous outputs are cleared, and the required applications and service modules are deployed.

During runtime, the replayed mobility stream drives a dynamic population model that continuously creates and updates taxi gateway entities. As simulation time advances, these events generate application requests that form the workload stream through the fog infrastructure, producing computational and network load on the nodes.

Execution then proceeds through the two experimental layers used in this thesis. In YAFS, the workload traverses the simulated fog graph and interacts with placement, routing, and service-processing logic. In Mininet, the same logical demand is injected into an emulated topology with explicit link, compute, and queue constraints. At regular decision intervals, the runtime collects system observations describing workload intensity, node utilisation, queue behaviour, and recent mobility patterns. For the proposed method, these observations are combined with short-horizon predictive signals before control decisions are produced.

Based on the current state, the controller issues provisioning actions that adjust node-level service capacity, such as worker limits or equivalent processing allocations at selected nodes. The resulting changes in latency, queue occupancy, throughput, and request completion provide the feedback used to assess each policy. This creates a closed loop between the trace-driven workload generator, the fog execution environment, and the adaptive controller.

After each experiment, a statistics engine processes the logs to compute latency, queue pressure, throughput, drop rate, delivery ratio, SLA violations, and recovery behaviour. Visualisation scripts then generate the figures used throughout the thesis from this unified evaluation pipeline.

## 3.7 Fog Topology Construction

The fog topology is constructed as the spatial substrate onto which the preprocessed Beijing mobility events are mapped. Rather than representing an abstract graph detached from the workload, the topology is designed to align with the central Beijing study region introduced earlier so that changes in taxi density translate directly into geographically meaningful demand patterns.

The study area is discretised into a tiled fog-computing fabric in which each tile corresponds to a fog node responsible for serving requests generated within its vicinity. This tiled representation preserves locality between mobility events and service demand, allowing hotspot formation and spatial demand migration to be expressed naturally at the node level.

To isolate the impact of control decisions from fixed infrastructure asymmetry, all fog nodes are initialised with homogeneous baseline processing and communication resources. Connectivity checks and repair operations are then applied to ensure that the resulting fog graph is fully reachable and that routing paths exist between all relevant endpoints during execution.

This topology therefore serves as a stable and reproducible substrate for all compared controllers. Because the same tiled mapping and base infrastructure are reused across experiments, observed differences in latency, congestion, throughput, and recovery can be attributed to control behaviour rather than to changes in the underlying fog fabric.

## 3.8 Fog Topology Generation and Connectivity

The fog topology used in this study is constructed using the TiledTopology model provided by the YAFS framework. The objective is to create a geographically coherent fog infrastructure that aligns with the urban region represented in the Beijing T-Drive dataset. Accordingly, the topology covers the same spatial window as the mobility traces—39.85°–40.05° N and 116.25°–116.55° E corresponding to central Beijing. This alignment ensures that each taxi position recorded in the dataset can be mapped consistently to a fog tile, preserving the spatial relationship between mobility patterns and computational demand.

The region is discretised into a uniform grid defined by a configurable size parameter S. In all experiments, S = 25 (configured via topology.size in config.ini), producing a 25 × 25 tiled lattice. Although TiledTopology supports a multilevel hierarchy, this work uses the leaf tiles (finest-resolution tiles) as the operational fog nodes. Each leaf tile is represented as a node located at the geometric centre of its tile, and node identifiers are generated systematically (e.g., n3lt2ln5) to encode hierarchy and tile indices, enabling direct traceability between the geographic grid and the topology graph.Connectivity is initially defined using the adjacency and parent–child relationships generated by TiledTopology. To ensure the network is fully connected and suitable for routing, a connectivity-repair procedure is applied when needed:

any isolated node is linked to its nearest neighbour based on tile-grid Euclidean distance. Each edge is annotated with uniform link attributes, including bandwidth (BW = default_BW) and packet-loss probability (PR = 0.1), so that routing and transport behaviour are evaluated under consistent link assumptions.

Consistent with the modelling constraints of this thesis, the topology is initialised as a homogeneous fog infrastructure. All fog nodes are tagged as fog nodes and assigned the same baseline compute rate (instructions per tick, IPT) and baseline communication capacity. No per-node heterogeneity in CPU, memory, storage, or bandwidth is introduced at topology-construction time; instead, any resource differentiation observed during experiments arises from runtime provisioning decisions made by the evaluated controllers.

Once constructed, the fog graph is loaded into YAFS and reused across all runs to ensure experimental consistency. In the Mininet–YAFS testbed, this logical graph is instantiated as an emulated network with corresponding endpoints for each fog node. Before each experiment, basic structural properties such as node and edge counts and overall connectivity are logged to verify reproducibility and to ensure that the underlying infrastructure remains unchanged across evaluations. This stable, tiled, and geographically aligned topology forms the substrate on which mobility-driven workloads are mapped and on which resource-management policies are assessed.

## 3.9 Application and Messaging Flow

The application and messaging model defines how mobility events from the Beijing T-Drive trace are translated into computational workloads within the fog environment. This mapping ensures that each mobility event corresponds to concrete data-plane activity and that the resulting compute and network pressure emerges directly from real spatio-temporal movement patterns.The central application, FogProcessingApp, represents the workflow of an intelligent transportation service. Each taxi is modelled as a TaxiGateway entity that emits multiple message types reflecting common vehicle-side interactions. Lightweight messages such as TAXI_GPS and TAXI_TRAFFIC represent localisation and congestion reporting, while heavier transactional messages—TAXI_PICKUP, TAXI_ROUTE, and TAXI_PAYMENT—represent dispatch requests, route planning, and payment confirmation. These messages are

processed by fog-side service modules including FogProcessor, RouteProcessor, and PaymentHandler, which execute computation and generate downstream outputs (PROCESSED, ROUTE_RESULT, and PAYMENT_ACK) before delivery to DataSink. Each message class is assigned instruction and byte budgets so that the platform captures realistic compute consumption and bandwidth utilisation throughout the workflow.

During experiment initialisation, a dynamic population manager binds the trace-derived taxi trajectories to this application model. Each timestamped GPS record triggers the creation, update, or migration of a TaxiGateway at the fog node corresponding to the taxi's current coordinates in the tiled topology. In this manner, real taxi mobility drives message generation and injects authentic TAXI_* events into the fog fabric. Because the same event stream is used for workload generation and for runtime observation, the platform preserves tight temporal alignment between mobility dynamics, request arrivals, and system measurements.

At runtime, messages traverse the fog network to reach available service instances and are processed through the defined module chain. Processing and communication delays accumulate as requests pass through compute queues, consume worker capacity, and traverse network links. This produces latency, queue pressure, throughput, and (when stressed) request drops that reflect the joint effect of mobility-driven demand and fog-side capacity constraints. The role of the evaluated controllers is to adjust service capacity over time (e.g., processing allocations and worker limits at selected nodes) to maintain service quality under this mobility-driven workload; the internal control mechanisms used to produce these decisions are described in Chapter 4.

## 3.10 Mininet Testbed Configuration

### 3.10.1 Network Topology and Link Emulation

The experimental evaluation is conducted using a Mininet-based emulation testbed, which enables realistic modeling of network behavior while maintaining full experimental control. The network topology is defined as a graph in the configuration file (config.ini), specifying the number of nodes, default link bandwidth, and propagation delay. Each graph node is

instantiated as a Mininet switch–host pair, creating a one-to-one mapping between logical fog nodes and emulated compute endpoints.

Network links are created using TCLink, allowing bandwidth limitations and propagation delays to be enforced at the Linux traffic control layer. This ensures that congestion, queueing, and transmission delays emerge naturally from the emulation rather than being artificially injected by the simulator. Open vSwitch is used in bridge mode (ovsbridge), and all switches operate in standalone mode, eliminating any dependency on an external SDN controller and ensuring that routing behavior remains fixed across experiments.

### 3.10.2 Fog Node Compute and Queue Model

Each Mininet host runs a fog-processing service that emulates compute and buffering constraints at the node level. The service is parameterized by four key attributes: the number of worker threads (parallelism), the queue size (buffer capacity), the per-request processing time (in milliseconds), and the metrics reporting interval. Together, these parameters define the effective compute capacity of each fog node and allow the system to be stressed under different bottleneck regimes.

By adjusting the number of workers and processing time, CPU-bound scenarios can be emulated, while queue-bound behavior is induced by restricting buffer sizes under high arrival pressure. This design enables controlled and repeatable stress conditions, ensuring that differences in performance are attributable to the control policy rather than uncontrolled system behavior.

### 3.10.3 Mobility-Driven Workload Generation

Workload generation is driven by a real-world mobility trace (T-Drive), providing spatially and temporally correlated demand patterns. GPS points from the trace are filtered and mapped onto the network topology using a tiled spatial projection, after which each mobility event generates a timestamped request assigned to the nearest fog node. This process produces a realistic, non-stationary workload that reflects urban mobility dynamics.

The intensity and temporal characteristics of the workload are controlled through trace sampling parameters (e.g., maximum number of taxis and sampling frequency) and time-scaling factors such as time compression and playback speed. Additionally, hotspot behavior can be

induced using a top-K gateway mechanism, which concentrates traffic on a subset of nodes to emulate flash-crowd and hotspot-shift conditions. Requests are injected in per-node gateway mode, ensuring that each fog node receives traffic proportional to its mapped local mobility density.

### 3.10.4 Control Loop and Policy Execution

A centralized controller operates in discrete time steps, periodically collecting metrics from each fog node, including latency, queue pressure, utilization, arrival rates, and drop counts. Based on these observations, the controller applies resource management actions such as scaling worker threads, adjusting CPU allocation, or modifying queue limits.

The testbed supports multiple control policies executed under identical conditions, including static allocation, threshold-based heuristics, population-based optimizers (GA and PSO), baseline reinforcement learning, and Soft Actor–Critic (SAC) variants with safety guards and prediction modules. Because all policies are evaluated using the same topology, workload, and random seeds, observed performance differences can be attributed solely to policy behavior.

### 3.10.5 Runtime Logging and Metric Extraction

Each experimental run generates a dedicated logging directory that records both system state and control decisions. Key artifacts include a per-step controller log containing metrics and actions, a sink-level summary of delivered and dropped requests, and a record of the exact mapped workload events used during the run. These logs are post-processed to compute end-to-end performance metrics such as mean and tail latency, queue pressure, drop rate, delivery ratio, SLA violation rate, and recovery time following overload events.

### 3.10.6 Rationale for Using Mininet

The use of Mininet provides a crucial balance between realism and controllability. By enforcing real bandwidth limits, queueing behavior, and propagation delays at the network layer, Mininet captures packet-level congestion and delay dynamics that are absent in purely abstract simulators. As a result, latency inflation, queue buildup, and packet drops arise organically from network and compute constraints. This ensures that observed improvements in

performance reflect genuine control advantages rather than artifacts of simplified timing models, strengthening the validity of the experimental conclusions.

# CHAPTER 4

# PREDICTIVE RESOURCE CONTROLE FRAMEWORK

## 4.1 Overview and Design Rationale

Accurate short-horizon demand forecasting is a prerequisite for proactive resource provisioning in mobility-aware fog computing systems. In urban environments, node-level workload evolves through two coupled mechanisms. First, spatial mobility redistributes demand across neighboring fog regions as taxis move through the city. Second, temporal burstiness increases local workload intensity because of flash crowds, queue buildup, congestion feedback, or dropped requests. Controllers that rely only on reactive measurements, or only on immediate mobility observations, often detect overload too late, after queues and latency have already deteriorated. This delayed response may lead to queue growth, tail-latency inflation, and service-level agreement (SLA) violations. Conversely, controllers driven only by local demand measurements lack spatial foresight and therefore cannot anticipate where the next workload hotspot will emerge.

To address this limitation, this chapter introduces FusionPredictor, a unified node-level forecasting model that integrates a mobility-aware component and a load-aware component within a single prediction pipeline. The mobility-aware component captures short-term spatial migration patterns from node-to-node taxi transitions and neighborhood context, while the load-aware component estimates local workload intensity from offered load and queue state. These two internal components are combined through an adaptive fusion stage that reallocates trust between spatial and local-load evidence according to the instantaneous congestion and load conditions of each node.

A central design requirement is that the predictor should reflect the true offered workload, rather than only the traffic that was successfully admitted for service. Under congestion, admitted traffic alone can significantly underestimate actual system pressure because a portion of the workload may be dropped before processing. For this reason, FusionPredictor uses offered load, defined as the sum of accepted and dropped request rates, as one of its primary input signals. In addition, the final forecasts are stabilized through conservative lower bounds and upper caps so that the predictor remains robust under abrupt workload shifts. The resulting model provides computationally lightweight, short-horizon node-level arrival forecasts that can be embedded directly in the fog control loop for proactive provisioning and bottleneck avoidance.

## 4.2 Input Signals and Observed State

At each control step $t$, FusionPredictor receives a set of node-level measurements that jointly describe local occupancy, mobility evolution, and system stress. For node $i$, these measurements are the current taxi count $n_{i,t}$, the short-term taxi inflow and outflow rates $\lambda_{i,t}^{\text{in}}$and $\lambda_{i,t}^{\text{out}}$, the offered-load rate $o_{i,t}$, the normalized queue pressure $q_{i,t}$, the incoming and outgoing neighbor-flow features $f_{i,t}^{\text{in}}$and $f_{i,t}^{\text{out}}$, and the mean neighboring taxi occupancy $\bar{n}_{\mathcal{N}(i),t}$, where $\mathcal{N}(i)$denotes the set of neighboring nodes of node $i$. The observed node state is therefore written as

$$z_{i,t}^{\text{obs}} = [n_{i,t}, \lambda_{i,t}^{\text{in}}, \lambda_{i,t}^{\text{out}}, o_{i,t}, q_{i,t}, f_{i,t}^{\text{in}}, f_{i,t}^{\text{out}}, \bar{n}_{\mathcal{N}(i),t}]^{\top} \tag{4.1}$$

The neighboring occupancy term is defined as

$$\bar{n}_{\mathcal{N}(i),t} = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} n_{j,t}. \tag{4.2}$$

The offered-load signal is defined as

$$o_{i,t} = r_{i,t}^{\mathrm{acc}} + r_{i,t}^{\mathrm{drop}}, \tag{4.3}$$

where $r_{i,t}^{\mathrm{acc}}$is the accepted request rate and $r_{i,t}^{\mathrm{drop}}$is the dropped request rate at node $i$during control step $t$. This definition is important because admitted traffic alone may underestimate actual system pressure precisely when overload becomes severe. By explicitly including dropped requests, the predictor preserves visibility of latent demand that the node was asked to serve but could not process.

For the mobility-aware component, count-like features are log-compressed to reduce the effect of heavy-tailed occupancy values. The transformed predictor input is therefore

$$x_{i,t}^{\mathrm{flow}} = [\log(1 + n_{i,t}), \lambda_{i,t}^{\mathrm{in}}, \lambda_{i,t}^{\mathrm{out}}, o_{i,t}, q_{i,t}, f_{i,t}^{\mathrm{in}}, f_{i,t}^{\mathrm{out}}, \log(1 + \bar{n}_{\mathcal{N}(i),t})]^{\top}. \tag{4.4}$$

This representation combines local occupancy, short-term mobility, current offered demand, congestion state, directional neighborhood flow, and surrounding spatial activity in a compact node-level predictor input.

## 4.3 FusionPredictor: Mobility-Aware Component

The first internal component of FusionPredictor is a mobility-aware recurrent component. Its purpose is to estimate short-horizon workload migration by learning spatial and temporal patterns in taxi movement across neighboring fog regions. This component is mobility-aware rather than mobility-only, because its recurrent input also includes offered-load and queue-

pressure terms to preserve robustness under bursty and congested operating conditions. Its output is an intermediate bounded forecast of expected arrivals at node $i$over horizon $\tau$, denoted by $\hat{A}_{i,t+\tau}^{\mathrm{mob}}$.

### 4.3.1 Mobility Representation and Recurrent Update

To encode spatial mobility, the model maintains a directed transition graph over fog nodes. Let $\Delta_{u,v}^{(t)}$denote the number of newly observed taxi handovers from node $u$ to node $v$ during control step $t$. The decayed flow strength on edge $(u, v)$is updated as

$$E_{u,v}^{(t)} = \rho E_{u,v}^{(t-1)} + \Delta_{u,v}^{(t)}, \tag{4.5}$$

where $\rho \in (0,1)$is a decay factor that assigns greater weight to recent movements while retaining short-term mobility memory.

Using these edge weights, the incoming and outgoing flow features of node $i$are defined as

$$f_{i,t}^{\mathrm{in}} = \min\left(2, \frac{1}{\phi}\sum_{j\in\mathcal{N}(i)} E_{j,i}^{(t)}\right), \tag{4.6}$$

$$f_{i,t}^{\mathrm{out}} = \min\left(2, \frac{1}{\phi}\sum_{j\in\mathcal{N}(i)} E_{i,j}^{(t)}\right), \tag{4.7}$$

where $\phi > 0$ is normalization constant. The clipping prevents unusually large transition bursts from destabilizing the recurrent update.

Using the transformed input vector $x_{i,t}^{\mathrm{flow}}$, each node maintains a hidden state $h_{i,t}$that summarizes recent mobility evolution. The hidden state is updated using a standard gated recurrent update:

$$z_{i,t} = \sigma\,,\left(W_z x_{i,t}^{\text{flow}} + U_z h_{i,t-1} + b_z\right), \tag{4.8}$$

$$r_{i,t} = \sigma\,,\left(W_r x_{i,t}^{\text{flow}} + U_r h_{i,t-1} + b_r\right), \tag{4.9}$$

$$\tilde{h}_{i,t} = \tanh\ \,,\left(W_h x_{i,t}^{\text{flow}} + U_h(r_{i,t} \odot h_{i,t-1}) + b_h\right), \tag{4.10}$$

$$h_{i,t} = (1 - z_{i,t}) \odot h_{i,t-1} + z_{i,t} \odot \tilde{h}_{i,t}, \tag{4.11}$$

where $z_{i,t}$and $r_{i,t}$are the update and reset gates, $\sigma(\cdot)$is the sigmoid function, and $\odot$denotes element-wise multiplication.

The hidden state is then projected to a nonnegative latent mobility score,

$$g_{i,t} = \max\ \,,\left(0,\, w_o^{\top} h_{i,t} + b_o\right). \tag{4.12}$$

### 4.3.2 Forecast Shaping and Online Calibration

The latent recurrent output is transformed into a preliminary forecast of arrivals over the prediction horizon $\tau$:

$$\bar{A}_{i,t+\tau}^{\text{mob}} = \beta n_{i,t} + s_i g_{i,t}, \tag{4.13}$$

where $n_{i,t}$is the current taxi count at node $i$, $\beta$is a baseline turnover coefficient, and $s_i$is a node-specific calibration factor.

To improve robustness under changing demand conditions, the model also constructs an anchor forecast using the measured arrival rate and offered load:

$$A_{i,t+\tau}^{\text{anchor}} = \tau(\lambda_{i,t}^{\text{in}} + \omega_a o_{i,t}), \tag{4.14}$$

where $\omega_a > 0$ controls how strongly offered load influences the anchor.

A queue-aware blending factor is then introduced to adaptively mix the preliminary forecast with the anchor forecast:

$$\alpha_{i,t} = \text{clip}\ \ , \left(\alpha_0 + \alpha_q q_{i,t}, 0, \alpha_{\max}\right), \tag{4.15}$$

with $\text{clip}\ (x, a, b) = \min\{b, \max\ (x, a)\}$. The mixed forecast is therefore

$$A_{i,t+\tau}^{\text{mix}} = (1 - \alpha_{i,t})\bar{A}_{i,t+\tau}^{\text{mob}} + \alpha_{i,t} A_{i,t+\tau}^{\text{anchor}}. \tag{4.16}$$

To avoid unrealistic underestimation during bursty demand periods, a queue-aware lower bound is imposed:

$$L_{i,t} = \tau o_{i,t}(\eta_0 + \eta_q q_{i,t}), \tag{4.17}$$

where $\eta_0$is a baseline floor coefficient and $\eta_q$controls how strongly congestion increases the floor.

To prevent unstable overprediction, an upper bound is defined as

$$C_{i,t} = \min\ \ , \left(C_{\max}, \max, \left(5,\ 1.5\tau\max\ (\lambda_{i,t}^{\text{in}}, \xi o_{i,t}),\ 0.6n_{i,t} + 15\right)\right), \tag{4.18}$$

where $C_{\max}$is a global cap and $\xi > 0$ controls how strongly offered load contributes inside the flow-based cap.

The final mobility-aware intermediate forecast is therefore

$$\hat{A}^{\text{mob}}_{i,t+\tau} = \min\ ,\ \left(C_{i,t}, \max\ ,\ \left(A^{\text{mix}}_{i,t+\tau}, L_{i,t}\right)\right). \tag{4.19}$$

Because mobility behavior varies across regions and may drift over time, this component includes a lightweight online calibration step. After the forecast horizon has elapsed, the realized arrivals $A^{\text{obs}}_{i,t+\tau}$are compared with the prediction, and the node-specific scale is updated by exponential smoothing:

$$r^{\text{cal}}_{i,t+\tau} = \frac{A^{\text{obs}}_{i,t+\tau}}{\max\left(\hat{A}^{\text{mob}}_{i,t+\tau}, 1\right)}, \tag{4.20}$$

$$s_i \leftarrow (1-\omega_s)s_i + \omega_s \text{ clip }\ ,\ \left(r^{\text{cal}}_{i,t+\tau}, s_{\min}, s_{\max}\right), \tag{4.21}$$

where $\omega_s \in (0,1)$is the calibration step size, and $s_{\min}$and $s_{\max}$bound the feasible correction range.

### 4.3.3 Modeling Properties

The mobility-aware component is spatially aware because directional transition features and mean neighbouring taxi occupancy are embedded directly in the input representation, allowing the model to capture local mobility diffusion and short-range hotspot migration across adjacent fog regions. It is temporally aware because the recurrent hidden state preserves recent mobility history across successive control steps, thereby retaining short-horizon momentum that cannot be inferred from a single observation. It is also congestion-aware, since offered load and queue pressure influence forecast shaping under stressed conditions. The recurrent gate equations themselves are standard; the contribution of this component lies in the flow-aware input design,

the conversion of recurrent features into horizon-level arrivals, and the stability-oriented shaping and calibration steps used to support proactive fog control. Consequently, the component captures both local hotspot migration and short-horizon mobility momentum while remaining robust under bursty and congested operating conditions.

## 4.4 FusionPredictor: Load-Aware Component

The second internal component of FusionPredictor is a load-aware component that explicitly models offered load and burst behavior. Its purpose is to provide a lightweight estimate of short-horizon workload intensity at each fog node and to convert that estimate into a bounded forecast of expected arrivals over the prediction horizon.

In this subsection, $o_{i,t}$denotes the offered-load rate at node $i$and control step $t$, $\ell_{i,t}$denotes the smoothed demand level, $b_{i,t}$denotes the short-term demand trend, $q_{i,t} \in [0,1]$denotes normalized queue pressure, $W_{\text{burst}}$denotes the burst-history window length, $Q_{0.9}(\cdot)$denotes the 90th-percentile operator, $\tau$denotes the forecast horizon, $\Delta$denotes the control-step duration, $H = \tau/\Delta$denotes the number of forecast steps, and $C_{\max}$denotes the maximum admissible output of this component.

### 4.4.1 Online Level–Trend Update

The offered-load rate is defined as

$$o_{i,t} = r_{i,t}^{\text{acc}} + r_{i,t}^{\text{drop}}, \tag{4.22}$$

where $r_{i,t}^{\text{acc}}$is the accepted request rate and $r_{i,t}^{\text{drop}}$is the dropped request rate. This definition is essential because admitted traffic alone may underestimate true system pressure precisely when congestion is severe.

For each node $i$, the load-aware component maintains a smoothed level term $\ell_{i,t}$and a trend term $b_{i,t}$. These quantities are updated using standard level trend smoothing equations:

$$\ell_{i,t} = \alpha_d o_{i,t} + (1 - \alpha_d)(\ell_{i,t-1} + b_{i,t-1}), \tag{4.23}$$

$$b_{i,t} = \beta_d(\ell_{i,t} - \ell_{i,t-1}) + (1 - \beta_d)b_{i,t-1}, \tag{4.24}$$

where $\alpha_d \in (0,1)$controls how quickly the level estimate reacts to new observations, and $\beta_d \in (0,1)$controls the responsiveness of the trend estimate.

### 4.4.2 Burst-Aware Load Forecast

To preserve sensitivity to abrupt load surges, the component augments the level trend estimate with a burst envelope computed from recent offered-load observations:

$$\text{burst}_{i,t} = Q_{0.9}\,, \left(o_{i,t-W_{\text{burst}}+1}, \ldots, o_{i,t}\right). \tag{4.25}$$

The number of forecast steps is defined as

$$H = \frac{\tau}{\Delta}. \tag{4.26}$$

The base load forecast is then

$$\hat{o}_{i,t+\tau}^{\text{base}} = \max\ \,, \left(\ell_{i,t} + Hb_{i,t}, \text{burst}_{i,t}, o_{i,t}\right). \tag{4.27}$$

To incorporate congestion feedback, the base forecast is amplified using the normalized queue pressure:

$$\hat{o}_{i,t+\tau}^{\text{load}} = \hat{o}_{i,t+\tau}^{\text{base}}(1 + \eta_q^{\text{load}} q_{i,t}), \tag{4.28}$$

where $\eta_q^{\text{load}} > 0$ is the queue-pressure amplification coefficient.

The demand-rate forecast is then converted into a horizon-arrival forecast as

$$\tilde{A}_{i,t+\tau}^{\text{load}} = \tau \hat{o}_{i,t+\tau}^{\text{load}}, \tag{4.29}$$

and bounded by

$$\hat{A}_{i,t+\tau}^{\text{load}} = \min\ , \left(C_{\max}, \tilde{A}_{i,t+\tau}^{\text{load}}\right). \tag{4.30}$$

In addition to the arrival forecast, the component computes a lightweight confidence indicator based on the stability of recent offered-load observations:

$$c_{i,t}^{\text{load}} = c_{\min} + 0.6\max\ , \left(0,\ 1 - \frac{\sigma_i(t)}{\mu_i(t) + \varepsilon}\right), \tag{4.31}$$

where $\mu_i(t)$and $\sigma_i(t)$are the mean and standard deviation of recent offered-load observations, $c_{\min}$is a minimum confidence floor, and $\varepsilon > 0$ avoids division by zero.

### 4.4.3 Modeling Properties

The load-aware component is burst-aware because it explicitly preserves recent high-load behaviour through the quantile-based burst envelope. It is congestion-aware because queue pressure directly amplifies the forecast under overload. Unlike the mobility-aware component, it does not model spatial migration; instead, it provides a low-overhead estimate of local offered-load intensity. The two internal components are therefore complementary: the mobility-aware component captures short-horizon spatial migration, whereas the load-aware component preserves local overload sensitivity. This component is intentionally classical rather than fully learned. Its purpose is not to replace the mobility-aware component, but to provide a stable, interpretable, and computationally lightweight overload-sensitive complement suitable for online deployment in the fog control loop.

## 4.5 Adaptive Fusion Layer and Safety Bounds

The adaptive fusion layer combines the mobility-head forecast and the demand-head forecast to obtain a single bounded short-horizon prediction suitable for proactive resource control. All

forecast quantities in this subsection are expressed in the same unit, namely the predicted number of arrivals at node $i$over horizon $\tau$. Let $\hat{A}^{\mathrm{mob}}_{i,t+\tau}$denote the mobility-head forecast, $\hat{A}^{\mathrm{dem}}_{i,t+\tau}$denote the demand-head forecast, $o_{i,t}$denote the offered-load rate, and $q_{i,t} \in [0,1]$denote the normalized queue pressure. Define

$$\mathrm{clip}\,(x, a, b) = \min\{b, \max(x, a)\}. \tag{4.32}$$

## 4.6 Demand-strength signal

To quantify current workload intensity, a normalized demand-strength signal is constructed from the offered-load rate. Let $o_{\mathrm{ref}} > 0$ denote a reference offered-load level and define

$$\kappa_o = \log(1 + o_{\mathrm{ref}}). \tag{4.33}$$

The demand-strength signal is then

$$s_{i,t} = \mathrm{clip}\ \left(\frac{\log(1 + o_{i,t})}{\kappa_o}, 0, 1\right). \tag{4.34}$$

This signal increases smoothly with offered load while remaining bounded in $[0, 1]$. When the offered load is low, $s_{i,t}$remains small and the fusion layer relies more heavily on the mobility branch. As the offered load rises, $s_{i,t}$increases and progressively shifts more authority toward the demand-aware branch.

## 4.7 Adaptive Fusion Weights

The contribution of the demand branch is controlled by an adaptive weight that depends on both queue pressure and demand strength:

Here, the baseline term sets the default contribution of the demand-aware branch, the queue-related term increases that contribution under congestion, and the demand-strength term increases it when current workload intensity becomes high. The lower and upper bounds ensure that neither internal component is completely ignored.

$$w_{i,t}^{\mathrm{dem}} = \mathrm{clip}\ \ , \left(\omega_0 + \omega_q q_{i,t} + \omega_s s_{i,t}, w_{\min}, w_{\max}\right), \tag{4.35}$$

$$w_{i,t}^{\mathrm{mob}} = 1 - w_{i,t}^{\mathrm{dem}}. \tag{4.36}$$

Here, $\omega_0$is the base demand weight, $\omega_q$controls sensitivity to queue pressure, and $\omega_s$controls sensitivity to workload intensity through $s_{i,t}$. The bounds $w_{\min}$and $w_{\max}$ensure that neither branch is completely ignored.

where $\beta_{\min} > 0$ is the baseline mobility-scaling factor and $\beta_s$controls the strength of demand-driven amplification.

To prevent unrealistic mobility-dominant over-prediction, the scaled mobility forecast is clipped relative to the demand-head forecast:

$$\tilde{\mathrm{A}}_{i,t+\tau}^{mob} = \min\left(A_{i,t+\tau}^{mob,scale}, \chi\ \hat{\mathrm{A}}_{i,t+\tau}^{dem}\right) \tag{4.37}$$

where $\chi > 0$ is a mobility-clipping multiplier. The use of $\chi$is preferable here because $\rho$has already been used earlier in the chapter as the decay factor of the mobility transition graph.

This clipping step ensures that the mobility branch cannot dominate the final prediction when transient flow patterns become exaggerated or poorly aligned with the currently observed offered load.

## 4.8 Component Scaling and Clipping

Before fusion, the mobility-aware intermediate forecast is adjusted according to the current demand-strength signal: The baseline scaling term preserves a minimum mobility contribution,

while the demand-strength term allows moderate amplification when current load is already non-negligible.

To prevent unrealistic overprediction, the scaled mobility-aware forecast is clipped relative to the load-aware forecast: This step ensures that the mobility-aware component cannot dominate the fused prediction when transient flow patterns become exaggerated or poorly aligned with the currently observed offered load.

$$A_{i,t+\tau}^{\mathrm{mob,sc}} = \hat{A}_{i,t+\tau}^{\mathrm{mob}}(\beta_{\min} + \beta_s s_{i,t}), \tag{4.38}$$

where $\beta_{\min} > 0$is the baseline scaling factor and $\beta_s$controls the strength of demand-driven amplification.

To prevent unrealistic overprediction, the scaled forecast is clipped relative to the load-aware forecast:

$$\tilde{A}_{i,t+\tau}^{\mathrm{mob}} = \min\ , \left(A_{i,t+\tau}^{\mathrm{mob,sc}}, \chi \hat{A}_{i,t+\tau}^{\mathrm{load}}\right), \tag{4.39}$$

where $\chi > 0$ is a clipping multiplier.

## 4.9 Fused Forecast

The raw fused forecast is obtained by combining the clipped mobility prediction with the demand-head prediction using the adaptive weights:

$$\hat{\mathrm{A}}_{i,t+\tau}^{raw} = w_{i,t}^{mob} \tilde{\mathrm{A}}_{i,t+\tau}^{mob} + w_{i,t}^{dem} \hat{\mathrm{A}}_{i,t+\tau}^{dem} \tag{4.40}$$

Because both branches are expressed in the same unit, namely arrivals over horizon $\tau$, the fusion is dimensionally consistent and directly interpretable by the controller. This equation performs the actual adaptive blending between spatial foresight and congestion-aware local demand estimation.

## 4.10 Safety Bounds: Demand Floor and Cap

To avoid harmful underprediction during overload, a demand-anchored lower bound is enforced:

$$L_{i,t}^{\text{fusion}} = \hat{A}_{i,t+\tau}^{\text{dem}}(\theta_0 + \theta_q q_{i,t}), \tag{4.41}$$

where $\theta_0$is a baseline floor coefficient and $\theta_q$increases the floor under congestion.

To prevent unstable overreaction, an upper bound is also enforced:

$$U_{i,t}^{\text{fusion}} = \max\ , \left(\hat{A}_{i,t+\tau}^{\text{dem}}(\kappa_0 + \kappa_q q_{i,t}), \kappa_f \tilde{A}_{i,t+\tau}^{\text{mob}}\right), \tag{4.42}$$

where $\kappa_0$and $\kappa_q$define a demand-anchored upper reference and $\kappa_f$limits the final prediction relative to the clipped mobility signal.

The final fused forecast is therefore

$$\hat{A}_{i,t+\tau}^{\text{fusion}} = \min\ , \left(\max\ , \left(\hat{A}_{i,t+\tau}^{\text{raw}}, L_{i,t}^{\text{fusion}}\right), U_{i,t}^{\text{fusion}}\right). \tag{4.43}$$

## 4.11 Algorithmic Description

At each control step $t$, FusionPredictor first updates its node-level telemetry and recent mobility-flow statistics. The mobility-aware component then produces an intermediate arrival forecast $\hat{A}_{i,t+\tau}^{\text{mob}}$, while the load-aware component updates its level, trend, and burst statistics to generate $\hat{A}_{i,t+\tau}^{\text{load}}$. Next, the fusion stage computes the demand-strength signal $s_{i,t}$, derives the

adaptive fusion weights, scales and clips the mobility-aware forecast, and forms the raw fused forecast. Finally, the floor and cap bounds are applied to obtain the bounded output $\hat{A}_{i,t+\tau}^{\text{fusion}}$.

To support online deployment, runtime is explicitly bounded by restricting the number of nodes processed per update and by enforcing a maximum per-step update duration. The resulting bounded forecast is then passed to the control framework for proactive decision making.

## 4.12 Architecture Overview

The proposed framework operates as a closed loop between forecasting, control, and runtime enforcement. At each control step, telemetry describing taxi mobility, offered load, queue pressure, and resource utilization is collected from the fog environment. FusionPredictor then produces a bounded short-horizon workload forecast for each node. These forecasts, together with current system measurements, are used to construct the aggregated CH-SAC state and to identify the most critical focus nodes. The actor network generates continuous multi-resource actions for the selected focus nodes, after which a guard layer enforces activity gating, forecast-aware scaling, queue emergency overrides, SLA guardrails, and node-specific feasibility bounds. The resulting safe actions are applied to the fog platform, and the environment evolves under the updated configuration. In this way, the framework couples predictive anticipation with constrained reinforcement-learning control in a unified online provisioning loop.

## 4.13 Outputs of the Fusion Predictor

FusionPredictor is the only predictive module used in the framework. Its formal output is the bounded short-horizon arrival forecast

$\hat{A}_{i,t+\tau}^{\text{fusion}}$,

which represents the expected number of arrivals at node $i$over the next control horizon $\tau$.

Any additional indicators used by the controller are derived diagnostics obtained from the same fused prediction pipeline rather than from any separate predictive model. Consequently, the

framework contains one consistent predictor, namely FusionPredictor, whose fused short-horizon forecast is the predictor signal used for proactive hotspot prioritization and predictor-aware control.

## 4.14 Dynamic Behavior and Robustness

FusionPredictor is designed to remain stable under abrupt workload changes, transient bursts, and congestion escalation. The mobility-aware component captures short-term spatial migration, while the load-aware component preserves visibility of latent local pressure through offered load and burst sensitivity. The fusion stage then reallocates trust between these two internal components according to current workload intensity and queue pressure. This means that the predictor remains spatially anticipatory under normal operating conditions but becomes more conservative and load-sensitive when congestion increases.

Robustness is further reinforced by explicit safety bounds. The lower bound prevents harmful underprediction when demand is rising under stress, whereas the upper bound prevents unstable overreaction when transient signals become exaggerated. The online calibration step compensates for persistent node-level bias without requiring expensive retraining. Together, these mechanisms ensure that FusionPredictor behaves as a bounded, control-oriented forecasting module suitable for real-time fog-resource management.

## 4.15 Complexity and Runtime Safety

Let $N'$ denote the number of nodes processed at a given update after runtime bounding, let $M$ denote the predictor input dimension, and let $d_h$ denote the GRU hidden-state dimension. For a single-layer GRU, the per-update computational cost is

$$T_{update} = O\left(N'(d_h M + d_h^2)\right) \tag{4.44}$$

while the memory required for node hidden states is

$$S_{state} = O(N' d_h) \tag{4.45}$$

In the present design, $M$and $d_h$are small, fixed constants, so the practical runtime grows approximately linearly with the number of processed nodes $N'$. Execution time is further bound by limiting node processing and by enforcing a maximum per-step update time. These safeguards ensure that the predictor remains suitable for real-time use even under heavy system load.

## 4.16 Innovation Summary

FusionPredictor contributes a single bounded online forecasting pipeline that integrates mobility-sensitive recurrent state evolution, offered-load and burst-aware local demand estimation, adaptive congestion-aware fusion, and explicit safety bounds. Its novelty lies in this operational integration for proactive fog control, rather than in proposing a new recurrent cell or a new classical smoothing rule.

CH-SAC does not replace the theoretical core of Soft Actor–Critic. Its contribution lies in the predictor-guided, constraint-aware control stack built around SAC: compact hotspot-focused state construction, continuous multi-resource actions, runtime-safe action mapping, and SLA-aware execution under dynamic workload conditions.

## 4.17 Integration of FusionPredictor with CH-SAC Control

FusionPredictor and CH-SAC are integrated at every control step. First, FusionPredictor converts mobility and system-stress telemetry into bounded short-horizon forecasts of expected arrivals. These forecasts are then used in three ways within the controller. First, they contribute to focus-node prioritization so that control effort is concentrated on the nodes most likely to experience near-future pressure. Second, they enter the predictor-related component of the aggregated state observed by the actor and critic. Third, they support forecast-aware action shaping within the runtime guard, allowing scale-up decisions to become more proactive when the predicted workload is rising rapidly. Consequently, prediction influences CH-SAC not through a separate side mechanism, but directly through hotspot selection, state construction, and safe multi-resource action execution.

## 4.18 CH-SAC for Fog Resource Provisioning

CH-SAC is the control component of the proposed framework. It combines a standard Soft Actor–Critic learning core with predictor-guided state design, hotspot-focused resource control, multi-resource continuous actions, and a runtime safety layer. The term "hybrid" refers to this integration of learning, forecasting, and rule-based safety enforcement. The term "constraint-aware" refers to the fact that actor outputs are not applied directly, but are filtered through queue-, SLA-, and feasibility-aware runtime guard logic before deployment.

### 4.18.1 Motivation and Design Objectives

The controller is designed to satisfy five requirements. First, it must react proactively to short-horizon workload changes by exploiting FusionPredictor outputs rather than relying only on delayed reactive measurements. Second, it must remain computationally tractable by acting on a compact set of focus nodes instead of the full node set at every step. Third, it must support simultaneous adaptation of CPU, memory, storage, and bandwidth rather than a single abstract capacity variable. Fourth, it must remain safe under overload, queue escalation, and SLA deterioration through bounded runtime action correction. Fifth, it must remain suitable for online deployment in dynamic fog environments where workload hotspots move over time.

### 4.18.2 Base Soft Actor–Critic Objective

The learning core of CH-SAC remains standard Soft Actor–Critic. Let $\pi_\theta(a \mid s)$denote the stochastic actor policy, let $Q_{\phi_1}(s,a)$and $Q_{\phi_2}(s,a)$denote the twin critic networks, let $\bar{\phi}_1$and $\bar{\phi}_2$denote their target parameters, and let $\alpha_{\text{ent}} > 0$ denote the entropy temperature. The actor's objective is

$$J_\pi(\theta) = \mathbb{E}_{s_t \sim \mathcal{D},\, a_t \sim \pi_\theta}\left[\alpha_{\text{ent}} \log\, \pi_\theta(a_t \mid s_t) - \min_{m \in \{1,2\}} Q_{\phi_m}(s_t, a_t)\right]. \tag{4.46}$$

For critic $m \in \{1,2\}$, the training objective is

$$J_Q(\phi_m) = \mathbb{E}_{(s_t, a_t^{\text{exec}}, r_t, s_{t+1})\sim\mathcal{D}}\left[\left(Q_{\phi_m}(s_t, a_t^{\text{exec}}) - y_t\right)^2\right], \tag{4.47}$$

with target

$$y_t = r_t + \gamma\, \mathbb{E}_{a_{t+1}\sim\pi_\theta}\left[\min_{m\in\{1,2\}} Q_{\bar{\phi}_m}(s_{t+1}, a_{t+1}) - \alpha_{\text{ent}} \log\, \pi_\theta(a_{t+1} \mid s_{t+1})\right], \tag{4.48}$$

where $\gamma \in (0,1)$is the discount factor.

Constraint awareness in CH-SAC does not come from replacing the SAC objective itself. Instead, it arises from the combination of predictor-guided state construction, multi-objective reward shaping, hotspot-focused control, and guard-based safe action execution.

## 4.19 CH-SAC State Representation

To keep the reinforcement-learning control problem computationally manageable, the CH-SAC controller does not consume the full node-by-node fog state directly. Instead, it operates on a compact aggregated observation that summarizes current system status, selected hotspot nodes, predictive information, temporal context, recent performance, confidence-related indicators, and multi-objective control summaries.

### 4.19.1 Focus-Node Selection

At each control step $t$, the CH-SAC controller restricts direct control to an ordered set of $K$focus nodes. Let $\mathcal{V}$denote the set of fog nodes. Because FusionPredictor is the only predictive module in the framework, the only predictive term used in focus-node prioritization is the fused short-horizon arrival forecast $\hat{A}_{i,t+\tau}^{\text{fusion}}$. To keep this term dimensionally consistent with current node telemetry, it is normalized as

$$\bar{A}_{i,t+\tau}^{\text{fusion}} = \frac{\hat{A}_{i,t+\tau}^{\text{fusion}}}{\max\limits_{j\in\mathcal{V}} \hat{A}_{j,t+\tau}^{\text{fusion}} + \varepsilon}, \tag{4.49}$$

where $\varepsilon > 0$is a small constant that avoids division by zero. Let $q_{i,t} \in [0,1]$denote the normalized queue pressure and let $u_{i,t} \in [0,1]$denote the normalized resource utilization. The node-priority score is then defined as

$$\psi_{i,t} = \nu_p \bar{A}^{\text{fusion}}_{i,t+\tau} + \nu_q q_{i,t} + \nu_u u_{i,t}, \tag{4.50}$$

where $\nu_p, \nu_q, \nu_u \geq 0$are weighting coefficients satisfying

$$\nu_p + \nu_q + \nu_u = 1. \tag{4.51}$$

The ordered focus-node set is therefore

$$F_t = \text{TopK}_{\,i \in \mathcal{V}}(\psi_{i,t}, K), \mid F_t \mid = K. \tag{4.52}$$

This formulation keeps hotspot selection explicitly prediction-aware, while queue pressure and utilization provide the current congestion context of each node.

Within the selected set, the index $k \in \{1, \ldots, K\}$refers to the current focus-slot position rather than to a permanent node identity. Let $i_k(t) \in F_t$denote the node occupying slot $k$at time $t$. The slot-level focus-node descriptor is then written as

$$x^{\text{focus}}_{k,t} = x^{\text{node}}_{i_k(t),t}, \tag{4.53}$$

where $x^{\text{node}}_{i,t}$is the bounded node-level feature vector used by the CH-SAC state builder.

### 4.19.2 Aggregated Controller State

The aggregated control observation is written as

$$s_t = [z^{\text{sys}}_t, \phi^{\text{focus}}_t(F_t), \phi^{\text{activity}}_t, \phi^{\text{pred}}_t(F_t), \phi^{\text{time}}_t, \phi^{\text{perf}}_t, \phi^{\text{conf}}_t, \phi^{\text{mo}}_t]^{\top}. \tag{4.54}$$

In this representation, $z_t^{\mathrm{sys}}$summarizes the global system condition, such as aggregate utilization, congestion, and system-wide efficiency. The term $\phi_t^{\mathrm{focus}}(F_t)$encodes the currently selected focus nodes through bounded slot-level descriptors. The block $\phi_t^{\mathrm{activity}}$summarizes overall workload intensity and spatial spread. The block $\phi_t^{\mathrm{pred}}(F_t)$contains predictor-derived quantities associated with the focus nodes, including fused short-horizon arrival forecasts and bounded forecast summaries. The block $\phi_t^{\mathrm{time}}$provides temporal context, such as normalized time-of-day or control-step phase. The block $\phi_t^{\mathrm{perf}}$contains recent service-quality and control-performance indicators, including queue pressure, latency, and drop behavior. The block $\phi_t^{\mathrm{conf}}$contains confidence and stability indicators associated with prediction and control. Finally, $\phi_t^{\mathrm{mo}}$contains bounded multi-objective control summaries, such as efficiency, proactive behavior, and prediction provisioning alignment.

Through this compact representation, the controller observes both the present system condition and the predicted short-horizon demand trend without requiring a full node-by-node state tensor.

## 4.20 Action Space and Multi-Resource Control Mapping

The CH-SAC controller operates in a continuous multi-resource action space. At each control step $t$, the actor produces one action vector for each of the $K$currently selected focus slots. For focus slot $k$, the raw action is defined as

$$a_{k,t}^{\mathrm{raw}} = \begin{bmatrix} a_{k,t}^{\mathrm{cpu,raw}} \\ a_{k,t}^{\mathrm{mem,raw}} \\ a_{k,t}^{\mathrm{sto,raw}} \\ a_{k,t}^{\mathrm{bw,raw}} \end{bmatrix} \in [-1,1]^4, \tag{4.55}$$

and the full action vector is obtained by stacking the $K$slot-level actions:

$$a_t^{\mathrm{raw}} = [(a_{1,t}^{\mathrm{raw}})^\top, \dots, (a_{K,t}^{\mathrm{raw}})^\top]^\top \in [-1,1]^{4K}. \tag{4.56}$$

Positive values represent scale-up actions, negative values represent scale-down actions, and values near zero indicate little or no change. This formulation is more faithful than a scalar-capacity abstraction, because the controller adjusts CPU, memory, storage, and bandwidth separately rather than through a single undifferentiated capacity variable.

Before deployment, the raw action passes through the runtime safety layer. Let $\xi_t$denote the runtime state inspected by the guard logic. The executed action is therefore

$$a_t^{\text{exec}} = \mathcal{G}(a_t^{\text{raw}}, \xi_t), \tag{4.57}$$

where $\mathcal{G}(\cdot)$denotes the hybrid guard operator described in Section 4.22.

For focus slot $k$, let

$$c_{k,t} = \begin{bmatrix} \text{cpu}_{k,t} \\ \text{mem}_{k,t} \\ \text{sto}_{k,t} \\ \text{bw}_{k,t} \end{bmatrix} \tag{4.58}$$

denote the current resource vector and let $c_k^{\min}$and $c_k^{\max}$denote the corresponding node-specific lower and upper bounds. The executed action is mapped to a resource-change vector $\Delta c_{k,t}$, after which the resource state is updated by

$$c_{k,t+1} = \text{clip}\ \ , \left(c_{k,t} + \Delta c_{k,t}, c_k^{\min}, c_k^{\max}\right), \tag{4.59}$$

where the clipping operation is applied elementwise.

A compact abstract mapping from executed action to resource changes is

$$\Delta c_{k,t} = \begin{bmatrix} \Phi_{\text{cpu}}\left(a_{k,t}^{\text{cpu,exec}}, \hat{A}_{i_k(t),t+\tau}^{\text{fusion}}, u_{i_k(t),t}, q_{i_k(t),t}\right) \\ \Phi_{\text{mem}}\left(a_{k,t}^{\text{mem,exec}}, \hat{A}_{i_k(t),t+\tau}^{\text{fusion}}, u_{i_k(t),t}, q_{i_k(t),t}\right) \\ \eta_{\text{sto}}\, a_{k,t}^{\text{sto,exec}} \\ \eta_{\text{bw}}\, a_{k,t}^{\text{bw,exec}} \end{bmatrix}, \tag{4.60}$$

where $\hat{A}_{i_k(t),t+\tau}^{\text{fusion}}$is the fused short-horizon arrival forecast for the node occupying focus slot $k$, $u_{i_k(t),t}$is its normalized utilization, $q_{i_k(t),t}$is its queue-pressure indicator, and $\eta_{\text{sto}}, \eta_{\text{bw}} > 0$are fixed per-step scaling coefficients. Storage and bandwidth are treated as direct bounded adjustments, whereas CPU and memory are shaped by forecast-aware and utilization-aware logic.

Figure 4.1 confirms that the four-dimensional action space is operational rather than merely formal. Applied updates are observed in all four dimensions: CPU, memory, storage, and bandwidth. CPU is the most frequently active resource, with non-zero updates in 62.5% of control steps, while memory and storage remain active in 17.6% and 16.0% of steps, respectively. Bandwidth is used only rarely, with non-zero updates in 2.1% of control steps. When a dimension is active, the mean normalized applied change is 0.136 for CPU, 0.128 for memory, 0.813 for storage, and 0.010 for bandwidth. Because these magnitudes are normalized by each dimension's own bounded action range, they should be interpreted as relative use within each resource type rather than as direct physical comparability across units.

The behavioural pattern shown in Figure 4.1 is consistent with the controller design. CPU is the most frequently used adaptation lever because it is closely tied to short-horizon congestion prevention and service responsiveness. Memory is also used regularly, but less frequently than CPU. Storage exhibits the largest normalized magnitude when active, indicating that when this dimension is used, it is typically applied decisively rather than continuously. Bandwidth remains a rare and small adjustment dimension, which is consistent with the reward design, where over-reliance on bandwidth-only actions is explicitly discouraged. Overall, the figure shows that the proposed action formulation genuinely operates as a bounded four-dimensional control space, even though the most visible downstream QoS effects in later chapters remain CPU- and queue-centred.

This multi-resource formulation is important because it allows the controller to respond flexibly to different workload conditions while remaining computationally manageable. Instead of collapsing all provisioning into one scalar capacity variable, CH-SAC can express differentiated resource adjustments at the selected focus nodes, while the guard ensures that those actions remain safe and feasible before deployment. The reward design described next then determines how these resource decisions are evaluated during learning in terms of QoS, efficiency, proactive behaviour, and operational stability.

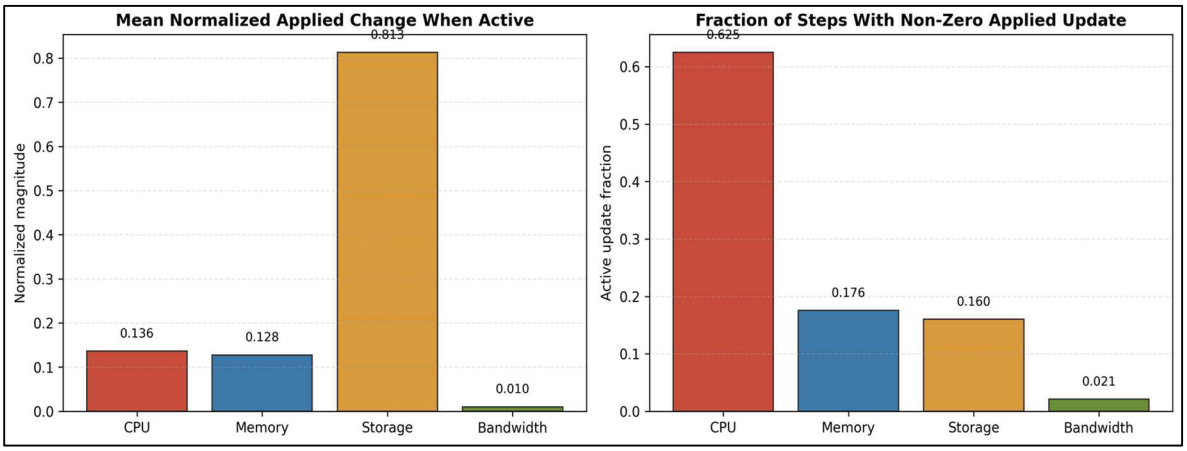


Figure 4.1 Operational 4D resource usage from CH-SAC (avg. over 3 runs)

## 4.21 Reward Design

The CH-SAC controller is trained using a shaped multi-objective reward that balances service quality, proactive prediction-aware provisioning, resource efficiency, and operational stability. The implemented reward is structured as a sum of positive components and penalty components. The raw reward is written as

$$r_t^{\text{raw}} = r_t^{\text{base}} - p_t, \tag{4.61}$$

with

$$r_t^{\text{base}} = r_t^{\text{action}} + r_t^{\text{mob}} + r_t^{\text{legacy}}, \tag{4.62}$$

and

$$p_t = p_t^{\text{action}} + p_t^{\text{idle}} + p_t^{\text{bandwidth}} + p_t^{\text{frequency}} + p_t^{\text{qos}} + p_t^{\text{efficiency}} + p_t^{\text{constraint}}. \tag{4.63}$$

Here, $r_t^{\text{action}}$denotes positive reward terms associated with useful control behavior, $r_t^{\text{mob}}$denotes mobility- and prediction-aware shaping terms, and $r_t^{\text{legacy}}$denotes retained baseline QoS-oriented reward components carried over from the earlier controller formulation. The penalty terms discourage excessive action magnitude, provisioning on low-value nodes, over-reliance on bandwidth-only actions, overly frequent reconfiguration, poor global efficiency, QoS degradation, and explicit constraint violation when constrained learning is enabled.

The QoS penalty is written as

$$p_t^{\text{qos}} = w_t^{\text{qos}}(p_t^{\text{queue}} + p_t^{\text{latency}}), \tag{4.64}$$

where $w_t^{\text{qos}}$is a weighting coefficient controlling the overall importance of queueing and latency penalties.

The mobility- and prediction-aware shaping component is written as

$$r_t^{\text{mob}} = 2r_t^{\text{corr}} + 3r_t^{\text{pred}} + r_t^{\text{act}} + r_t^{\text{pro}} + r_t^{\text{focus}} + r_t^{\text{dir}} - p_t^{\text{waste}}. \tag{4.65}$$

In this expression, $r_t^{\text{corr}}$rewards alignment between resource placement and mobility-driven demand migration, $r_t^{\text{pred}}$rewards effective exploitation of prediction signals, $r_t^{\text{act}}$rewards efficiency in active regions, $r_t^{\text{pro}}$rewards proactive rather than delayed provisioning, $r_t^{\text{focus}}$rewards alignment with the selected focus nodes, $r_t^{\text{dir}}$rewards consistency between action direction and system need, and $p_t^{\text{waste}}$penalizes low-value or wasteful provisioning.

The final reward used for learning is obtained after cost-aware adjustment and stabilization:

$$r_t = \text{Stabilize}\ , (\text{EconomicAdjust}\ (r_t^{\text{raw}})). \tag{4.66}$$

Here, $\text{EconomicAdjust}\ (\cdot)$incorporates cost sensitivity into the raw reward, while Stabilize, reduces reward volatility through deterministic bounded post-processing.

## 4.22 Constraint-Aware Guard Mechanism

Because fog-resource control is safety-critical and SLA-sensitive, CH-SAC includes a runtime guard layer that modifies raw actor outputs before they are applied. This layer is represented as a composition of operational protection stages:

$$\mathcal{G} = \mathcal{G}_{\text{bounds}} \circ \mathcal{G}_{\text{sla}} \circ \mathcal{G}_{\text{queue}} \circ \mathcal{G}_{\text{forecast}} \circ \mathcal{G}_{\text{activity}} \circ \mathcal{G}_{\text{bind}}, \tag{4.67}$$

with composition applied from right to left.

### 4.22.1 Focus-Node Binding

The actor outputs one action vector per focus slot, not per permanent node identity. Let $i_k(t) \in F_t$ denote the node currently occupying focus slot k. Because the focus set is re-evaluated at every control step, the slot index k identifies the current control position rather than a fixed fog node. The binding stage therefore maps the slot-level action to the corresponding runtime node:

$$a_{i_k(t),t}^{\text{bind}} = \mathcal{G}_{\text{bind}}(a_{k,t}^{\text{raw}}, i_k(t)). \tag{4.68}$$

This mapping is necessary because the physical node associated with a given slot may change over time as workload hotspots move across the fog topology.

### 4.22.2 Activity Gating

After slot-to-node binding, the activity-gating stage suppresses unnecessary scale-up on nodes that show no meaningful workload evidence and no predictor support. Operationally, this stage prevents the controller from wasting resources on inactive or weakly active nodes even if the raw actor proposal is positive. When the node occupying focus slot k exhibits recent arrival activity, non-negligible offered load, queue stress, or a meaningful fused short-horizon forecast, the bound action is passed forward to the next protection stage. Otherwise, the guard attenuates or nulls unnecessary scale-up while leaving harmless or stabilising actions unaffected. This mechanism preserves responsiveness to emerging hotspots while discouraging wasteful provisioning on persistently idle regions of the fog topology.

### 4.22.3 Forecast-Aware Scaling

The forecast-aware scaling stage adjusts the bound action according to the fused short-horizon forecast. Denoting the forecast for the node occupying slot $k$by $\hat{A}^{\text{fusion}}_{i_k(t),t+\tau}$, this stage is written abstractly as

$$a^{\text{for}}_{i_k(t),t} = \mathcal{G}_{\text{forecast}}(a^{\text{bind}}_{i_k(t),t}, \hat{A}^{\text{fusion}}_{i_k(t),t+\tau}). \tag{4.69}$$

This stage allows stronger proactive actions when the fused forecast indicates imminent pressure.

### 4.22.4 Emergency Queue Override

Let $d^{\text{med}}_{k,t}$denote the median queue delay, measured in milliseconds, at the node occupying focus slot $k$. The queue-emergency scale factor is defined as

$$\gamma(d_{k,t}^{\text{med}}) = \begin{cases} 1, & d_{k,t}^{\text{med}} < 500 \text{ ms}, \\ 1.3, & 500 \le d_{k,t}^{\text{med}} < 1000 \text{ ms}, \\ 1.8, & 1000 \le d_{k,t}^{\text{med}} < 2000 \text{ ms}, \\ 2.5, & d_{k,t}^{\text{med}} \ge 2000 \text{ ms}. \end{cases} \tag{4.70}$$

Under severe queue distress, CPU and memory overrides are applied as

$$\Delta\text{cpu}_{k,t} \leftarrow \text{cpu}_{k,t}(\gamma(d_{k,t}^{\text{med}}) - 1), \tag{4.71}$$

$$\Delta\text{mem}_{k,t} \leftarrow \text{mem}_{k,t}(\gamma(d_{k,t}^{\text{med}}) - 1). \tag{4.72}$$

### 4.22.5 SLA Guardrails

For any resource dimension $j \in \{\text{cpu}, \text{mem}, \text{sto}, \text{bw}\}$, the SLA guardrail is written as

$$\Delta c_{k,t}^{j} = 0, \text{if } \Delta c_{k,t}^{j} < 0 \text{ and } \text{SLA}_{k,t}^{\text{unhealthy}} = 1, \tag{4.73}$$

$$\Delta c_{k,t}^{j} \leftarrow 1.15\, \Delta c_{k,t}^{j}, \text{if } \Delta c_{k,t}^{j} > 0 \text{ and } \text{SLA}_{k,t}^{\text{unhealthy}} = 1. \tag{4.74}$$

Thus, scale-down is blocked during service distress, while recovery-oriented scale-up actions are mildly amplified.

### 4.22.6 Bounds Enforcement

Finally, the bounds stage clips all resource changes to node-specific feasible ranges before application. This ensures that the action executed remains safe, feasible, and operationally meaningful under overload, queue escalation, or SLA deterioration.

## 4.23 Training and Runtime Execution

The CH-SAC control loop operates as a closed online decision process. Online policy refinement is carried out primarily in the trace-driven YAFS environment, while Mininet is used to execute the same logical control policy under explicit network, compute, and queue constraints for stress-based validation.

At each control step $t$, telemetry is collected from the current fog environment, including utilization, queue state, throughput, latency, drop behavior, and mobility-driven workload indicators. FusionPredictor then updates its internal states and produces the bounded short-horizon forecast $\hat{A}_{i,t+\tau}^{\text{fusion}}$together with any associated diagnostic indicators. Using these quantities, the controller observation $s_t$is constructed from the current telemetry, the selected focus-node information, and the predictor outputs.

The actor then produces the raw action

$$a_t^{\text{raw}} = \pi_\theta(s_t), \tag{4.75}$$

after which the guard layer generates the executed action

$$a_t^{\text{exec}} = \mathcal{G}(a_t^{\text{raw}}, \xi_t). \tag{4.76}$$

The executed action is mapped to node-level resource adjustments and applied through the runtime platform. The environment then evolves under the updated configuration, producing new queues, latency, utilization, and service outcomes. Once the system response is observed, the shaped reward $r_t$is computed and the next state $s_{t+1}$is formed.

The replay-buffer transition stored for learning is therefore

$$(s_t, a_t^{\text{exec}}, r_t, s_{t+1}), \tag{4.77}$$

where $a_t^{\text{exec}}$denotes the action applied to the environment after guard correction. These transitions are then used to update the actor and critic networks through replay-buffer-based learning while the control loop continues.

# CHAPTER 5

# CONTROLLER BEHAVIOUR AND ABLATION

## 5.1 Chapter Overview

This chapter provides the behavioural validation of the proposed framework. Whereas Chapter 4 defined the predictor, controller, guard, and online execution loop, the purpose of this chapter is to explain why the framework behaves differently from reactive baselines during execution. The central argument is that prediction changes the timing of control: resource adjustments are issued before queue growth becomes dominant, rather than after congestion has already formed.

The chapter begins by examining how fused short-horizon prediction changes controller behaviour in general. It then validates the forecasting signal through node-level prediction accuracy and error analysis, before showing how those predictions are converted into proactive resource actions through proactive headroom, overload prediction, precision–recall behaviour, lead-time distributions, and the temporal relation between taxi load and provisioning activity. The next group of sections explains how CH-SAC concentrates control on focus nodes, aligns resource allocation with mobility demand, and maintains consistent service behaviour despite infrastructure-level latency outside the controller's direct authority.

The chapter then turns to SAC-family ablation under three stress regimes: CPU stress, queue stress, and hotspot or flash-crowd stress. These ablations are not full all-controller benchmark comparisons. Their role is to isolate the separate contributions of baseline SAC, the runtime guard, and the Fusion Predictor inside the SAC family. Together, these sections show that the framework is not only accurate in forecasting, but also genuinely proactive, spatially selective, and behaviourally stable. Prediction is most decisive when overload signals are delayed, as in CPU- and queue-stress conditions, whereas under abrupt flash-crowd stress the guard provides

most of the immediate safety benefit, with prediction mainly improving redistribution and hotspot alignment.

The remainder of the chapter is organised as follows. Section 5.2 explains why prediction changes control outcomes and how the controller shifts from reactive to anticipatory operation. Section 5.3 validates the short-horizon forecasting signal. Section 5.4 analyses forecast-driven proactive scaling, while Section 5.5 studies focus-node prioritisation and spatial demand dynamics. Sections 5.6, 5.7, and 5.8 then examine latency behaviour, fog-tier delay decomposition, and ingress latency. Section 5.9 considers QoS stability and recovery behaviour. Finally, Sections 5.10, 5.11, and 5.12 present SAC-family ablation under CPU stress, queue stress, and hotspot or flash-crowd stress, followed by continuous-learning diagnostics in Section 5.13 and a synthesis of behavioural insights in Section 5.14.

## 5.2 Why Prediction Changes Control Outcomes

The integration of the Fusion Predictor changes the controller's operating regime mainly by changing the timing of decisions. In the non-predictive setting, baseline SAC operates in a reactive loop: scaling decisions lag behind workload growth, queues accumulate, and recovery becomes slow once congestion has formed. When the Fusion Predictor is integrated into the control loop, the controller becomes anticipatory. Scale-up actions are issued before overload fully materialises, helping keep the system in a stable operating region rather than allowing it to enter persistent congestion.

This difference is visible in the dynamics of queue pressure and latency. Under predictive control, queue pressure rises only briefly during early load increases and then remains close to zero for most of the run, while latency stabilises at a lower level. In the non-predictive case, queue pressure continues to rise, remains elevated, and translates into sustained latency inflation. Once the queue saturates, reactive control struggles to drain it efficiently, producing runaway delay accumulation. The key change is therefore not simply that the controller scales more often, but that it scales earlier, before queues begin to dominate service time.

The causal mechanism is the temporal ordering introduced by prediction-guided scaling: prediction peaks first, proactive scaling follows, and queue stability is preserved afterwards. Early

signals of rising arrivals trigger scale-up actions before queue pressure becomes severe, so additional capacity is already active when the workload surge reaches the fog nodes. This means the framework prevents congestion rather than merely recovering from it after the fact. Effective proactive control does not require perfect prediction. For control purposes, the predictor mainly needs to detect rising demand early, even if extreme peaks are somewhat underestimated. Once these trend-level signals are available, CH-SAC can move from reactive load-following to anticipatory provisioning, reducing queue pressure, limiting latency growth, and improving recovery under stress.

Prediction also improves control quality without forcing aggressive over-provisioning. The number of active nodes remains bounded, while the guard constrains unstable reactions. In this way, the controller does not achieve stability by blindly adding capacity; instead, it uses short-horizon forecasts to place resources where and when they are needed. This is especially important in fog systems, where bursty mobility-driven demand can overwhelm a purely reactive controller within only a few control intervals.

The benefit of prediction lies mainly in changing decision timing rather than simply increasing control sensitivity. The proposed framework acts before congestion becomes visible in raw performance metrics, whereas non-predictive control reacts only after queues and latency have already deteriorated. This shift from reactive recovery to proactive congestion prevention is the main behavioural reason why the predictor-enhanced controller outperforms non-predictive baselines under stress.

## 5.3 Prediction Accuracy and Model Validation

### 5.3.1 Node-Level Prediction Accuracy

Figure 5.1 compares predicted and observed short-horizon taxi counts at the node level. Most points cluster close to the diagonal, showing that the predictor captures short-term spatial demand variation well. The distribution is naturally denser in the low- to mid-count range, reflecting the skewed Beijing workload, while a mild positive bias appears at higher counts. For proactive control, this slight conservatism is operationally acceptable because it favours earlier

capacity allocation in emerging hotspot regions rather than delayed reaction. Both axes are measured as taxi counts per fog node over the short-horizon prediction window.

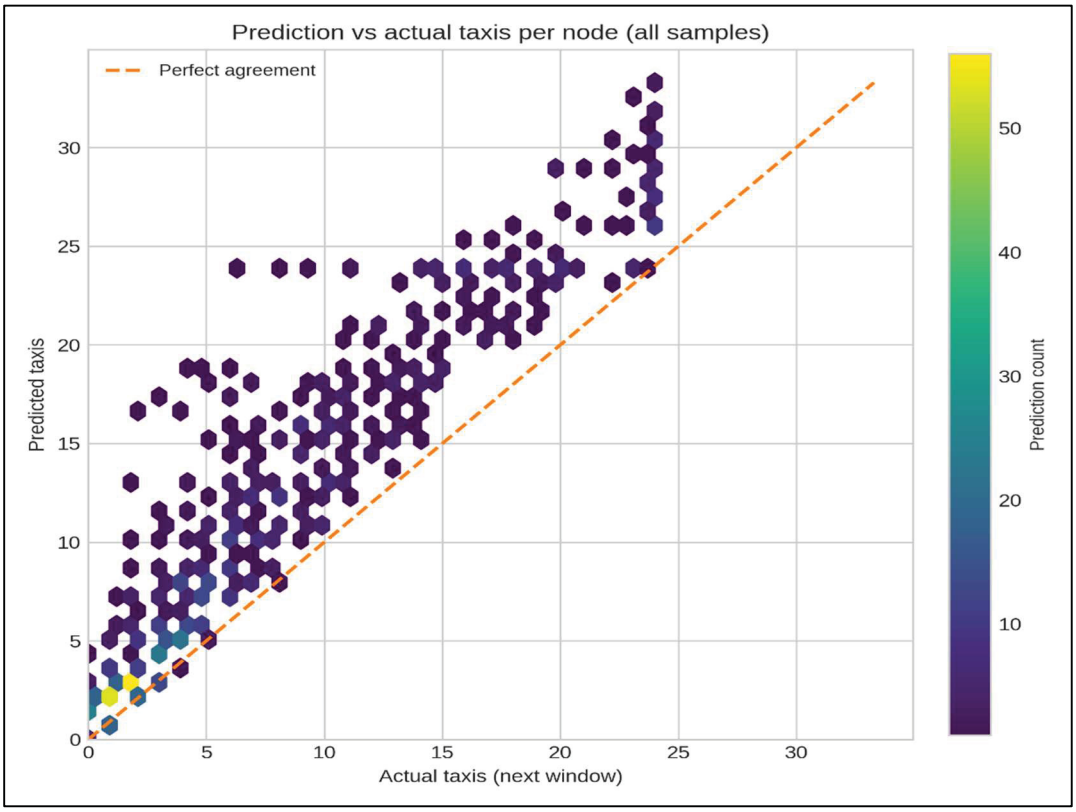


Figure 5.1 Prediction vs actual taxis per node

### 5.3.2 Prediction-Error Analysis

Figure 5.2 confirms this result statistically. The error distribution is centered near zero, and the cumulative absolute-error curve shows that small deviations dominate. The predictor achieves MAE = 3.45 taxis and Pearson correlation r = 0.95; approximately 50% of forecasts lie within about ±3 taxis and 90% within about ±7.2 taxis. Overall, the forecasting module is accurate and slightly conservative, which is desirable for short-horizon proactive fog control. Prediction

error is measured in taxis, while the cumulative curve reports the proportion of samples whose absolute error is below each threshold.

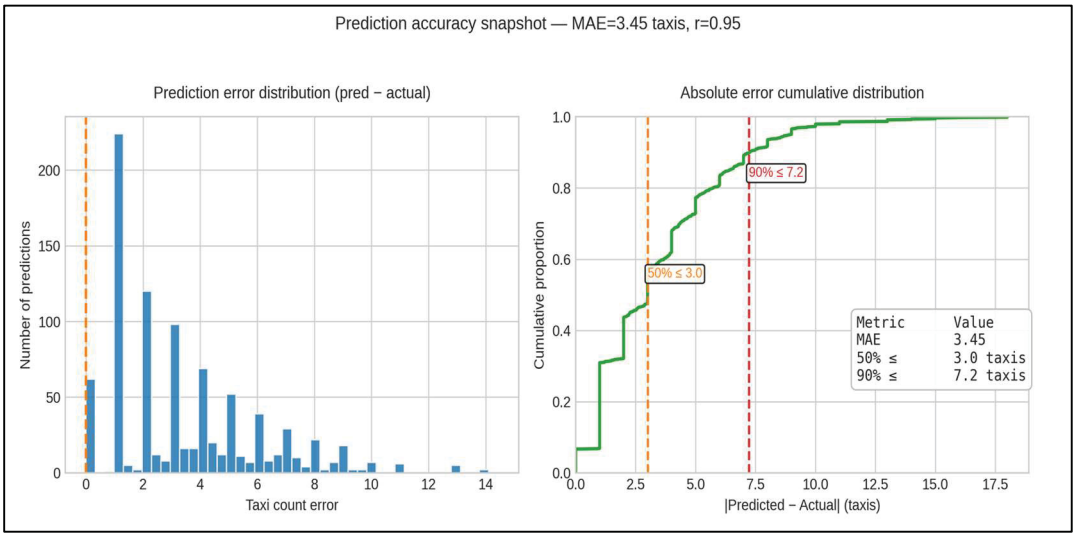


Figure 5.2 Prediction error analysis

### 5.3.3 Prediction Accuracy and Reliability

The node-level prediction observation comparison and the statistical error analysis show that the forecasting subsystem provides accurate and slightly conservative short-horizon estimates that are well suited to real-time fog control. The strong correlation between predicted and observed taxi counts, the low MAE, and the steep cumulative error profile indicate that the predictor tracks short-term mobility fluctuations closely. This level of accuracy is sufficient for control because the main requirement is early detection of rising demand rather than exactly matching every peak magnitude.

## 5.4 Forecast-Driven Proactive Scaling

Figure 5.3 shows the cumulative number of proactive and reactive decision-step scale-up actions over time. Proactive actions are those issued before overload becomes visible according to the proactive-action criterion used in this thesis, whereas reactive actions are triggered only after overload or QoS degradation has already appeared. The shaded gap between the curves

represents the cumulative advantage of anticipatory control. During the initial phase, both curves rise at similar rates as the controller explores the action space and calibrates its response to mobility-driven load fluctuations. As the run progresses, the proactive curve steepens while the reactive curve grows more slowly, indicating the emergence of a stable predictive strategy. By the end of the simulation, the proactive lead reaches 533 actions, showing that the policy increasingly acts ahead of congestion rather than after the onset of degradation. This controller-level behaviour should be interpreted separately from any node-level proactive-share metric, since one decision step may affect multiple nodes. The horizontal axis represents simulation time in seconds (s), and the vertical axis reports the cumulative number of scale-up decisions.

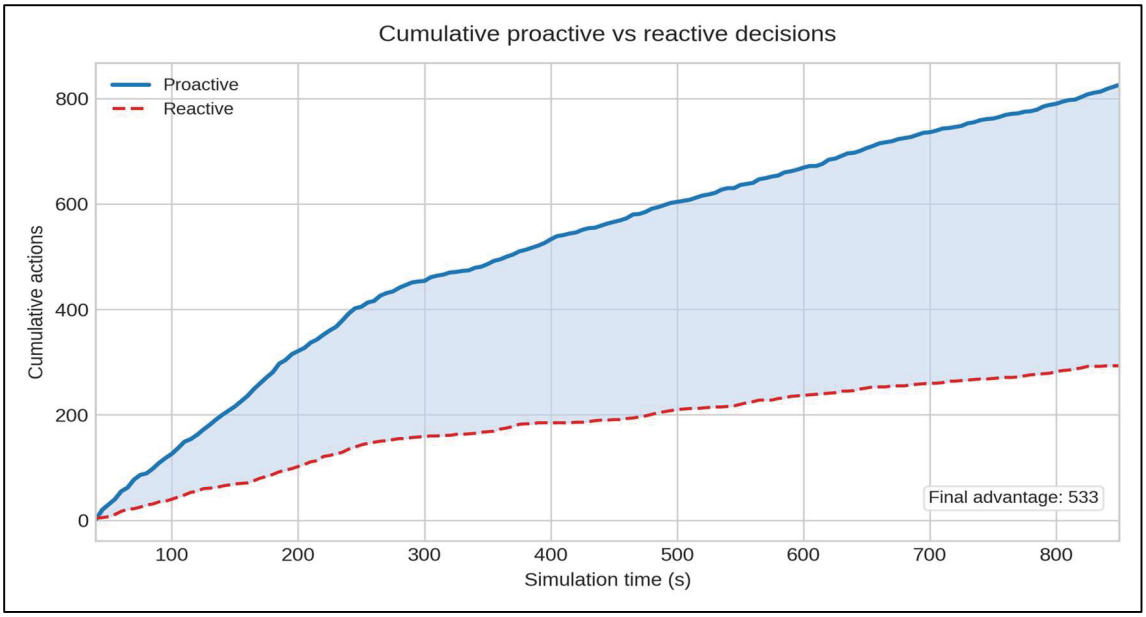


Figure 5.3 cumulative proactive and reactive decisions over time

Figure 5.4 examines whether these early actions are actually guided by forecasts. It plots predicted short-horizon arrivals against the magnitude of proactive CPU scale-up actions and shows a positive relationship, with Pearson $r = 0.26$ and Spearman $\rho = 0.33$. Although moderate, this correlation is behaviourally meaningful in a multi-factor control setting where scaling decisions are also influenced by queue pressure, latency, recent utilisation, and smoothness constraints. At lower forecast levels, most actions cluster around modest CPU increments,

which can be interpreted as lightweight preparatory allocations. As predicted load increases, larger scale-ups become more frequent. The pattern is therefore proportional rather than aggressive: the controller responds to forecast growth but remains constrained by other operational objectives. This shows that proactive actions are forecast-driven, not merely early. Predicted arrivals are measured as short-horizon taxi/request counts, while proactive CPU scale-up magnitude represents the CPU resource increment applied by the controller.

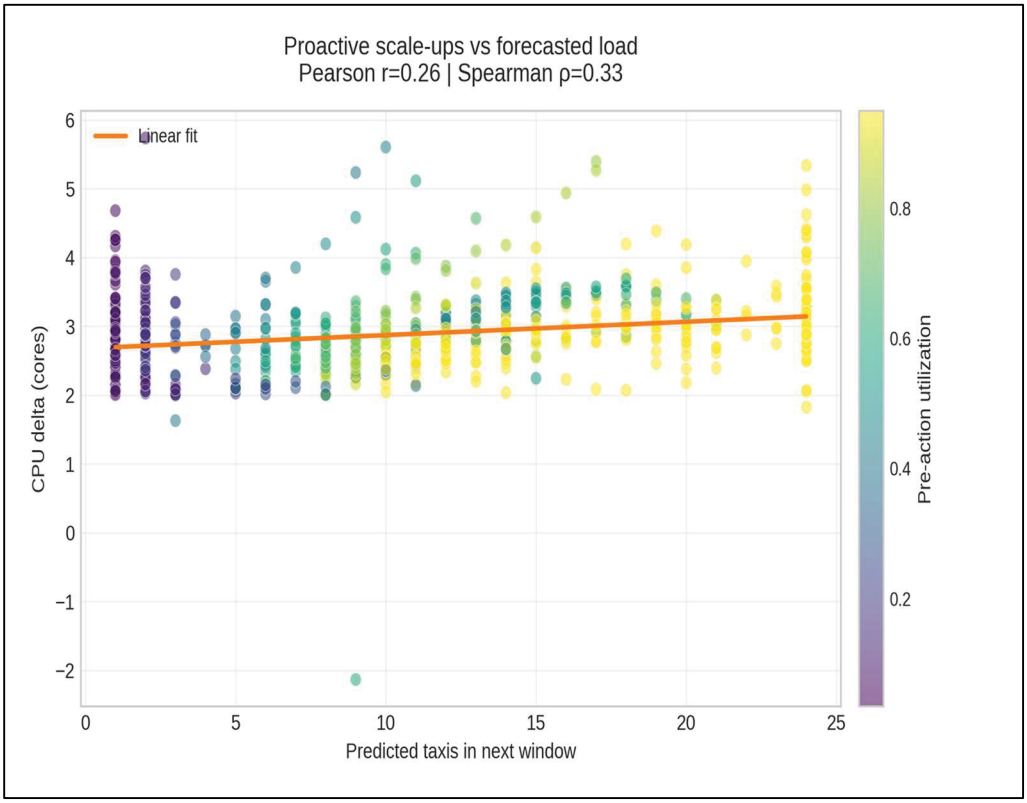


Figure 5.4 Predicted arrivals vs proactive CPU scale-up

Figure 5.5 strengthens this interpretation by summarising the same behaviour with 95% confidence bounds over n = 893 proactive events. The correlation between predicted load and CPU scale-up magnitude is r = 0.25, with a 95% confidence interval of [0.19, 0.31], confirming that larger forecasted surges tend to trigger larger proactive allocations and that this trend is not driven by a small number of outliers. The second relationship is much stronger: proactive CPU

allocation is correlated with post-action utilisation at r = 0.81, with a 95% confidence interval of [0.79, 0.83]. This indicates that the extra CPU is not wasted; it is later absorbed by the workload and therefore represents efficient anticipatory provisioning rather than idle over-provisioning. Pearson and Spearman correlations are dimensionless statistical measures in [-1,1], and the confidence intervals report uncertainty around these correlation values.

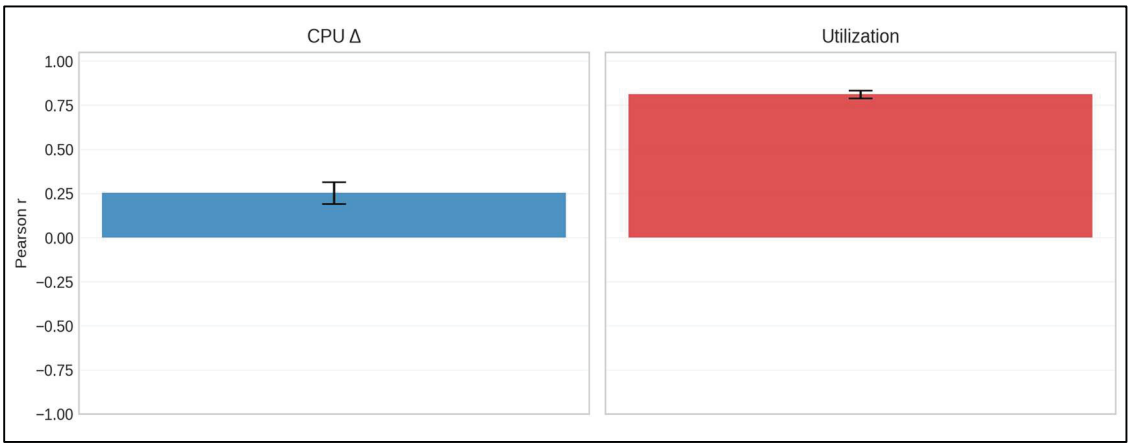


Figure 5.5 Proactive correlations with 95% confidence intervals

### 5.4.1 Proactive Headroom Profile

Figure 5.6 shows the share of proactive actions in each time bin, with bars indicating the total number of actions per bin and the dashed line marking the 50% reference level. Across the 23 analysed bins, the overall proactive share is 78.2%, meaning that most scale-up decisions occur before overload or QoS degradation becomes visible.

After a short adaptation phase, the controller settles into a predominantly proactive regime. The proactive share remains mostly between 80% and 90%, well above the 50% baseline, indicating that the policy relies mainly on anticipatory rather than corrective scaling once it has stabilised. This suggests that the framework maintains headroom at busy nodes instead of waiting for congestion to form first.

Taken together, the cumulative-action, forecast–action, correlation, and proactive-share results show that proactive actions are not only forecast-linked and efficiently used, but also sustained over time. Proactive share is a ratio reported as a percentage or fraction of total scale-up actions in each time bin, while the bars represent action counts.

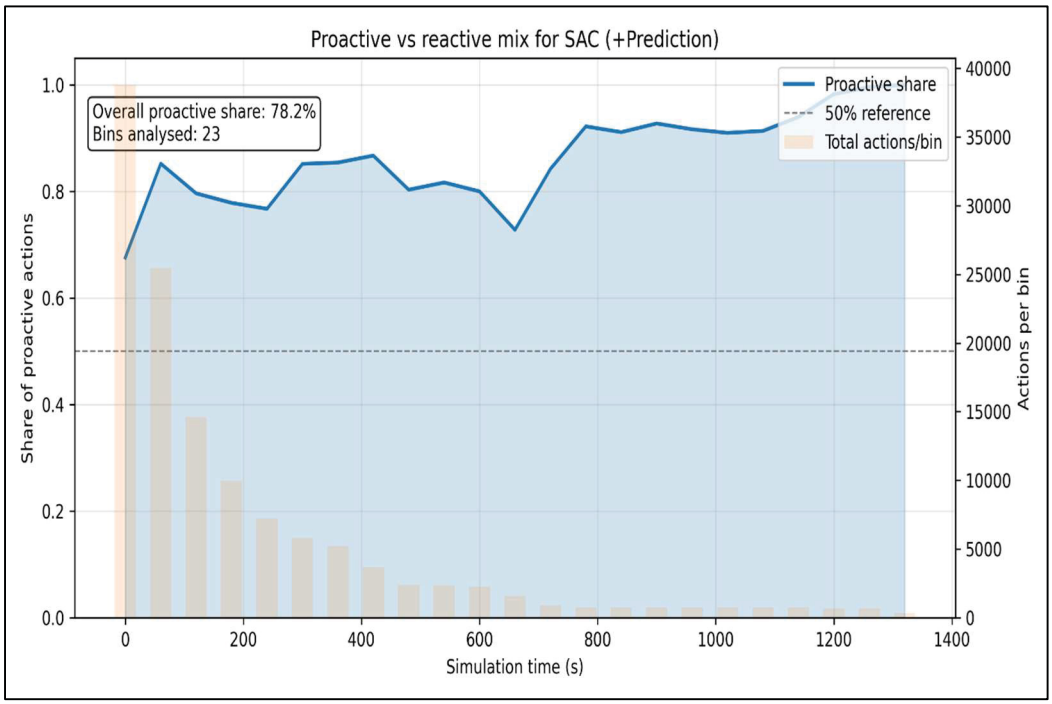


Figure 5.6 Proactive headroom over time

### 5.4.2 Proactive Overload Prediction Accuracy

Figure 5.7 evaluates whether proactive interventions correspond to genuine upcoming overloads. A prediction is counted as correct when a node is scaled before an overload episode with lead ≥ 2 taxi arrivals. The matrix contains 542 true positives, 2 false positives, 132 false negatives, and 1,227 true negatives.

These counts give precision = 0.996, recall = 0.804, and F1 = 0.890. The very small number of false positives shows that proactive warnings are highly selective, while the large number

of true positives indicates that most overload episodes are detected in advance. Some overloads are still missed, but the controller captures the majority before congestion fully develops.
Overall, the confusion-matrix result shows that predictive scale-ups are rarely wasted and are operationally useful for preventing overload escalation. The confusion-matrix entries are sample counts. Precision, recall, and F1-score are dimensionless classification metrics in [0,1], where values closer to 1 indicate better overload-prediction performance.

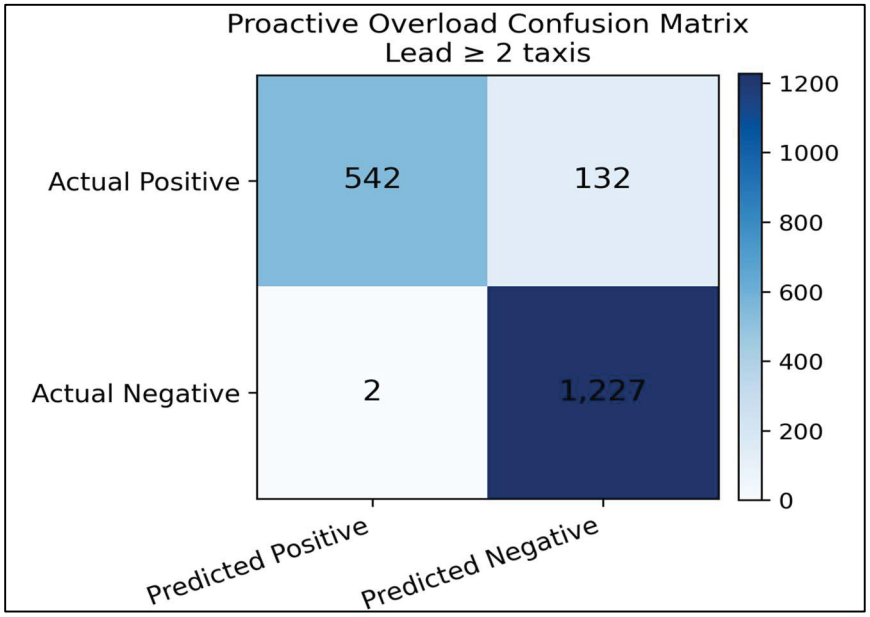


Figure 5.7 Confusion matrix for proactive overload prediction

### 5.4.3 Precision–Recall Analysis

Figure 5.8 complements the confusion matrix by showing performance across decision thresholds. The main result is that the controller maintains strong discrimination over a wide range of operating points. After the very low-recall region, precision remains broadly in the high-0.8 range across most of the curve, with average precision ≈ 0.87.
This means the system can increase recall without a severe collapse in precision. In practice, that gives flexibility: a deployment can use a stricter threshold to minimise unnecessary scale-

ups or a more permissive one to maximise overload protection. Proactive overload prediction is therefore not tied to a single threshold but remains robust across operating conditions.

The confusion-matrix and precision–recall results show that the controller's anticipatory behaviour is both strong at the chosen operating point and robust across thresholds. Precision and recall are dimensionless values in [0,1]; precision measures the fraction of predicted overload events that are correct, while recall measures the fraction of actual overload events detected by the predictor.

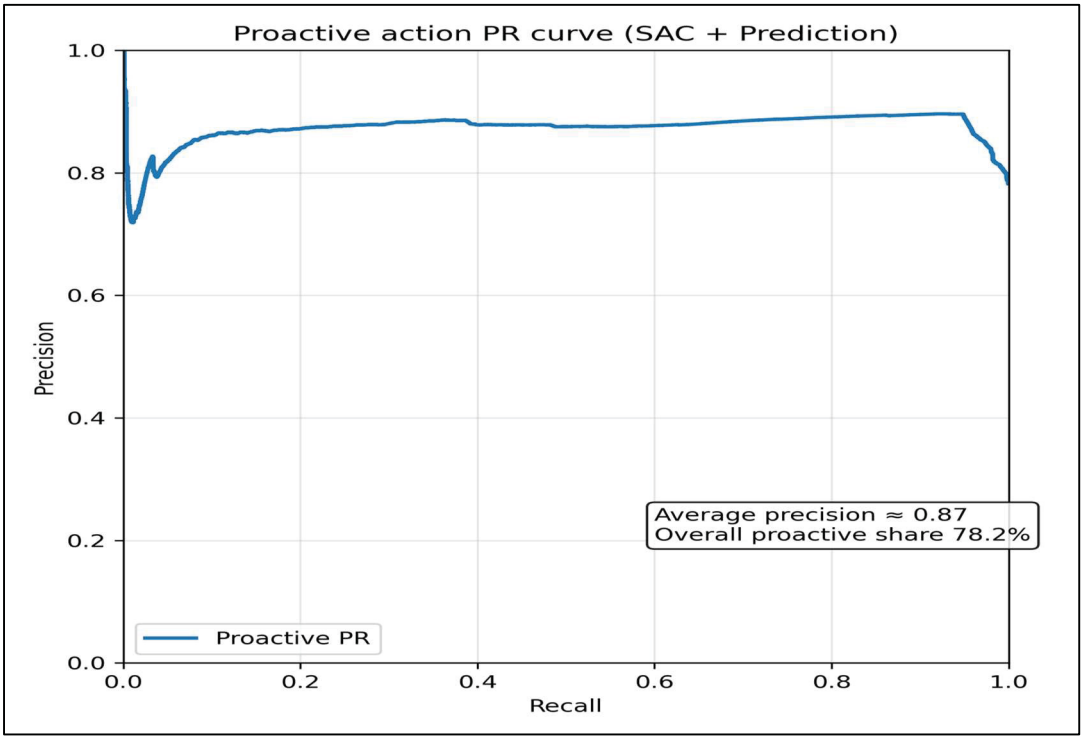


Figure 5.8 Precision–recall curve for proactive overload prediction

### 5.4.4 Spatial Distribution of Proactive Behaviour

Figure 5.9 shows the proactive share for the fifteen most active nodes. Most of these nodes exhibit very high proactive shares, often close to 100%, while several others remain in the 80–90% range. Only a small number of lower-activity nodes show clearly lower proactive shares.

This pattern indicates that proactive provisioning is concentrated where it is most useful: nodes that face heavier or more persistent demand are managed mainly in advance, whereas quieter nodes rely more on occasional reactive correction. The controller therefore adapts not only over time, but also across space, concentrating proactive headroom in the parts of the topology where congestion risk is highest. Proactive share is reported as the fraction or percentage of actions at each fog node that occur before overload becomes visible

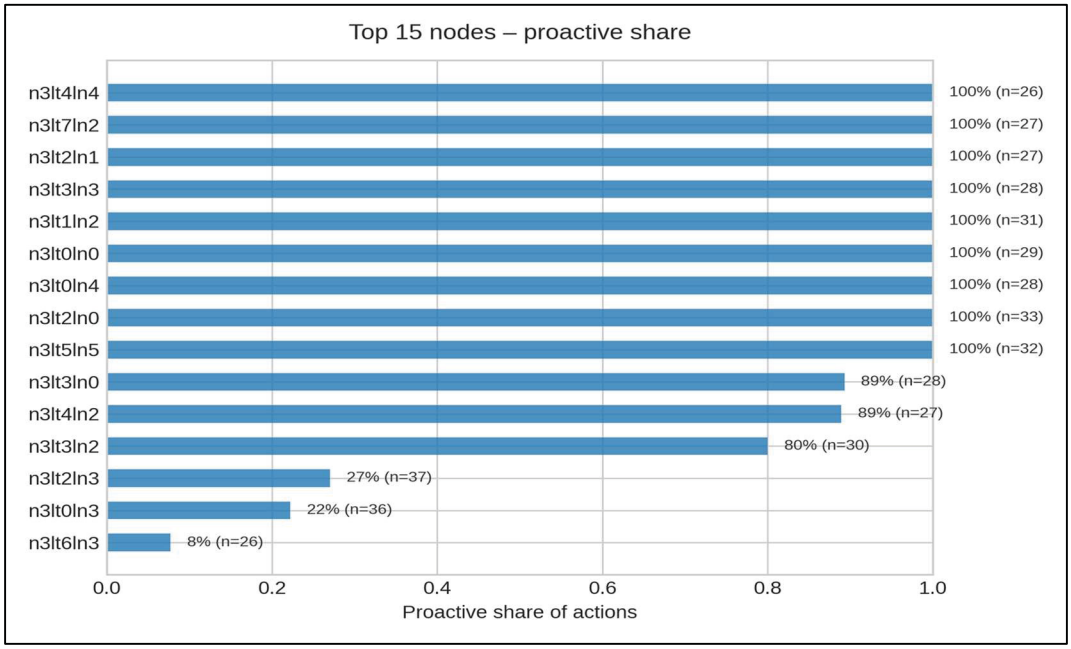


Figure 5.9 Proactive provisioning share across top 15 fog nodes

### 5.4.5 Lead-Time Distribution of Proactive Scaling Actions

Figure 5.10 shows the distribution of node-level lead times between a proactive scale-up and the arrival of the associated workload burst. Positive lead time means that provisioning occurs before the burst reaches the node.

Across 13,689 proactive events, most lead times fall between roughly 5 s and 35 s, with a median of 17.0 s. The 90th percentile is 148.0 s, there are no late proactive events (< 0 s), and the maximum observed lead time reaches 223.0 s. These results confirm that proactive actions

are usually issued early enough for added resources to become effective before queues build. The long right tail also shows that some rising demand patterns are identified far in advance. Overall, Figure 5.10 provides direct evidence that the controller repeatedly converts short-horizon forecasts into timely interventions rather than reacting after overload has already formed. Lead time is measured in seconds (s); positive values indicate that scaling occurs before the associated workload burst reaches the node.

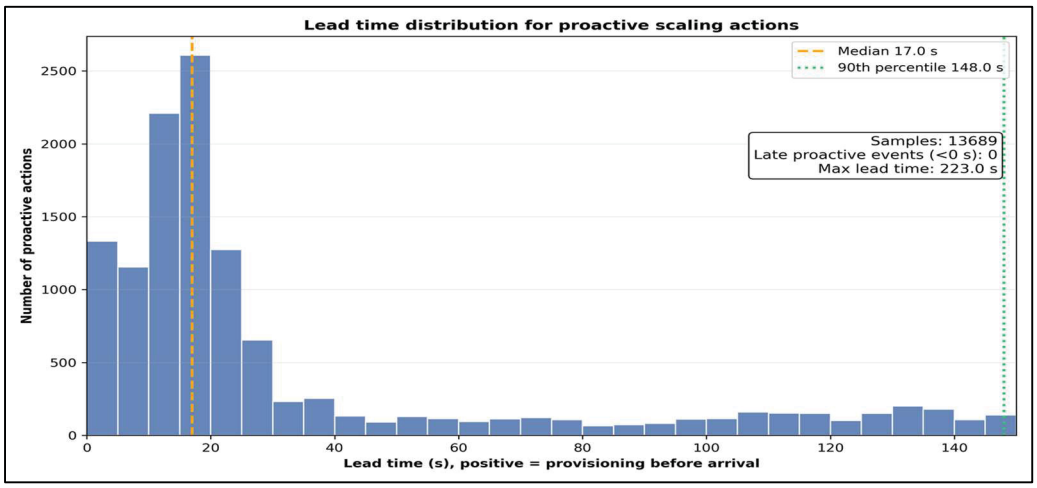


Figure 5.10 Lead-time distribution for proactive scale-up

### 5.4.6 Temporal Evolution of Load and Provisioning

Figure 5.11 compares system-wide taxi load with proactive and reactive provisioning activity over time. The top panel shows taxi message rate, while the lower panels show proactive and reactive actions in each 30-second interval. Both workload and provisioning activity are high in the early phase, when the controller is still adapting to rapid changes. As the run stabilizes, proactive actions remain present but become smoother, while reactive interventions decline more sharply. Toward the end of the simulation, both load and actuation frequency decrease.This temporal pattern shows that provisioning remains coupled to mobility dynamics throughout the run, while the control regime gradually shifts from mixed reactive–proactive behavior to a smoother and predominantly predictive mode. Taxi load is measured as

message/request rate over time, while provisioning activity is measured as the number of proactive or reactive actions per 30-second interval.

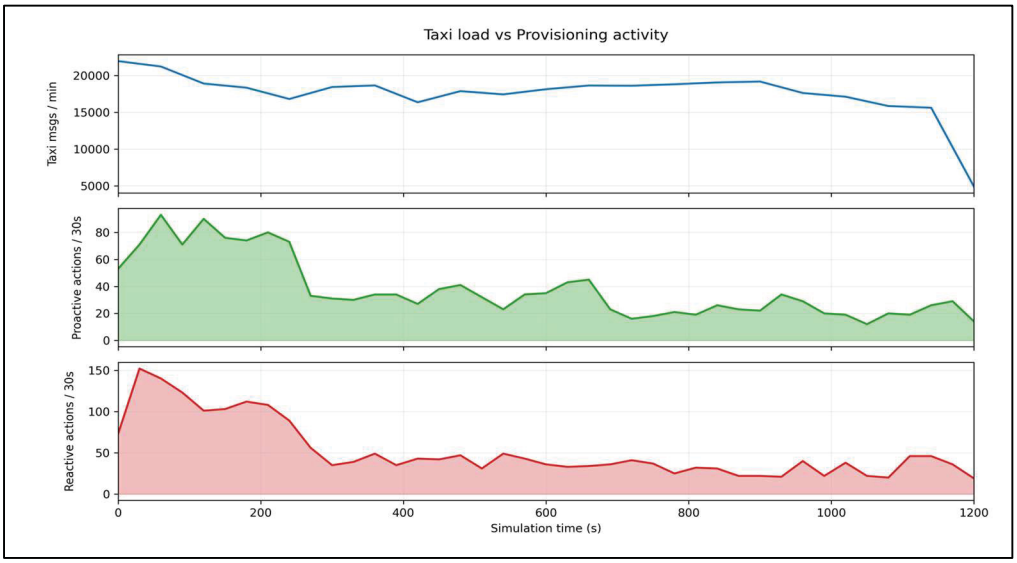


Figure 5.11 Taxi load and provisioning activity over time

## 5.5 Focus-Node Prioritization and Spatial Demand Dynamics

The proposed framework does not treat all fog nodes as equally urgent at every control step. Instead, it concentrates control on a small set of focus nodes whose predicted or observed demand is most critical. This is necessary because the workload is both spatially concentrated and temporally shifting: a small subset of nodes carries most of the traffic at any given time, but the identity of those dominant nodes changes across successive windows.

This behaviour justifies the use of a dynamic top-$K$ hotspot abstraction. Rather than monitoring the full topology with equal emphasis, the controller tracks the nodes most likely to experience near-term pressure. The fused predictor strengthens this process by selecting nodes using expected demand rather than current congestion alone. As a result, CH-SAC directs attention and scaling effort to the parts of the topology where intervention is most likely to matter.

This mechanism is particularly important under hotspot migration. If control reacts only after queues begin to grow, resources may be shifted too late because the dominant hotspot may already have moved. Prediction-guided focus selection reduces this delay by allowing the

controller to follow demand as it moves across the city. Focus-node prioritisation is therefore not only a dimensionality-reduction strategy; it is a key part of how the framework converts short-horizon prediction into spatially targeted proactive control.

### 5.5.1 Spatial Mobility–Resource Alignment

Figure 5.12 shows the spatial relationship between predicted taxi demand and CPU provisioning across the top 30 active nodes. Nodes ranked higher by predicted arrivals generally receive more CPU, which indicates that resource allocation follows the spatial structure of expected mobility demand rather than being applied uniformly across the topology.

The alignment is not perfectly one-to-one, and that is useful rather than problematic. Some mid-ranked nodes receive slightly more CPU than their forecast alone would suggest, which is consistent with safety-oriented smoothing and spillover protection near hotspot boundaries. The overall pattern shows that proactive control is spatially selective: the controller scales where demand is expected to emerge, while retaining some continuity around neighbouring nodes. Predicted demand is measured as expected taxi/request count, while CPU provisioning represents relative CPU allocation intensity across the selected fog nodes.

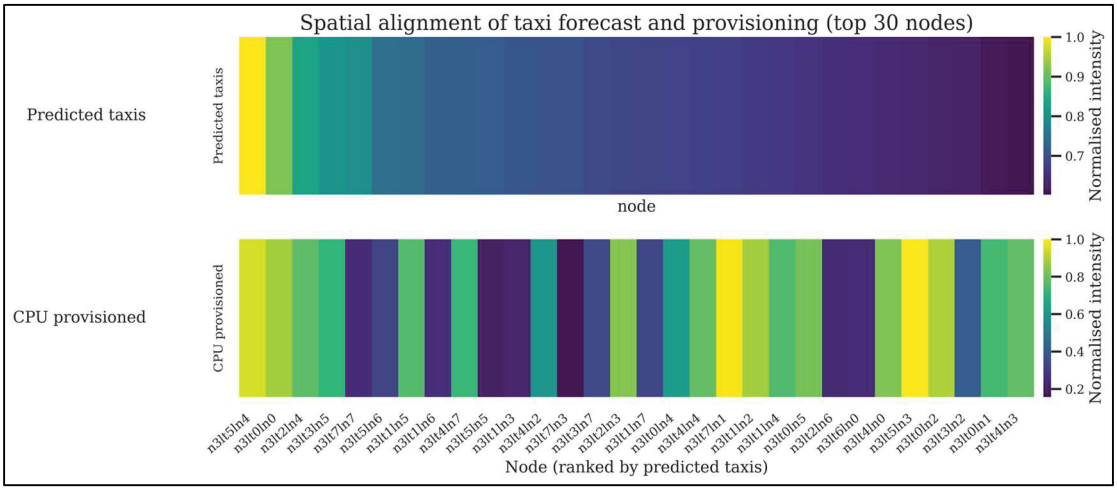


Figure 5.12 Predicted demand vs CPU provisioning across top 30 fog nodes

### 5.5.2 Mobility Forecast–Provisioning Correlation

Figure 5.13 shows a positive relationship between predicted taxi load and CPU scaling magnitude. The densest region lies at moderate predicted loads and moderate CPU increments, which suggests that most control actions are small, frequent, and stabilising rather than extreme.

At higher forecast levels, the spread becomes wider, indicating that scaling decisions also depend on additional context such as headroom, neighbouring conditions, and safety constraints. Even so, the upward trend confirms that predicted mobility remains a primary driver of resource allocation. The controller therefore behaves proactively but not mechanically: forecasted load guides the action, while the final adjustment remains controlled.

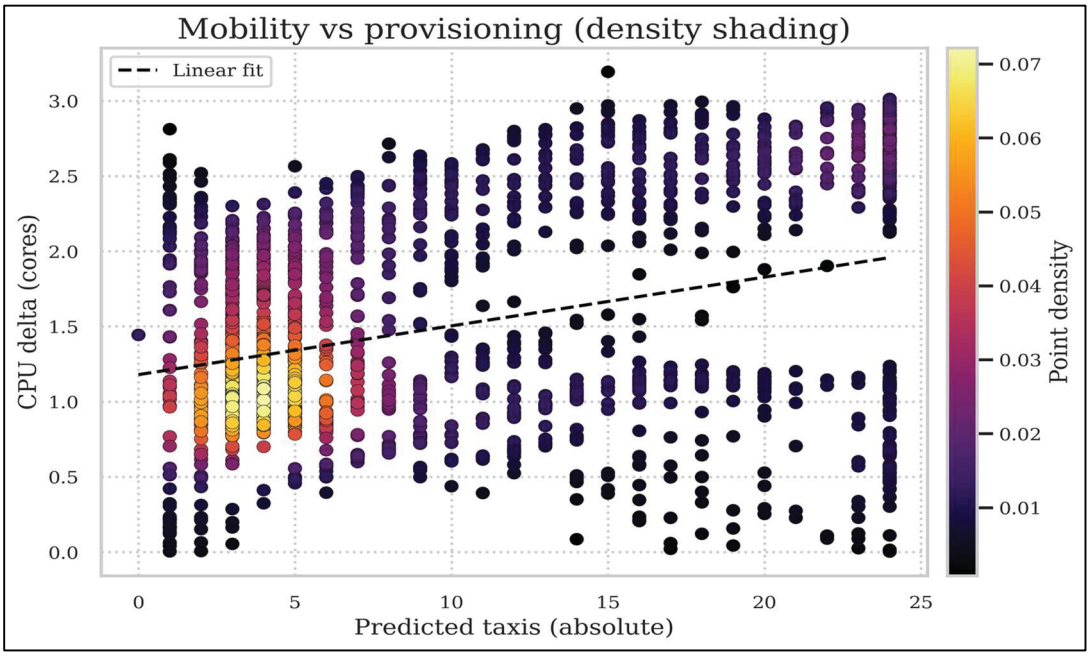


Figure 5.13 Predicted taxi load vs CPU scaling across active fog nodes

### 5.5.3 Mobility–Provisioning–Utilization Correlation.

Figure 5.14 adds current utilisation to the forecast-action relationship. The positive slope remains visible, confirming that higher predicted load generally leads to larger CPU increments.

The utilisation overlay shows that scaling decisions are not driven by forecast alone; they are also conditioned by the node's current operating state.

Most actions occur while utilisation is still moderate, which is consistent with proactive control. High-utilisation events appear more often at the upper end of the predicted-load range, where stronger interventions are occasionally needed. The figure shows that the controller combines foresight with live system state: it acts because demand is expected to rise, but it also adapts the action to current operating pressure. Utilisation is normalized in [0,1], where 0 indicates no use and 1 indicates full utilisation; CPU scaling magnitude represents the applied CPU adjustment.

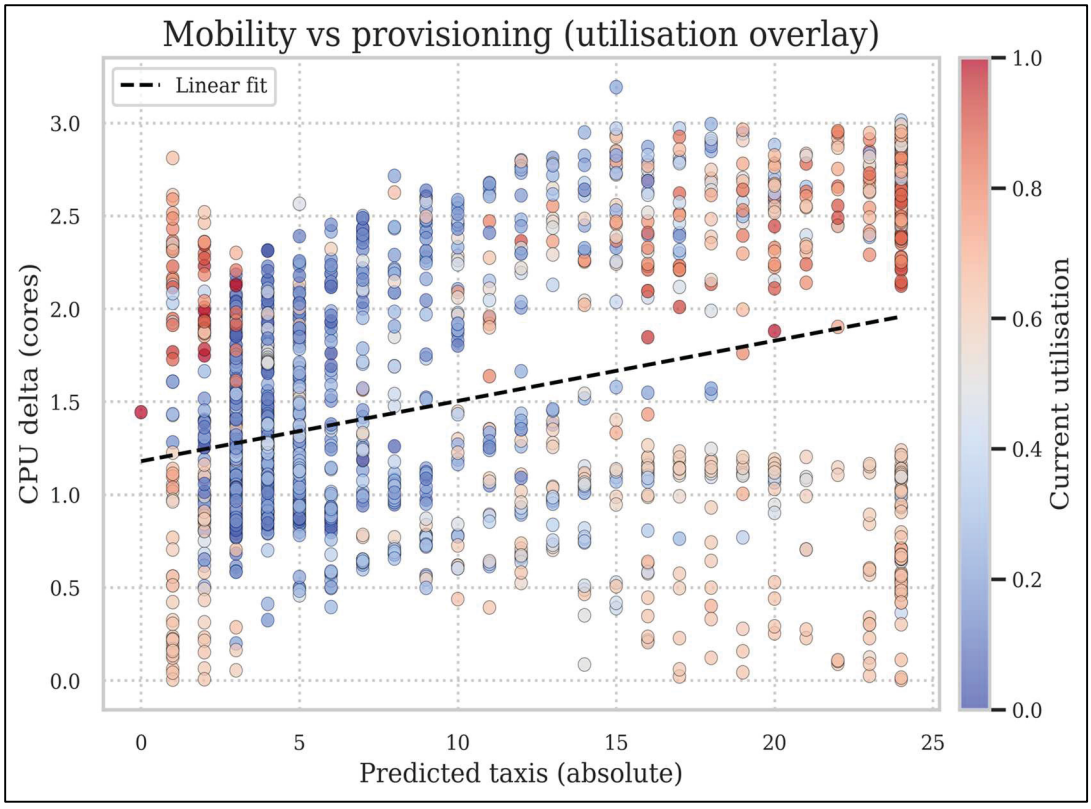


Figure 5.14 Predicted load vs CPU scaling with utilisation

### 5.5.4 Correlation between CPU Allocation and Mobility Load

Figure 5.15 compares utilisation levels for proactive and reactive actions. Proactive actions occur over a higher utilisation range overall, whereas reactive actions are more dispersed and include many lower-utilisation cases. This suggests that proactive provisioning is concentrated in the structurally busier parts of the topology, where demand pressure is persistent enough to justify anticipatory scaling.

The figure therefore supports the same interpretation as the spatial analysis above: proactive control is not applied uniformly everywhere. It is concentrated at nodes that are more operationally important, while reactive actions remain a secondary correction mechanism for less critical or more irregular conditions.

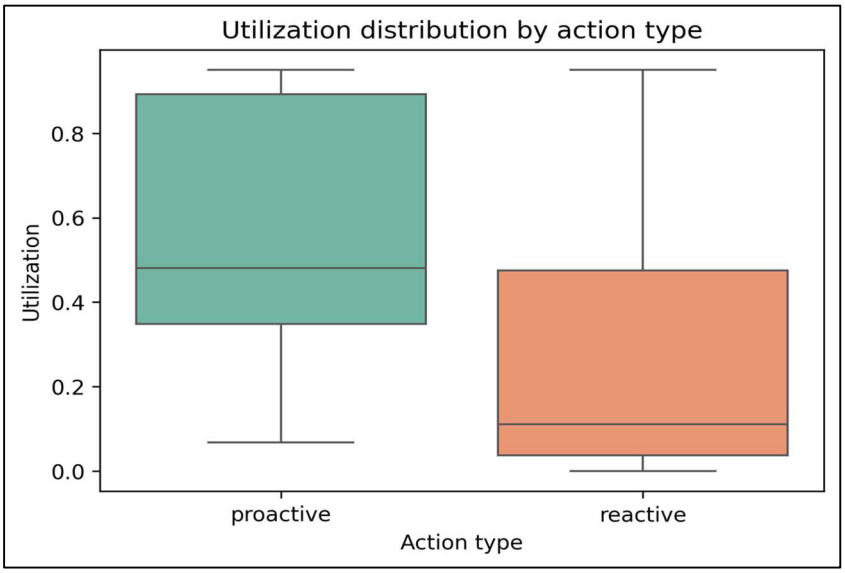


Figure 5.15 Node utilisation by action type

## 5.6 CH-SAC Latency Performance Overview

Figure 5.16 shows end-to-end latency over the full YAFS simulation. The rolling mean follows the workload cycle smoothly, which indicates stable adaptation under mobility-driven demand. Although latency remains above the 5 s SLA for much of the run, the main result is the absence of abrupt oscillations or collapse-like spikes. The controller therefore improves temporal

stability even when absolute latency is limited by the environment. In this run, the average latency is 31,340.9 ms, the 95th percentile is 85,600.0 ms, and SLA compliance is 13.45%. The latency distribution is strongly right-skewed, with a long tail extending to high values. This indicates that a large share of end-to-end delay comes from transmission and routing overhead rather than local fog computation. The figure should therefore be interpreted as evidence of stable controller behaviour under fixed network constraints, not as a failure of the controller to optimise the components it can influence.

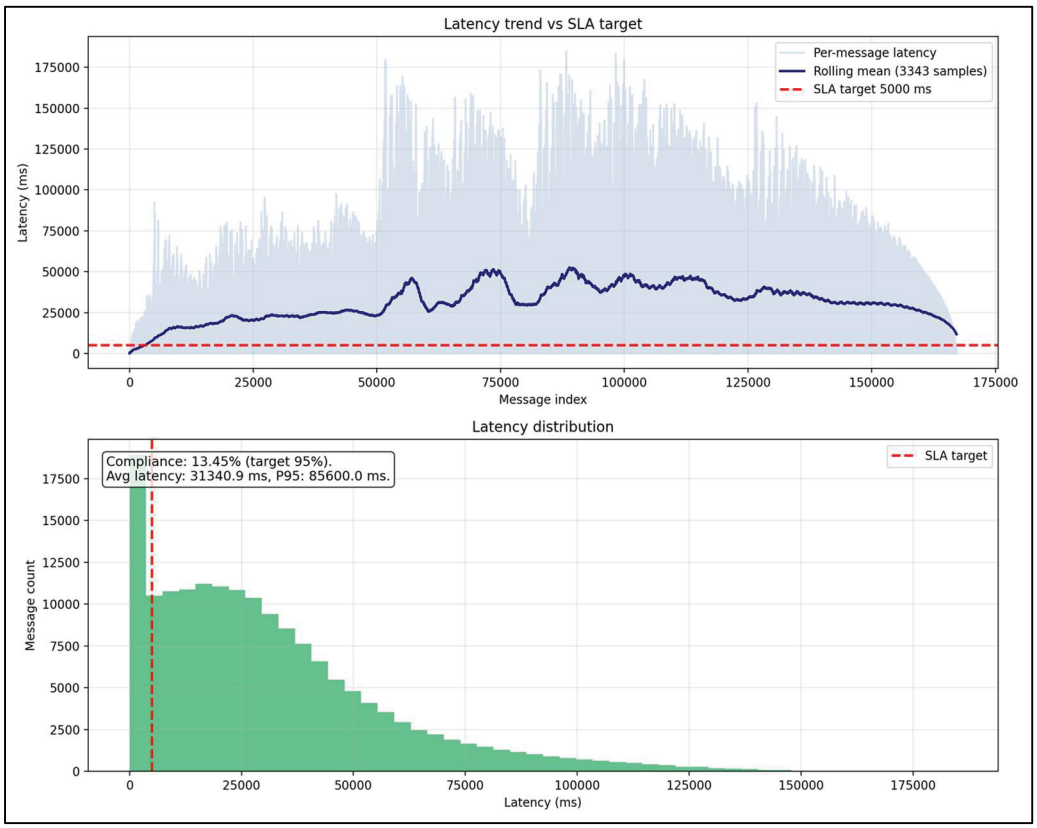


Figure 5.16 CH-SAC latency performance overview

## 5.7 Fog-Tier Latency Decomposition

Figure 5.17 shows that FogProcessor stage latency remains in the microsecond range, confirming that local computation is not a bottleneck. By contrast, Figure 5.18 shows that ingress latency is much larger and more variable, with visible spikes during high-

demand periods. Together, these results show that residual end-to-end delay is dominated by transport and routing effects, while the controller's contribution is to reduce instability and limit queue buildup at the fog tier.

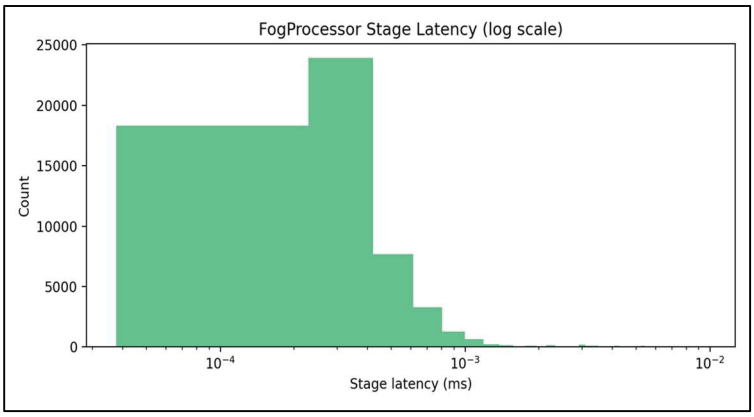


Figure 5.17 Stage Latency

## 5.8 Ingress Latency (Sensor → FogProcessor)

Figure 5.18 reports ingress latency, defined as the time between message emission at the sensor and arrival at the FogProcessor. In contrast to the microsecond-scale stage latency, ingress latency operates at a much larger time scale and is strongly shaped by network-level conditions such as gateway congestion, routing contention, and backhaul delay.

The figure shows substantial variability in ingress latency, with occasional pronounced spikes during periods of heavy mobility-driven traffic. At the same time, the rolling-mean profile remains comparatively smooth, indicating that the controller mitigates some of the volatility indirectly through predictive placement and scaling. Although the controller cannot change physical link speeds or routing topology, it can reduce queue formation in front of overloaded fog nodes by allocating capacity earlier and maintaining better spatial alignment between demand and provisioning.

With the FogProcessor-stage figure, the ingress-latency plot clarifies a central boundary of controllability in this thesis: fog-tier computation is effectively fully provisioned, while the dominant remaining delay arises from structural network overhead. The abstract file explicitly

notes that end-to-end latency includes components outside the controller's direct influence, such as backhaul links and sink-tier processing, and that these infrastructure effects limit absolute SLA compliance even when internal controller behaviour remains strong.

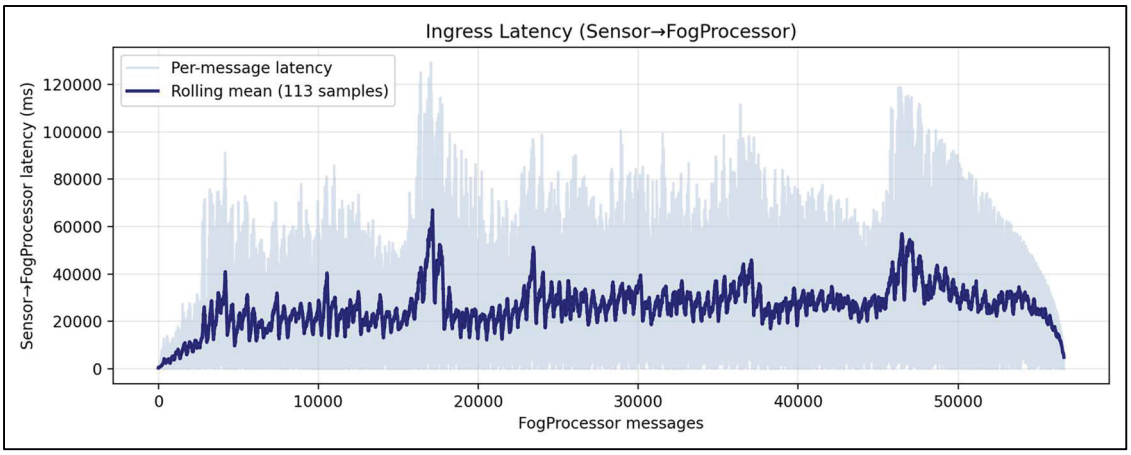


Figure 5.18 Ingress Latency

## 5.9 QoS Stability and Recovery Behaviour

### 5.9.1 QoS Stability Measurement

Figure 5.19 reports the QoS stability measurement and its final value of 59.6%. This metric captures how consistently the system remains close to its SLA target over time, rather than measuring only the fraction of requests that strictly satisfy the end-to-end delay bound. In the present YAFS behavioural layer, absolute compliance is constrained in part by backhaul, routing, and sink-side delay components outside the controller's direct action space. The important result is therefore not only the absolute compliance level, but the fact that service behaviour remains bounded and non-erratic throughout the run.

The final stability index of 59.6% indicates that the controller maintains a moderately stable service regime despite strong mobility-driven demand variation. This supports the behavioural interpretation developed in earlier sections: predictive scaling helps prevent abrupt degradation and keeps the system away from collapse-like oscillation even when it cannot fully eliminate

every source of end-to-end delay. QoS stability is reported as a percentage, where higher values indicate more consistent behaviour relative to the SLA target.

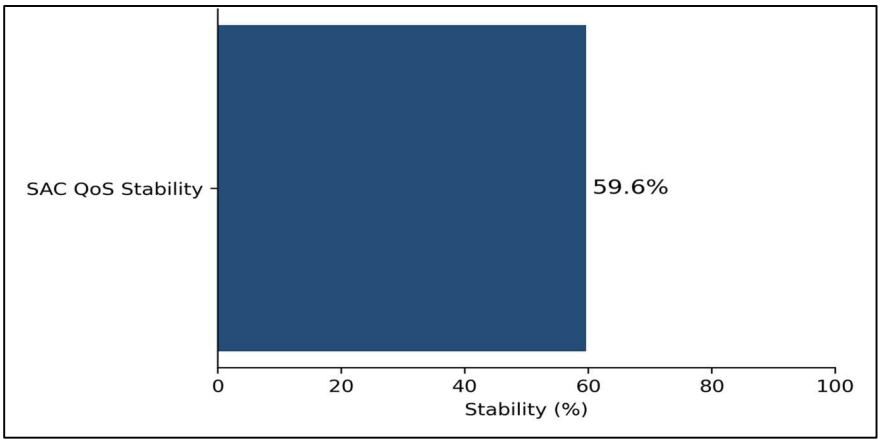


Figure 5.19 QoS Stability Measurement

### 5.9.2 QoS Stability and Compliance Over Time

Figure 5.20 shows how QoS stability and compliance evolve over time. After a short warm-up period, the stability curve converges and remains close to 60% for most of the simulation, even though instantaneous SLA compliance remains constrained by infrastructure-level delay outside the controller's direct authority. The key point is that the trajectory becomes stable rather than erratic.This distinction matters because QoS level and QoS stability are not the same. Absolute compliance reflects all end-to-end delay sources, including those beyond fog-tier control, whereas QoS stability reflects how consistently the controller avoids severe fluctuation and prolonged degradation. The temporal behaviour in Figure 5.20 therefore supports the claim that the proposed framework improves the consistency of service even when it cannot fully eliminate network-level delay. QoS stability and SLA compliance are reported as percentages over simulation time.

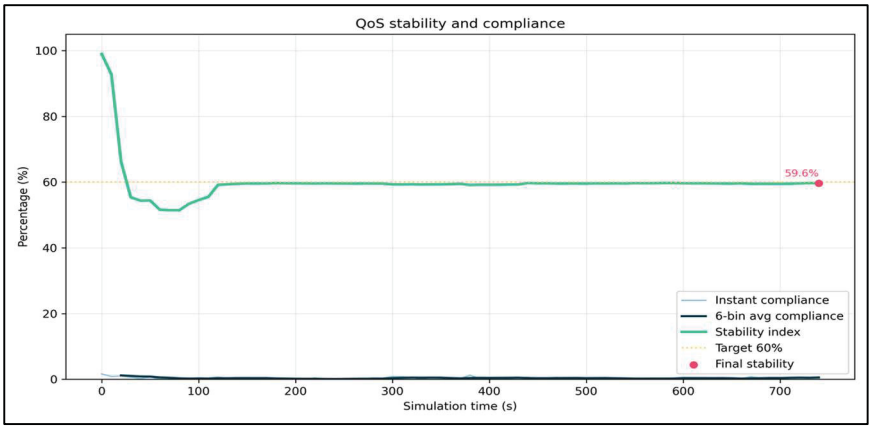


Figure 5.20 QoS Stability

### 5.9.3 Violation Burst Durations

Figure 5.21 measures the duration of continuous SLA-violation periods. The median burst duration is 180 s, which serves as a practical indicator of recovery time after disturbance. This metric is useful because it reflects how long the system remains in a degraded state when explicit queue-length measurements are unavailable. Shorter and more contained bursts suggest that predictive scaling helps limit backlog growth and restore steadier operation more quickly.

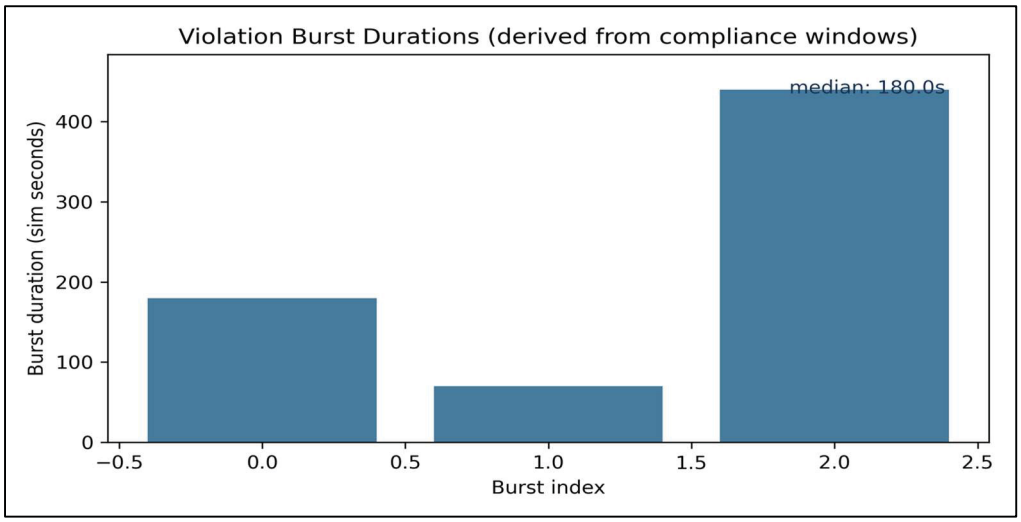


Figure 5.21 Violation Burst Duration

## 5.10 Guard and Prediction under CPU Stress

The CPU-stress scenario provides the clearest view of the different roles of the guard and the Fusion Predictor. Here, congestion develops because compute capacity becomes the bottleneck, so delayed scaling quickly leads to queue growth and service collapse. This subsection reports a SAC-family ablation experiment and should be interpreted as component-isolation evidence rather than as a duplicate of the full multi-controller comparison reported later in Chapter 6.Baseline SAC performs poorly in this regime. Mean latency reaches 4823.0 ms, mean queue pressure rises to 0.171, drop rate reaches 80.9%, and completed throughput falls to 37,199 requests. This is a collapse regime rather than a simple degradation: the controller reacts too late to prevent sustained overload.

Adding the guard substantially improves survivability. Mean latency drops to 986.4 ms, mean queue pressure falls to 0.054, and drop rate decreases to 22.1%, while throughput rises to 149,325 processed requests. This shows that the guard is effective as a safety mechanism once overload becomes visible. However, the behaviour remains reactive because intervention still happens after congestion signals appear.

The full controller, with both guard and prediction, performs much better. Relative to guard-only SAC, mean latency falls further to 117.0 ms, mean queue pressure to 0.007, and drop rate to 5.3%, while throughput increases to 182,145 processed requests. The key difference is timing: prediction allows scale-up before queues enter the growth regime. In this scenario, the guard preserves survivability, while prediction provides the decisive preventive control.

Figure 5.22 summarises this SAC-family CPU-stress ablation, showing mean latency, mean queue pressure, drop rate, and throughput for baseline SAC, SAC + Guard, and SAC + Guard + Fusion Predictor. Latency is reported in milliseconds (ms), queue pressure is normalized in [0,1], drop rate is reported as a percentage, and throughput is measured as completed requests.

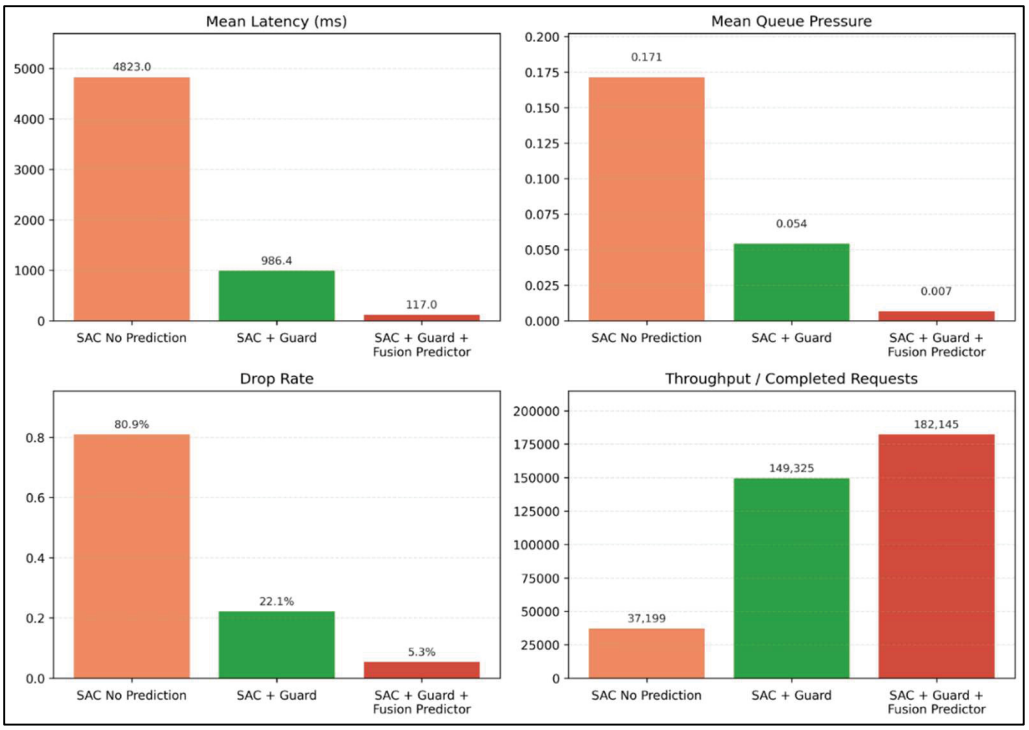


Figure 5.22 SAC-family ablation under CPU stress

## 5.11 Guard and Prediction under Queue Stress

The queue-stress scenario tests a different failure mode: buffers saturate faster than a reactive controller can respond. In this setting, delayed action is especially costly because once a queue enters sustained growth, recovery becomes slow and expensive. This subsection reports a SAC-family ablation experiment and should be interpreted as component-isolation evidence rather than as a duplicate of the full multi-controller comparison reported later in Chapter 6.

Baseline SAC again performs poorly, with mean latency of 2755.8 ms, mean queue pressure of 0.163, a drop rate of 78.9%, and a delivery ratio of only 21.1%. These results indicate that reactive SAC fails to manage queue growth under sustained offered load, leading to wide-spread request loss and unstable service.

Guard-only SAC improves survivability substantially. Mean latency falls to 270.5 ms, mean queue pressure drops to 0.031, drop rate decreases to 13.0%, and delivery increases to 87.0%. Even so, the controller remains reactive because the guard intervenes only after queue growth

has already become visible.When prediction is added, the controller moves into a much stronger operating regime. Mean latency falls further to 70.9 ms, mean queue pressure to 0.005, drop rate to 3.5%, and delivery rises to 96.5%. The reason is again temporal ordering: prediction peaks first, scale-up follows, and backlog growth is suppressed before it becomes self-reinforcing.

Figure 5.23 summarises this SAC-family queue-stress ablation, showing mean latency, mean queue pressure, drop rate, and delivery ratio for SAC No Prediction, SAC + Guard, and SAC + Guard + Fusion Predictor. This scenario shows even more clearly than CPU stress that prediction need not be perfect to be useful. What matters is early recognition of rising pressure, not exact peak matching. In queue-limited systems, that early warning is the difference between staying in a stable region and entering runaway congestion. Mean latency is reported in milliseconds (ms), mean queue pressure is normalized in [0,1], drop rate and delivery ratio are percentages, and the metrics are computed under queue-stress conditions.

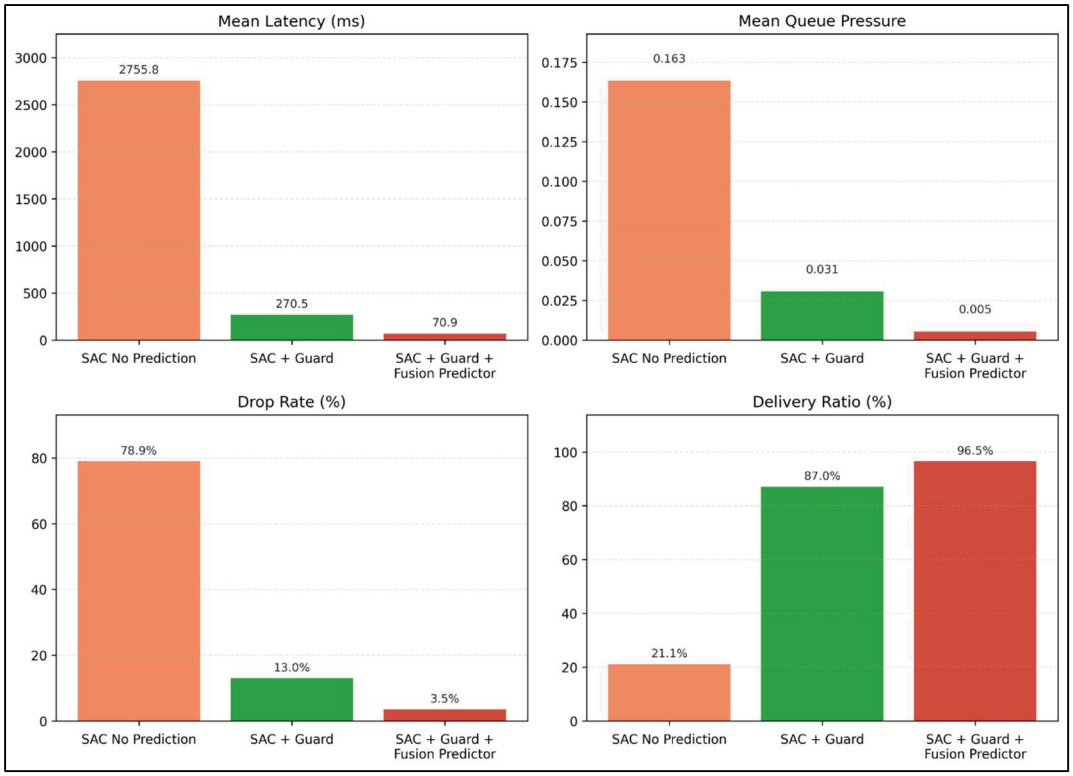


Figure 5.23 SAC-family ablation under queue stress

## 5.12 Guard and Prediction under Hotspot (Flash-Crowd) Stress

The hotspot or flash-crowd scenario produces abrupt spatial overload rather than gradual saturation. Demand becomes sharply concentrated at a small set of nodes and then migrates, so the main challenge is fast spatial adaptation rather than long lead-time anticipation. This subsection reports a SAC-family ablation experiment and should be interpreted as component-isolation evidence rather than as a duplicate of the full multi-controller comparison reported later in Chapter 6.Baseline SAC again performs poorly, with mean latency of 1409.3 ms, mean queue pressure of 0.070, and a delivery ratio of only 33.7%. These values indicate that unconstrained SAC cannot react fast enough once demand becomes suddenly concentrated in a hotspot region.

By contrast, both guarded variants remain stable. SAC + Guard reduces mean latency to 69.7 ms, lowers mean queue pressure to 0.007, and raises delivery to 91.7%. SAC + Guard + Fusion Predictor performs similarly on the same core service metrics, reducing mean latency slightly further to 65.0 ms, lowering mean queue pressure to 0.006, and increasing delivery to 92.0%. The most visible additional contribution of prediction in this regime is not a dramatic further reduction in latency, but better hotspot-aligned redistribution. Figure 5.24 shows that the predictive variant produces a much larger number of scale / redistribution events than the guard-only controller, indicating that prediction mainly improves spatial alignment and proactive hotspot following. This supports a different interpretation from CPU and queue stress: under abrupt flash overload, the guard provides the dominant immediate safety benefit, while prediction mainly improves hotspot tracking and control smoothness. Latency is reported in milliseconds (ms), queue pressure is normalized in [0,1], delivery ratio is reported as a percentage, and scale or redistribution events are counted as controller actions.

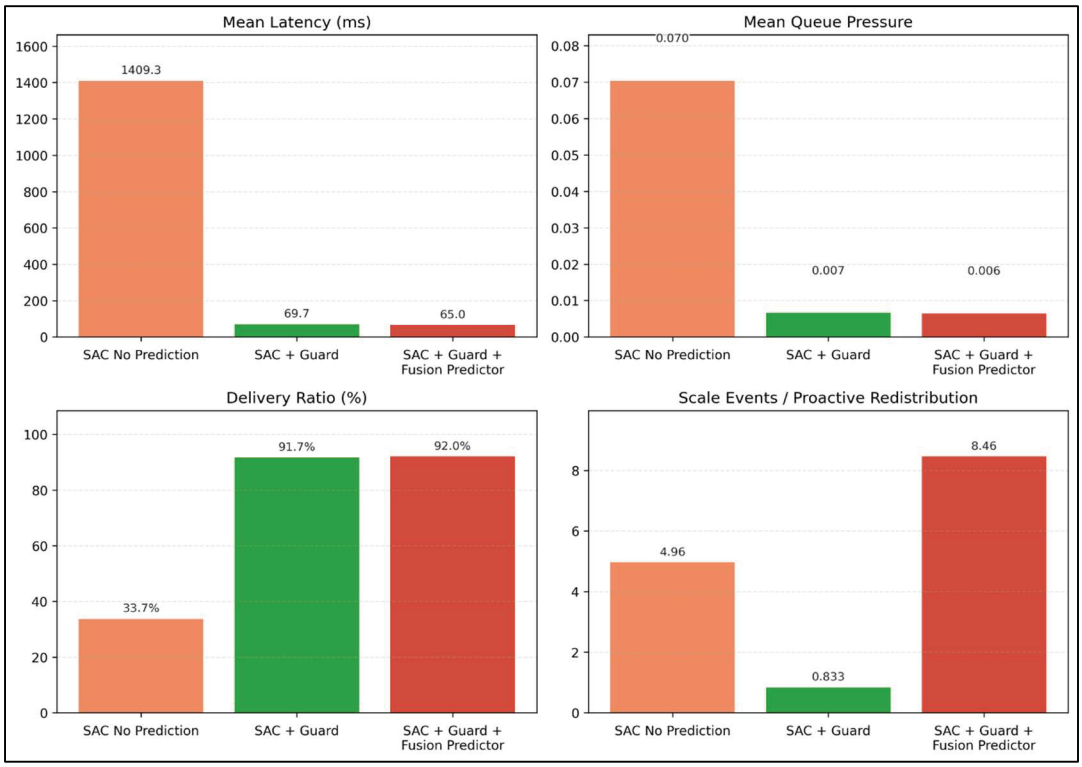


Figure 5.24 SAC-family ablation under hotspot (flash-crowd) stress

## 5.13 Continuous-Learning Diagnostics

### 5.13.1 CH-SAC Continuous Learning Timeline

Figure 5.25 illustrates the temporal behaviour of the CH-SAC continuous-learning pipeline during online execution. The aligned panels show experience accumulation, replay-buffer occupancy, and checkpoint events against wall-clock time. Together, these traces verify that the learning system remains active throughout the run and that model updates proceed concurrently with environment interaction.

The experience-accumulation curve increases steadily over time, indicating that the controller continuously generates new state–action–reward transitions while interacting with the fog environment. The replay-buffer trace mirrors this growth, showing that newly collected transitions are inserted promptly and that storage remains synchronised with environment

interaction. The checkpoint staircase confirms that model snapshots are saved periodically without interrupting learning or simulation progress.

These diagnostics show that the controller does not rely on disjoint "collect-then-train" phases. Instead, it continues learning while the simulation is running, which is exactly the always-learning behaviour required for non-stationary mobility-driven workloads.

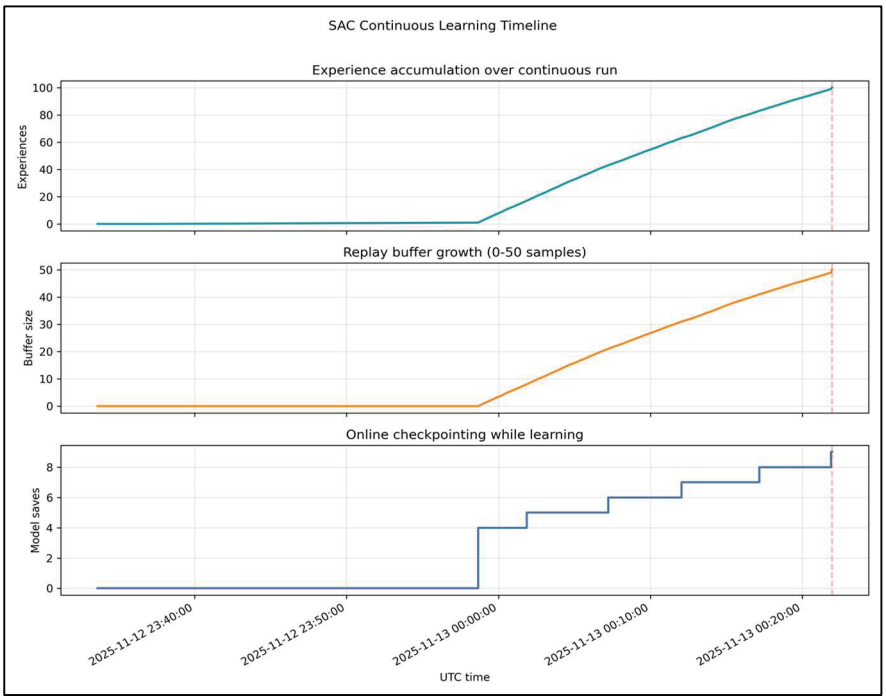


Figure 5.25 CH-SAC continuous learning timeline

### 5.13.2 Continuous Experience Collection in CH-SAC

Figure 5.26 provides the corresponding view of continuous experience collection in CH-SAC. The experience-generation curve and the replay-buffer curve evolve in near lockstep, indicating that each generated transition is promptly inserted into the replay buffer and made available for learning. Periodic buffer-refresh or sampling cycles do not stall interaction; instead, the buffer refills rapidly, showing that simulation and learning remain synchronised.

This behaviour validates several important operational properties. First, online adaptation is feasible: the controller continues learning from fresh experience rather than relying solely on

a static pre-trained policy. Second, the replay mechanism remains stable, with no sign of starvation or desynchronisation between environment interaction and learner updates. Third, checkpointing and persistence proceed in parallel with data collection, which supports reproducibility and long-run stability.

These observations strengthen the methodological contribution of the thesis by showing that CH-SAC with Fusion Predictor can function as a stable, online, continuously adapting controller within an event-driven fog-simulation environment.

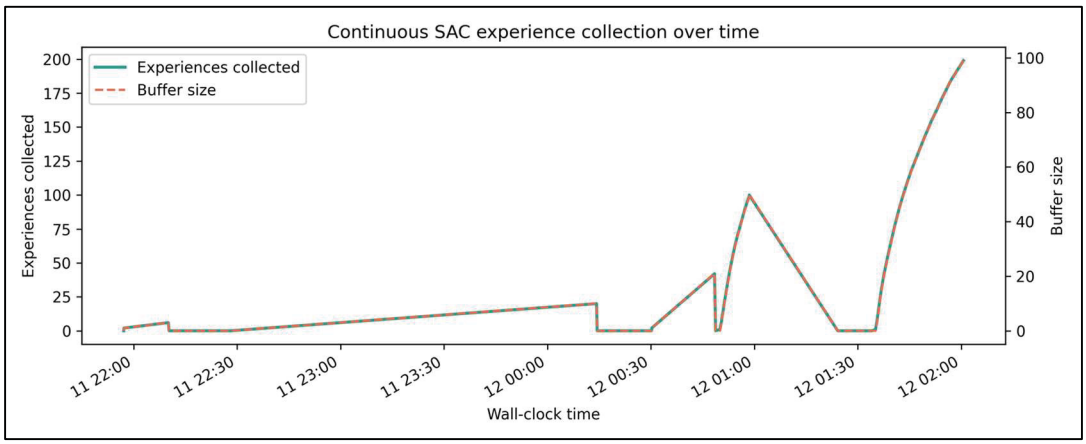


Figure 5.26 Continuous SAC experience collection over time

## 5.14 Synthesis of Behavioural Insights

This chapter shows that the proposed framework differs from reactive controllers not only in performance level, but in behaviour. The combination of the Fusion Predictor, CH-SAC, and the guard shifts fog-resource management from reacting to visible congestion toward preventing congestion before it becomes dominant.

Across all stress regimes, the same control hierarchy appears. Prediction provides foresight about where and when demand will rise. CH-SAC translates that information into adaptive scaling. The guard preserves stability when overload develops too quickly for learned control alone. These components are complementary rather than redundant.

Their relative importance depends on the stress regime. Under CPU stress and especially queue stress, prediction is decisive because early scaling prevents queue growth before it becomes

self-reinforcing. Under flash-crowd stress, the guard becomes the main safety mechanism because overload appears abruptly, while prediction mainly improves spatial adaptation. The focus-node analysis adds a second key insight: effective control requires knowing not only that demand will rise, but also where it will rise. By tracking a dynamic set of high-pressure nodes, the framework makes proactive control feasible under non-stationary mobility-driven demand. The ablation results confirm that the full framework is stronger than any individual component. Baseline SAC remains largely reactive and can collapse under sustained overload. Guard-only SAC improves robustness but still acts after congestion appears. Predictive CH-SAC with guard is the only configuration that consistently combines safety with anticipatory control. This is the main behavioural conclusion of the chapter.

# CHAPTER 6

# EXPERIMENTAL EVALUATION AND RESULTS

## 6.1 CPU-Stress Evaluation.

### 6.1.1 Experimental Objective

This scenario evaluates the proposed controller under a severe compute bottleneck, where insufficient processing capacity can quickly produce queue buildup, tail-latency spikes, and request loss. Fog nodes are restricted to one worker with 30 ms service time per request, queues are bounded to 120, and the workload is intensified through accelerated replay of trace-driven taxi demand. All policies are tested under the same topology, placement, workload, and random seed so that performance differences reflect control quality rather than experimental conditions.

Performance is assessed using mean latency, P95 latency, mean and P95 queue pressure, and sink throughput. Figures 6.1–6.5 report the core comparison together with the temporal behaviour behind the results. Unless otherwise stated, Chapter 5 reports behavioural evidence from the trace-driven YAFS layer, whereas Chapter 6 reports quantitative stress results from the Mininet emulation layer.

### 6.1.2 Results Under CPU Stress

Figure 6.1 shows that CH_SAC + Fusion Predictor performs best under compute stress. It achieves the lowest latency and queue pressure, while also delivering the highest throughput. In the core comparison, it reduces mean latency from 4812.1 ms under SAC (no prediction) to 212.6 ms, lowers mean queue pressure from 0.1716 to 0.0173, and increases sink-received traffic from 37,070 to 172,880. The Threshold policy performs better than non-predictive SAC

but remains clearly worse than the proposed method, with 702.0 ms mean latency, 0.0544 mean queue pressure, and 145,955 successfully received messages.

The extended comparison in Figure 6.1 shows the same ranking across additional baselines. Threshold, GA, and PSO mitigate congestion to some extent, but they remain well behind the proposed controller. Static control, baseline RL, and especially SAC without prediction perform poorly, indicating that reactive policies cannot keep pace once compute capacity becomes the limiting resource. The key result is that reinforcement learning alone is not sufficient here; the gain comes from combining learning with short-horizon prediction.

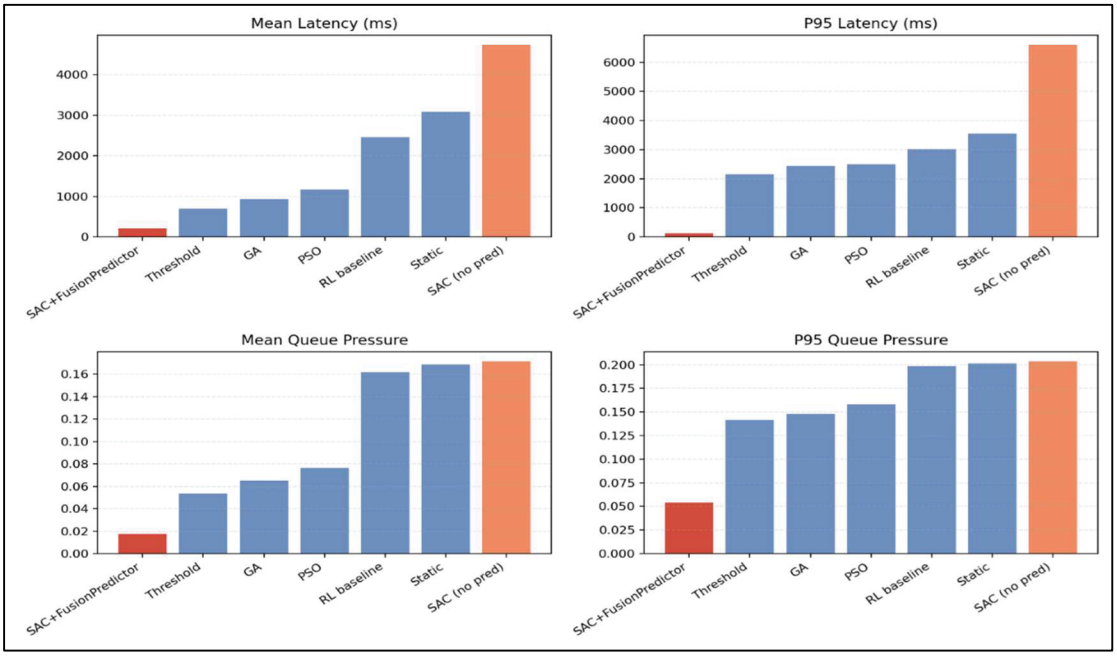


Figure 6.1 Performance comparison of all algorithms under CPU stress

Figure 6.2 reinforces this conclusion through throughput. CH-SAC + Fusion Predictor achieves the highest sink-received traffic, showing that its lower latency is not obtained by discarding workload. Instead, it processes more requests successfully under the same compute constraints. By contrast, the poor throughput of SAC (no prediction) confirms that delayed scaling leads to sustained congestion and large request loss.

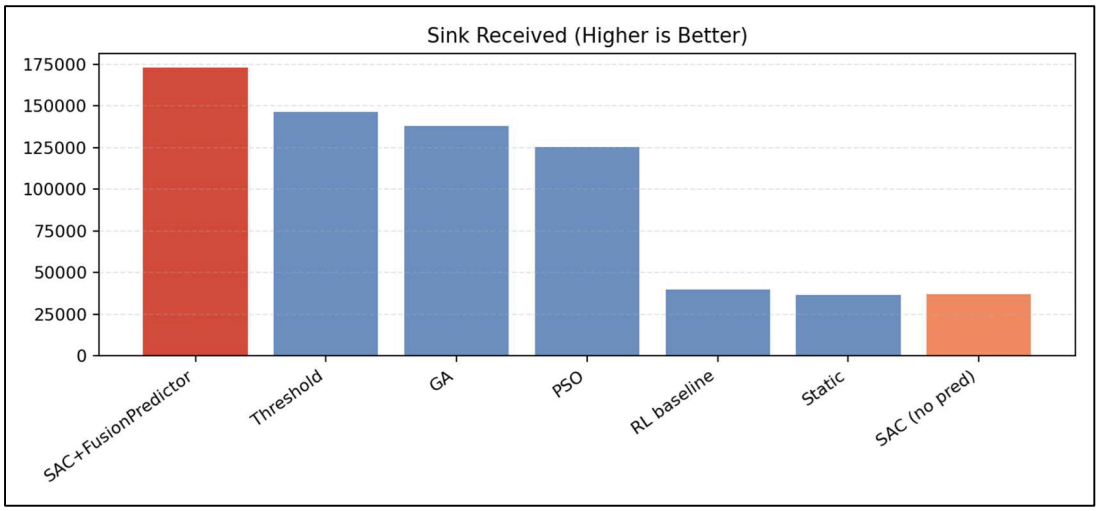


Figure 6.2 Sink throughput under CPU stress

### 6.1.3 Temporal Behaviour under CPU Stress

The time-series results explain why the aggregate gains occur. In Figure 6.3, the proposed controller shows only a brief early rise in queue pressure, after which queue pressure remains close to zero for most of the run. Latency follows the same pattern, stabilising at a low level after the initial transient. In contrast, SAC (no prediction) enters a persistent congestion regime: queue pressure remains high and latency continues to rise throughout the simulation. This shows that the proposed controller does not merely recover faster; it prevents prolonged congestion from forming.

Figure 6.3 compares the temporal behaviour of latency and queue pressure under CPU stress for SAC with and without the Fusion Predictor. With prediction, both metrics remain bounded and stable after the initial transient, indicating effective proactive scaling before congestion develops. Without prediction, the controller reacts later, resulting in larger queue accumulation, higher latency fluctuations, and slower recovery. This confirms that predictive guidance improves stability and congestion avoidance under compute bottlenecks.

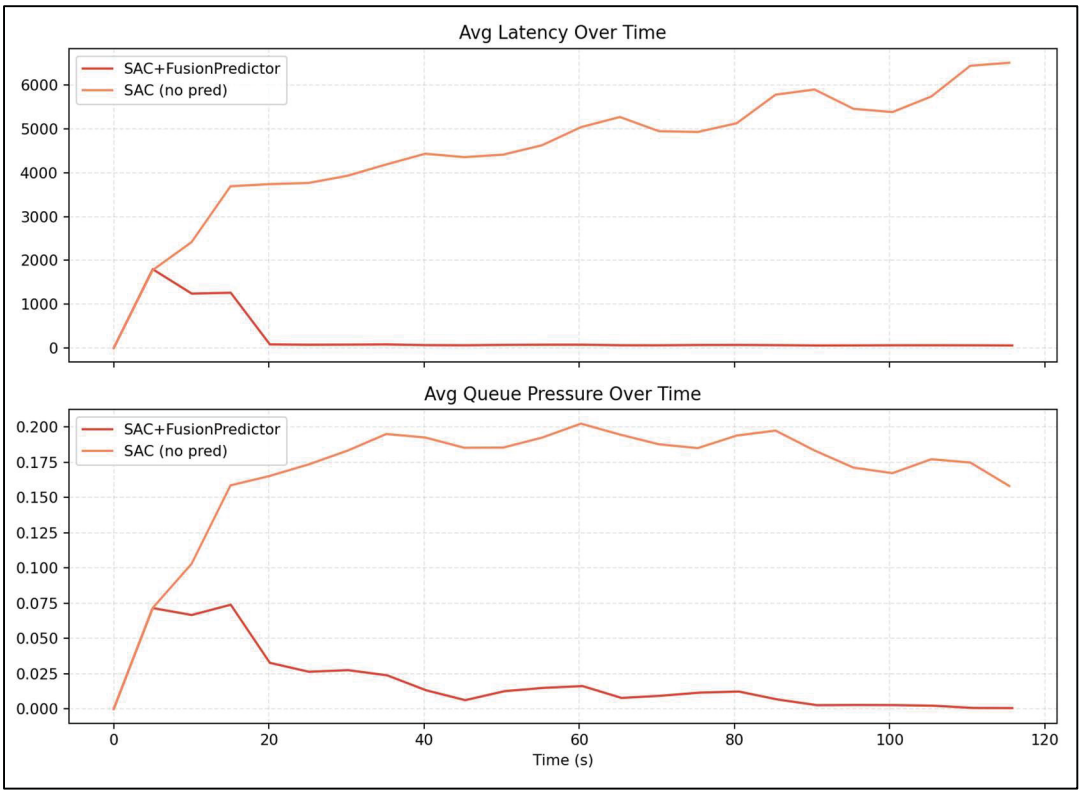


Figure 6.3 Latency and queue pressure over time under CPU stress

Figure 6.4 complements the previous time-series view by adding the predictor signal. Peaks in predicted arrivals appear before the largest divergence between the predictive and non-predictive controllers becomes visible in queue pressure and latency. The figure therefore supports the main causal interpretation of this section: the controller receives useful early warning of rising load and remains in a low-congestion regime, whereas SAC without prediction does not. What the plot shows most clearly is the timing of the forecast signal relative to subsequent congestion behaviour, rather than the exact magnitude of each individual actuation.

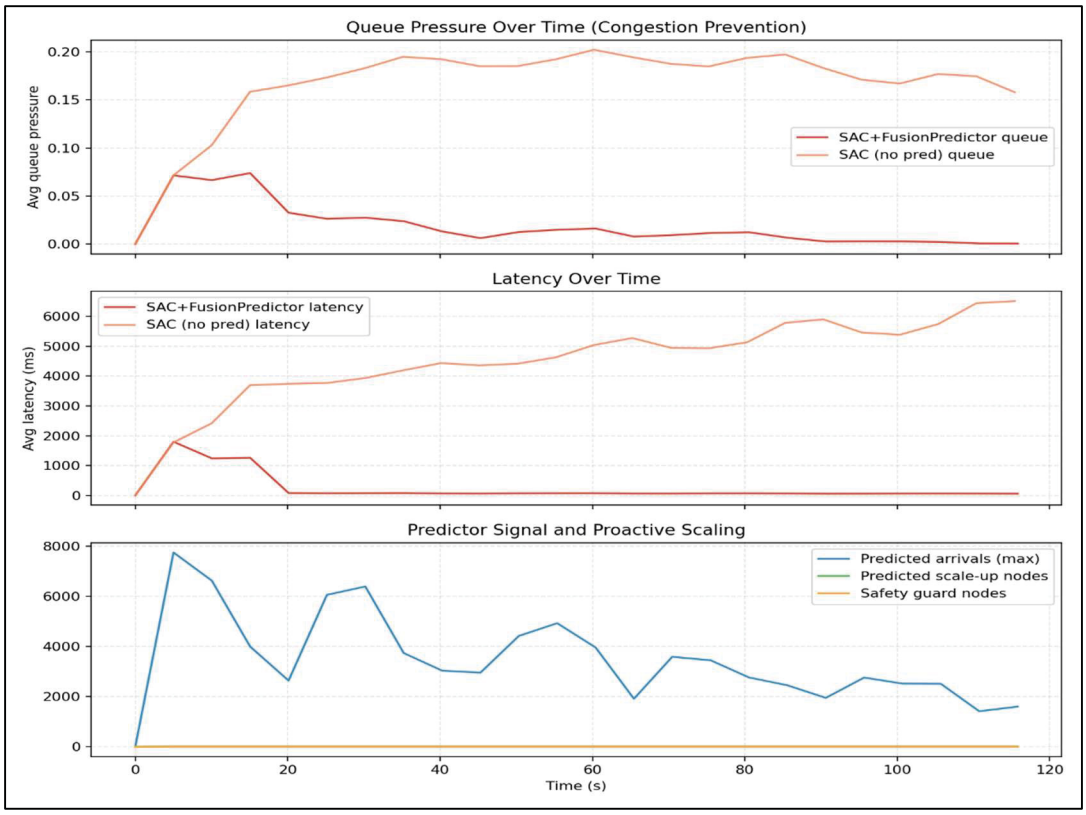


Figure 6.4 Dynamic behaviour under CPU stress

### 6.1.4 Role of the Fusion Predictor

The contribution of the Fusion Predictor is illustrated in Figure 6.5, which compares predicted and actual demand for both focus-average and focus-maximum workload. The predictor tracks the timing and overall shape of demand well, with correlation = 0.84 and MAPE = 0.19. The main limitation is that peak magnitudes are visibly damped, especially for focus-maximum demand, so the predictor should be interpreted as an early-warning model rather than an exact peak-reconstruction model. For control, this is sufficient: the controller mainly needs timely indication that load is rising, not perfect reproduction of every peak amplitude.

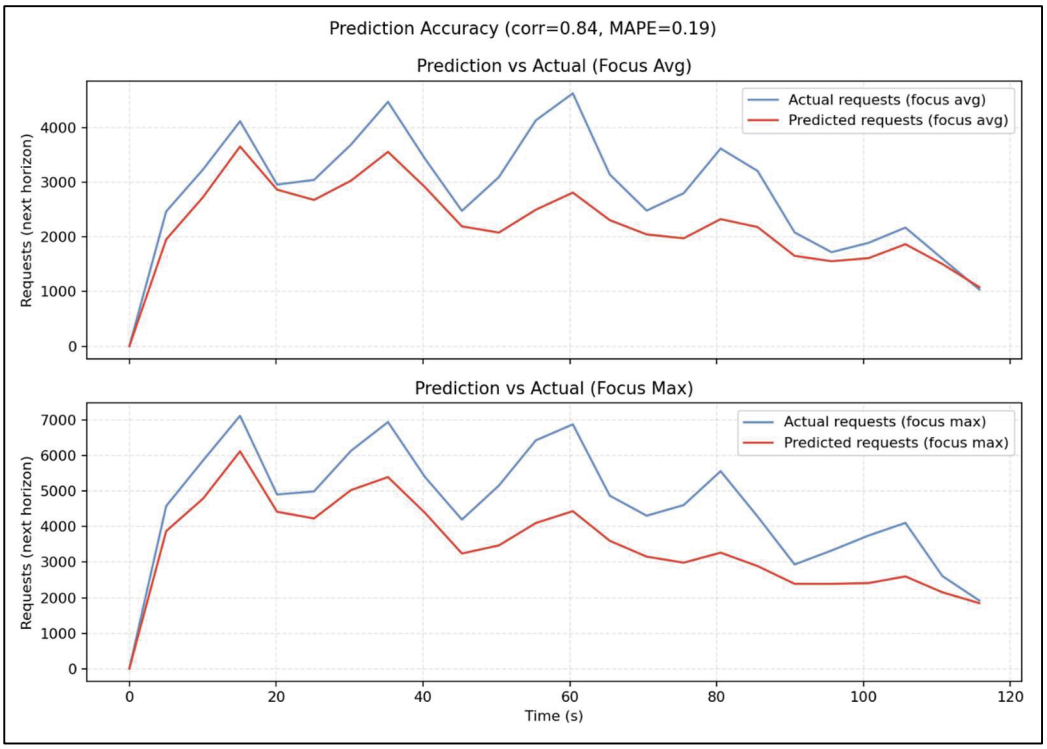


Figure 6.5 Predicted vs actual workload under CPU stress

### 6.1.5 Reliability, Stability, and Recovery under CPU Stress

Figure 6.6 presents the main reliability and control-overhead comparison under CPU stress. The three panels report SLA Risk Score, End-to-End Drop Rate, and Scale Overhead Score. CH-SAC + Fusion Predictor is best on all three metrics, with approximately 0.037 SLA Risk, 3.7% Drop Rate, and 2.4 Scale Overhead. The strongest classical baseline on the service-facing panels is Threshold, at about 0.246 SLA Risk and 24.6% Drop Rate, but it remains far behind the proposed controller. Predictor-augmented GA and PSO improve over their non-predictive versions, which confirms that predictive information is valuable under compute saturation, yet their drop rates remain in the mid-20% to low-30% range and their effective control burden remains much higher than that of CH-SAC. The proposed controller protects service quality without relying on excessive or wasteful actuation. SLA Risk Score and Scale Overhead Score are dimensionless evaluation scores, while End-to-End Drop Rate is reported as a percentage.

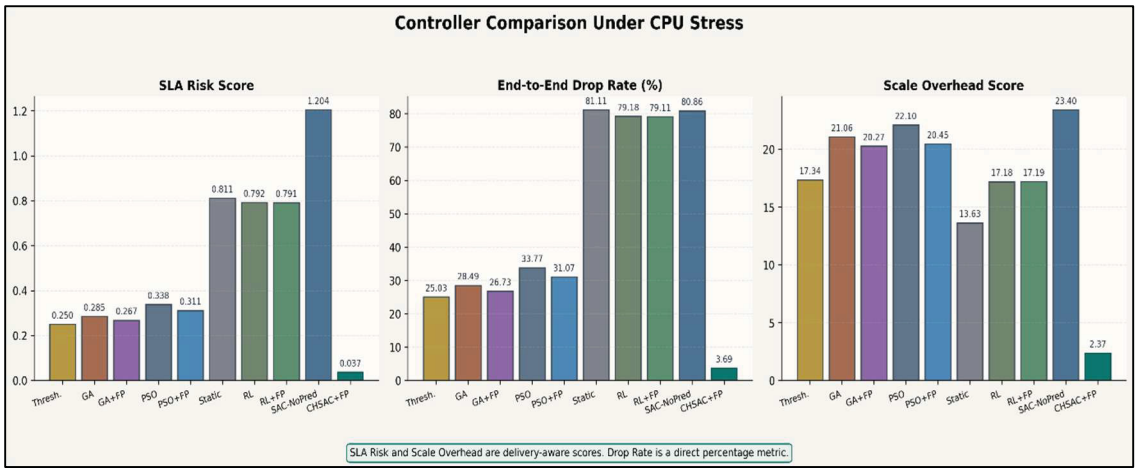


Figure 6.6 Reliability and control-overhead comparison under CPU stress

Figure 6.7 explains why this advantage is sustained over time. The stability comparison reports Latency Stability Score, Queue Stability Score, Delivery Ratio, and CPU Control Stability Score. CH-SAC + Fusion Predictor again ranks first on all four metrics, achieving approximately 322 on latency stability, 20.4 on queue stability, 96.31% delivery ratio, and 0.136 on CPU control stability. Threshold is the closest overall baseline, but it remains much worse on every panel, with approximately 1070, 71.5, 74.97%, and 0.414, respectively. These results show that CH-SAC does not appear stable merely because it adapts weakly. It is simultaneously smoother and more effective, preserving high delivery while keeping latency, queues, and control actions more orderly.


Figure 6.7 Stability, delivery, and controller smoothness under CPU stress

Figure 6.8 focuses on the harder part of CPU-stress behaviour: stressed-tail latency, queue recovery, and useful CPU efficiency. CH-SAC + Fusion Predictor remains best on all three metrics, with approximately 1157 ms P99 Controller Latency, 18.2 Queue Recovery Score, and 0.00565 Useful CPU Cost Score. Threshold is the strongest baseline on recovery and tail latency, while GA + Fusion Predictor is the strongest baseline on useful CPU cost, but no baseline is competitive across all three panels together. This matters because it shows that the proposed controller is not only good on average behaviour. It also controls the stressed tail, recovers fastest after backlog growth, and converts CPU effort into useful delivered work more efficiently than the alternatives.

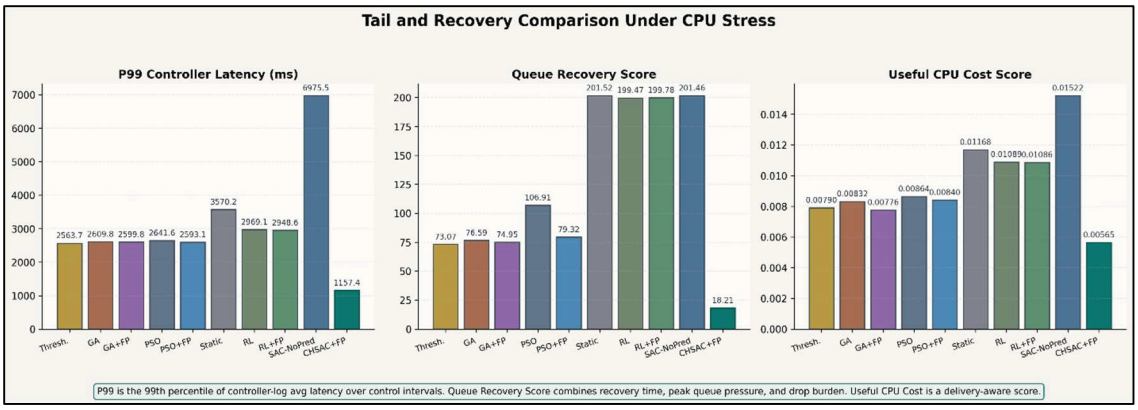


Figure 6.8 Tail, recovery, and CPU efficiency under CPU stress

Figure 6.9 reframes the CPU-stress comparison in terms of relative improvement over every baseline. Every cell in the heatmap is positive, meaning that CH-SAC + Fusion Predictor improves over every baseline on every plotted metric. Relative to Threshold, the gains are about 85.3% in SLA Risk and Drop Rate, 86.3% in Scale Overhead, 69.9% in Latency Stability, 71.5% in Queue Stability, 28.5% in Delivery Ratio, 67.0% in CPU Stability, 54.9% in P99 Controller Latency, 75.1% in Recovery, and 28.5% in Useful CPU Cost. The same pattern remains visible against predictor-enabled baselines such as GA + Fusion Predictor and PSO + Fusion Predictor. This confirms that the advantage of CH-SAC is broad rather than local, and that it cannot be explained by access to prediction alone. All heatmap values are relative improvements expressed as percentages (%), where positive values indicate better performance by CH-SAC + Fusion Predictor.

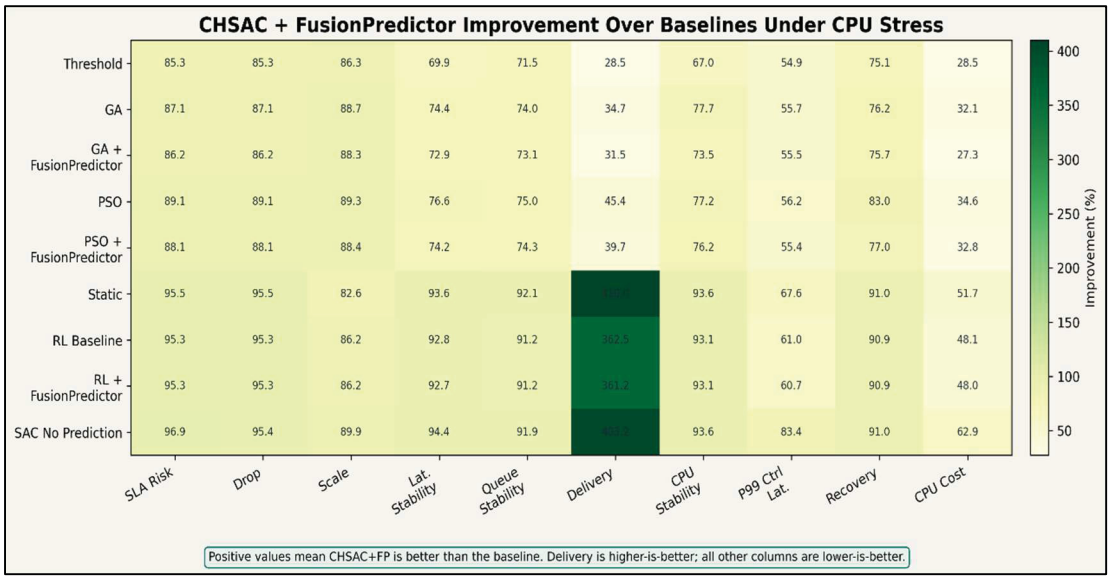


Figure 6.9 Relative improvement under CPU stress

Figure 6.10 provides the cleanest overall synthesis of the CPU-stress results. Instead of using a weighted aggregate score, it ranks each controller on the ten plotted metrics and counts the number of first-place finishes. CH-SAC + Fusion Predictor ranks first on all ten metrics, giving it 10 metric wins and an average rank of 1.0. Threshold is the strongest overall baseline with an average rank of 2.4, followed by GA + Fusion Predictor at 3.3. This ranking is important because it shows that the proposed controller is not dominating through a carefully selected subset of metrics. It is the best controller across the full CPU-stress metric set.

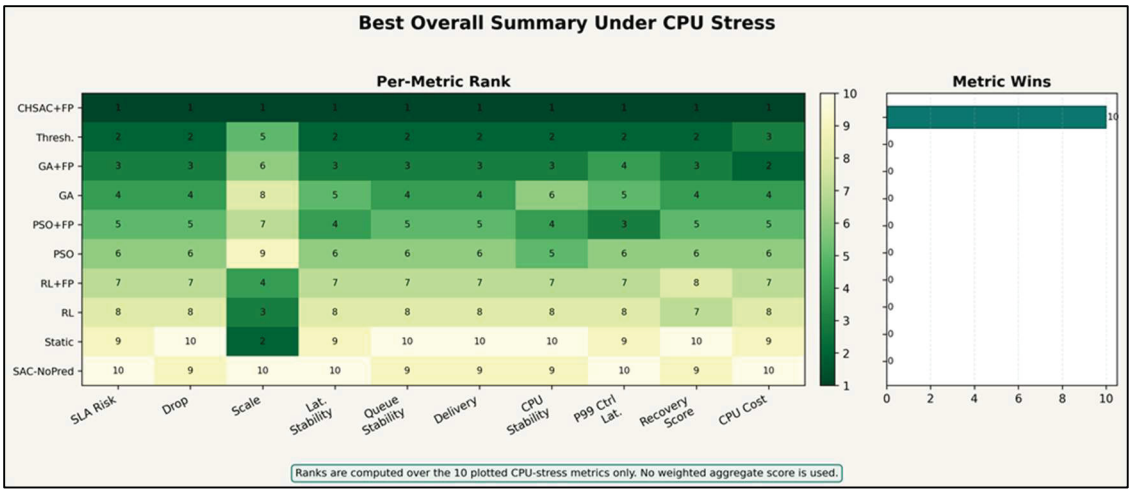


Figure 6.10 Best overall summary under CPU stress

Figure 6.11 confirms that this advantage is robust across repeated runs. In the multi-seed comparison, each bar shows the mean across seeds and the error bars show the standard deviation. CH-SAC + Fusion Predictor remains best on all three plotted metrics, with 0.0369 ± 0.0001 SLA Risk, 3.69 ± 0.01% Drop Rate, and 2.41 ± 0.06 Scale Overhead. The closest baseline, Threshold, remains far worse at 0.2458 ± 0.0064, 24.58 ± 0.64%, and 17.05 ± 0.42, respectively. The very small error bars for CH-SAC show that its advantage is not only strong but also repeatable.

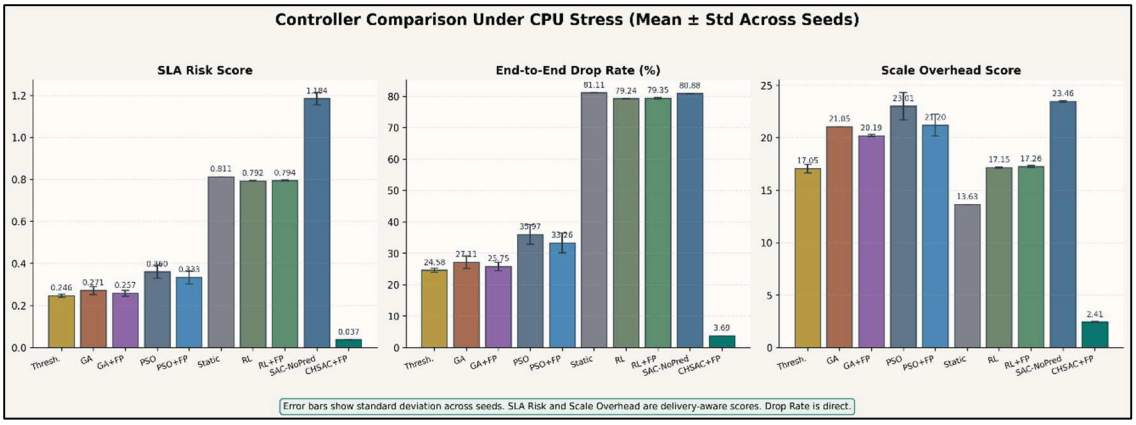


Figure 6.11 Multi-seed reliability comparison under CPU stress

Figures 6.6–6.11 support a clear conclusion. Under CPU stress, Threshold is the strongest classical baseline and GA + Fusion Predictor is the strongest predictor-enabled heuristic baseline, yet both remain clearly behind CH-SAC + Fusion Predictor. Prediction helps the baseline families, but the decisive improvement comes from combining short-horizon predictive guidance with constraint-aware continuous control.

## 6.2 Flash-Crowd and Hotspot-Shift Stress

This scenario evaluates controller behaviour under rapidly shifting, spatially concentrated demand. Unlike CPU stress, where overload develops mainly through compute saturation, flash-crowd stress is dominated by sudden hotspot formation and migration across nodes. In this setting, reactive policies are disadvantaged because by the time congestion is visible, the hotspot may already have moved.

### 6.2.1 Hotspot Dynamics

Figure 6.12 shows that the workload is concentrated but not fixed. Across 15 windows, the identity of the top node changes 10 times, with 6 unique nodes appearing as the dominant hotspot. The average top-node share is only 0.16, while the average top-$K$ share is 0.57. This means that demand is repeatedly concentrated within a small subset of nodes, but that subset evolves over time.

These dynamics justify the use of rotating focus-node prioritisation. A static hotspot strategy would be inefficient because no single node remains dominant throughout the run. Effective control therefore requires both concentration on the busiest nodes and adaptation to hotspot migration.

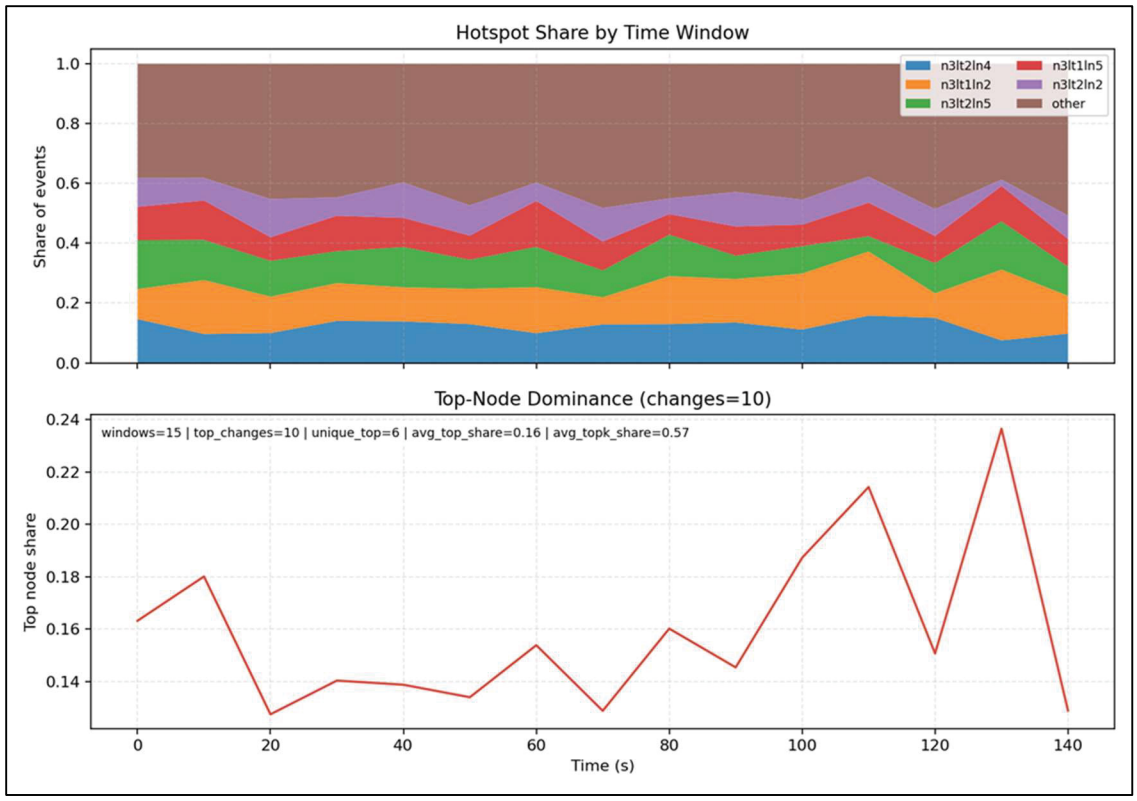


Figure 6.12 Hotspot dynamics

Figure 6.13 compares the main QoS metrics under flash-crowd and hotspot-shift stress. SAC + Fusion Predictor achieves the best performance on all four metrics: 41.65 ms mean latency, 48.50 ms P95 latency, 0.00194 mean queue pressure, and 0.00593 P95 queue pressure. The best heuristic baseline, Threshold, is markedly worse, with 309.21 ms mean latency and 0.02291 mean queue pressure, while SAC without prediction performs worst, with 1347.13 ms mean latency and 2045.51 ms P95 latency.

These results show that predictions remain decisive under spatially shifting demand. The controller reduces average latency by more than 32× relative to non-predictive SAC and cuts P95 latency by about 95.5% relative to the best baseline. Queue-pressure reductions show that these gains come from preventing congestion, not merely reacting after it has formed. By anticipating where demand will surge next, the controller reallocates capacity early enough to keep queues short and service stable even during abrupt hotspot transitions.

Overall, the flash-crowd results confirm the same core conclusion as the CPU-stress

evaluation: the benefit of the proposed framework comes from changing the timing and placement of control. Instead of responding after congestion becomes visible, it follows short-horizon demand shifts and acts before overload becomes dominant.

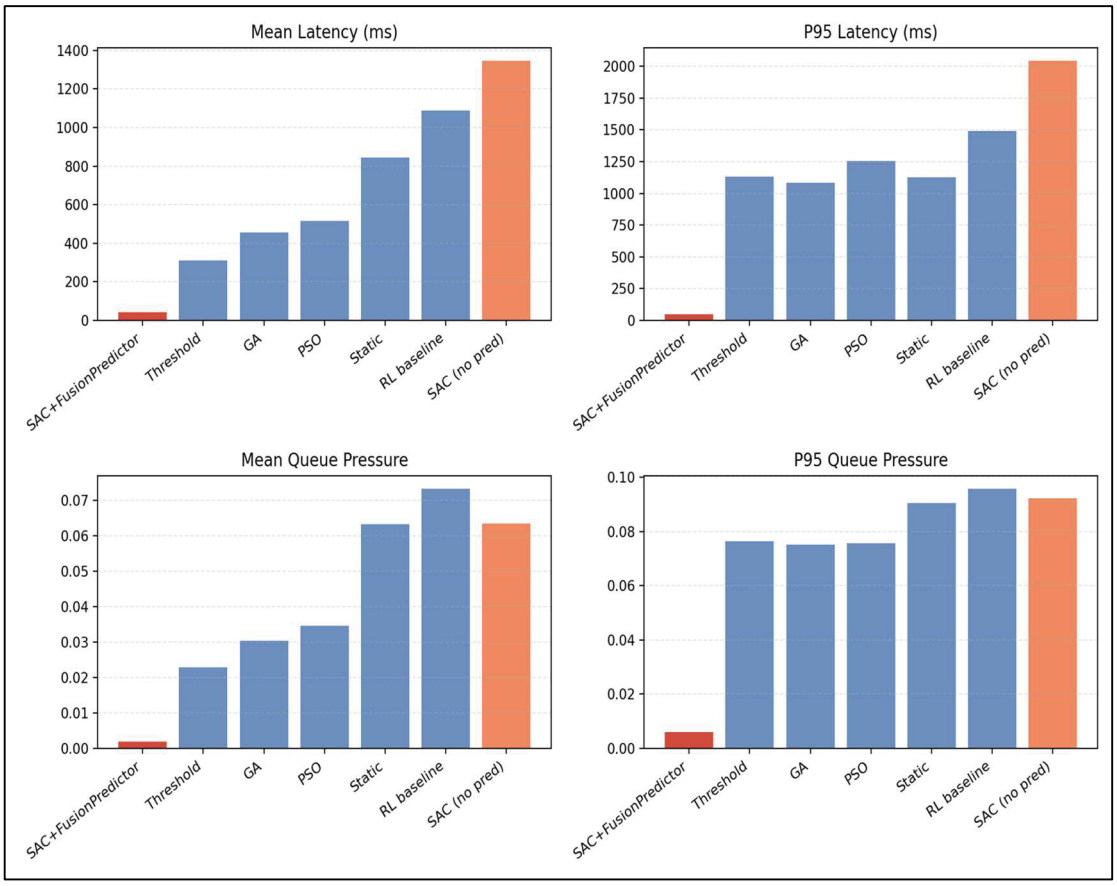


Figure 6.13 QoS under flash-crowd & hotspot shift

### 6.2.2 Temporal Dynamics under Flash-Crowd Stress

Figure 6.14 shows the time evolution of latency and queue pressure. Under SAC + Fusion Predictor, latency rises briefly to 191.00 ms and then stabilises, ending at 24.56 ms. Queue pressure follows the same pattern, peaking at 0.0157 before falling to 0.0000. In contrast, SAC (no prediction) fails to recover: latency rises to 1630.59 ms and remains high at 1508.27 ms, while queue pressure peaks at 0.0906 and ends at 0.0323. This shows that predictive control

prevents persistent congestion, whereas reactive SAC remains trapped in an overloaded regime.

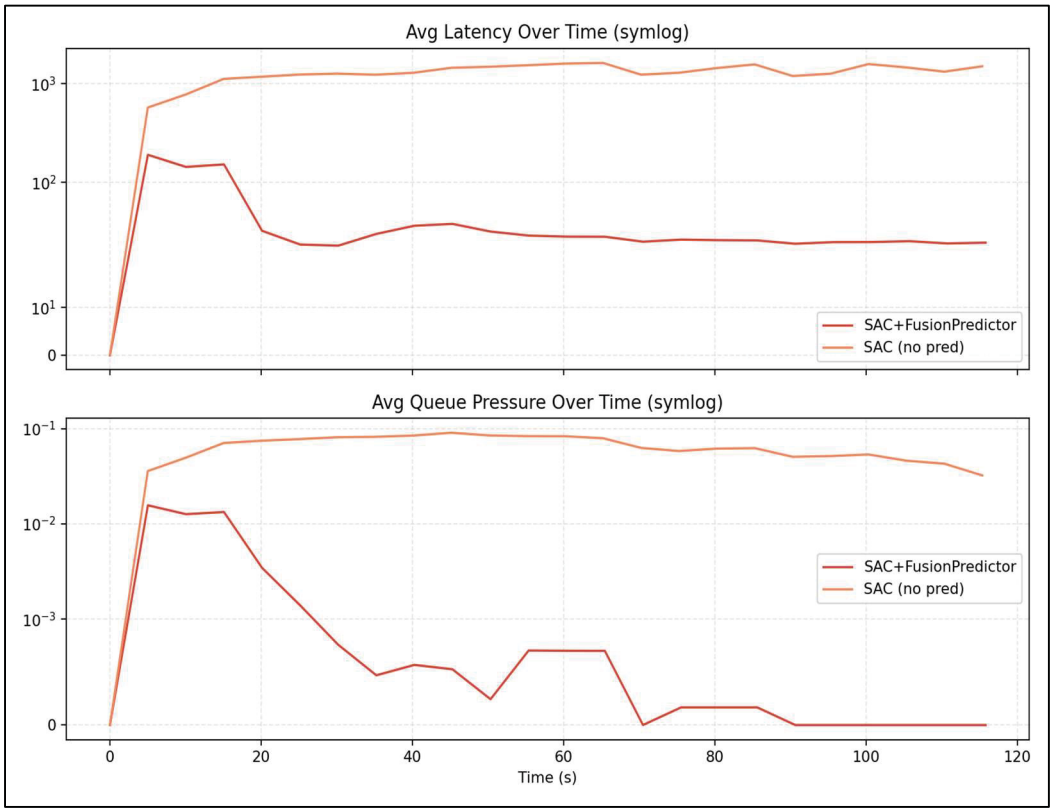


Figure 6.14 Latency and queue evolution under flash-crowd stress

Figure 6.15 shows that these gains are not achieved by suppressing workload. SAC + Fusion Predictor processes 162,285 requests, outperforming Threshold (126,773), GA (120,818), PSO (114,578), Static (67,067), Baseline SAC (69,374), and Baseline RL (50,209). Relative to the strongest baseline, this is a 28.01% throughput gain. The proposed controller therefore improves both responsiveness and completed work.

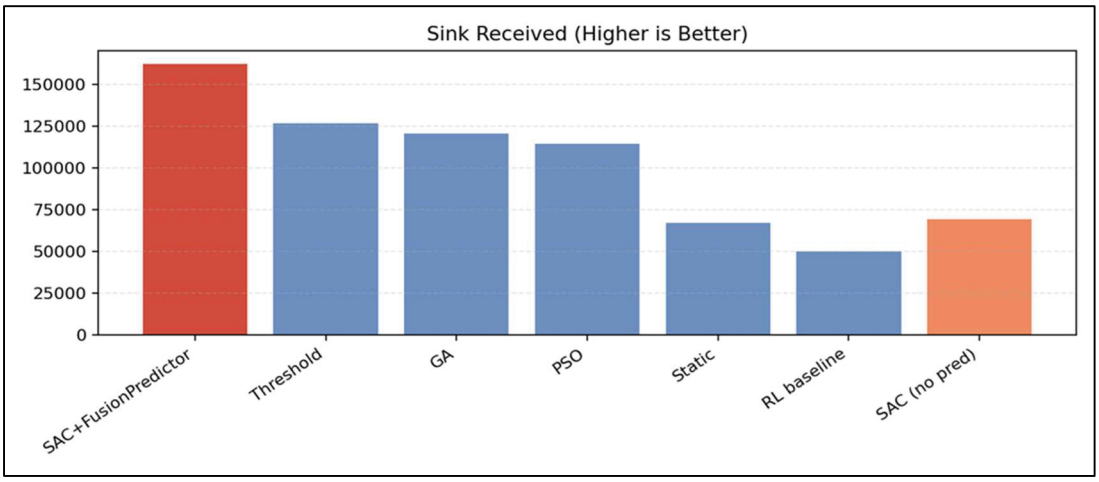


Figure 6.15 Total sink-received throughput under flash-crowd stress

### 6.2.3 Reliability, Resilience, and Control Stability

Figures 6.16–6.21 extend the flash-crowd and hotspot-shift evaluation beyond mean latency and throughput by examining reliability, service smoothness, stressed-tail behaviour, broad relative gain, overall ranking, and repeated-run robustness. In this non-stationary setting, the controller must react to both sudden load growth and shifting hotspot location. Across the full figure set, CH-SAC + Fusion Predictor remains the strongest controller.

Figure 6.16 presents the main reliability and adaptation comparison for the flash-crowd / hotspot-shift scenario. The three panels report SLA Risk Score, End-to-End Drop Rate, and Scale Overhead Score. CH-SAC + Fusion Predictor is best on all three metrics, with SLA Risk = 0.0360, Drop Rate = 3.60%, and Scale Overhead = 6.49. The closest baseline is Threshold, which achieves 0.2484 SLA Risk, 24.84% Drop Rate, and 9.74 Scale Overhead. This means that under abrupt flash arrivals and hotspot migration, CH-SAC reduces both service degradation and outright traffic loss very substantially, while still keeping the effective burden of adaptation low. Prediction helps some heuristic baselines, especially GA + Fusion Predictor and PSO + Fusion Predictor, but those improvements do not close the gap to CH-SAC.

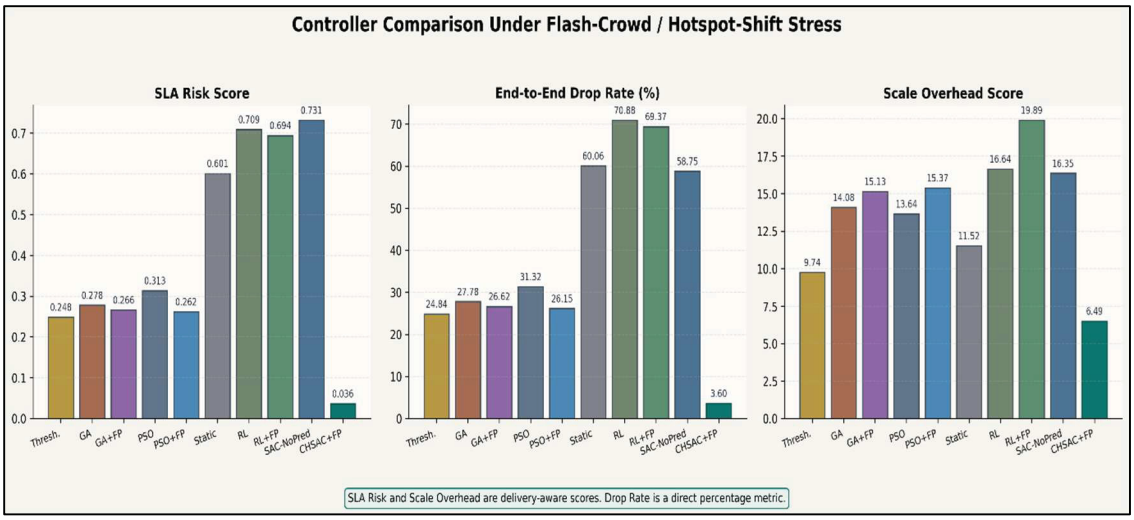


Figure 6.16 Reliability & adaptation under flash-crowd / hotspot-shift

Figure 6.17 shows why the proposed controller remains strong as the workload moves. The stability comparison reports Latency Stability Score, Queue Stability Score, Delivery Ratio, and CPU Control Stability Score. CH-SAC + Fusion Predictor again ranks first on all four metrics, with Latency Stability = 110.48, Queue Stability = 8.55, Delivery Ratio = 96.40%, and CPU Control Stability = 0.0433. Threshold is the strongest classical baseline, but it remains much worse at 769.4, 45.23, 75.16%, and 0.305, respectively. The strongest predictor-enabled baselines are split across panels, yet none of them is competitive across the full four-metric set. The result shows that CH-SAC does not achieve high delivery by becoming unstable, and it does not merely appear smooth because it serves less of the workload. It is the only controller that is simultaneously effective and orderly under shifting burst pressure.

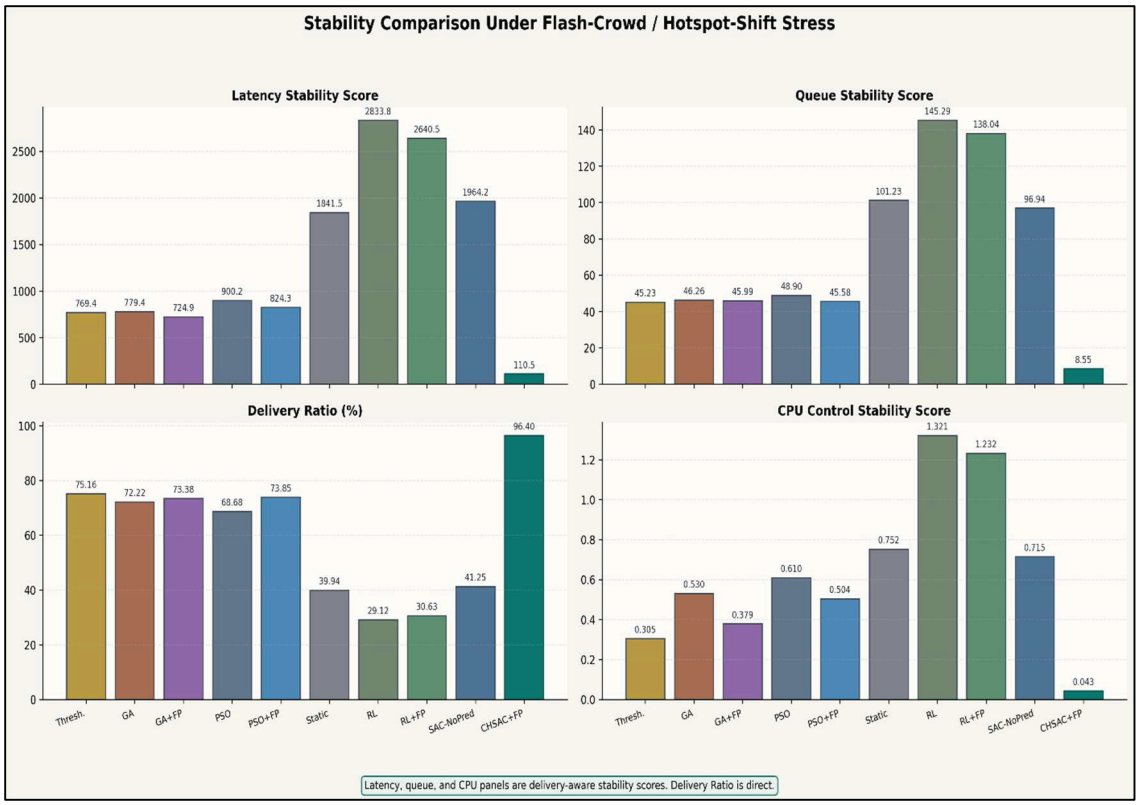


Figure 6.17 Stability and control smoothness under flash-crowd stress

Figure 6.18 focuses on the most difficult operating regime in the flash/hotspot scenario by comparing P99 Controller Latency, Queue Recovery Score, and Useful CPU Cost Score. CH-SAC + Fusion Predictor is again best on all three metrics, with P99 Controller Latency = 303.52 ms, Queue Recovery Score = 10.26, and Useful CPU Cost = 0.00375. Threshold remains the strongest overall baseline in this figure, with 1537.7 ms, 74.00, and 0.00475, but CH-SAC still outperforms it by a wide margin. Relative to Threshold, the proposed method improves P99 control latency by about 80.3%, recovery by about 86.1%, and useful CPU cost by about 21.2%. This figure is important because it shows that the proposed controller is not only strong on average metrics. It also controls the stressed tail, recovers much faster after burst buildup, and maintains the best useful efficiency once delivered work is taken into account. P99 Controller Latency is reported in milliseconds (ms), while Queue Recovery Score and Useful CPU Cost Score are dimensionless recovery and efficiency scores.

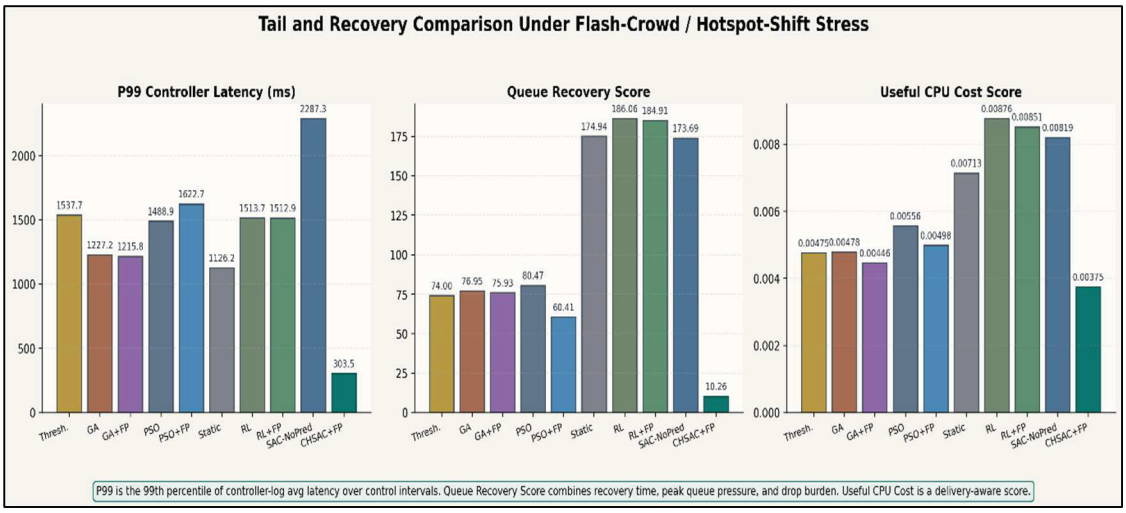


Figure 6.18 Tail, recovery & CPU efficiency under flash-crowd / hotspot-shift

Figure 6.19 reframes the flash/hotspot comparison as relative gain over every baseline. Every cell in the heatmap is positive, which means CH-SAC improves over every baseline on every plotted metric. Against Threshold, the improvements remain large across the full metric set, including roughly 85.5% in SLA Risk and Drop Rate, 85.6% in Latency Stability, 81.1% in Queue Stability, 85.8% in CPU Stability, 80.3% in P99 Controller Latency, 86.1% in Recovery, and 21.2% in Useful CPU Cost. The heatmap is even more revealing against weaker baselines, where gains in Delivery Ratio exceed 130% and rise above 230% in some comparisons. This confirms that the CH-SAC advantage is broad, large, and not limited to one isolated metric family.

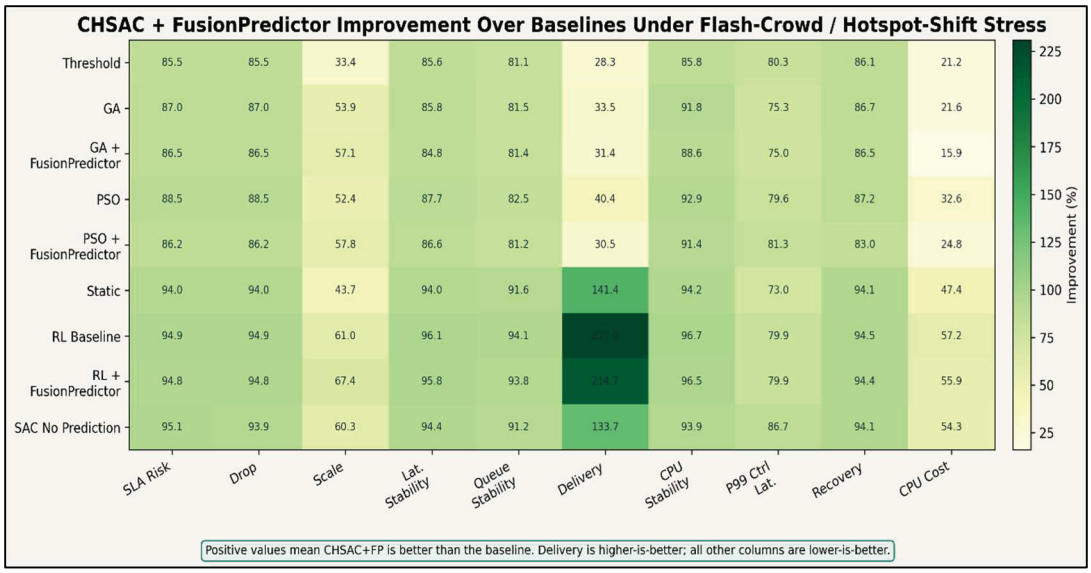


Figure 6.19 Relative improvement under flash-crowd stress

Figure 6.20 provides the cleanest overall synthesis of the flash/hotspot results. Each controller is ranked separately on the same ten plotted metrics and the number of metric wins is counted. CH-SAC + Fusion Predictor is rank 1 on all ten metrics, giving it 10 metric wins and an average rank of 1.0. Threshold is the strongest overall baseline with average rank = 2.9, followed by GA + Fusion Predictor at 3.6 and PSO + Fusion Predictor at 4.4. This ranking matters because it shows that prediction helps the heuristic baselines, but does not bring them close to the proposed controller. Only CH-SAC is consistently first across the complete flash/hotspot metric set.

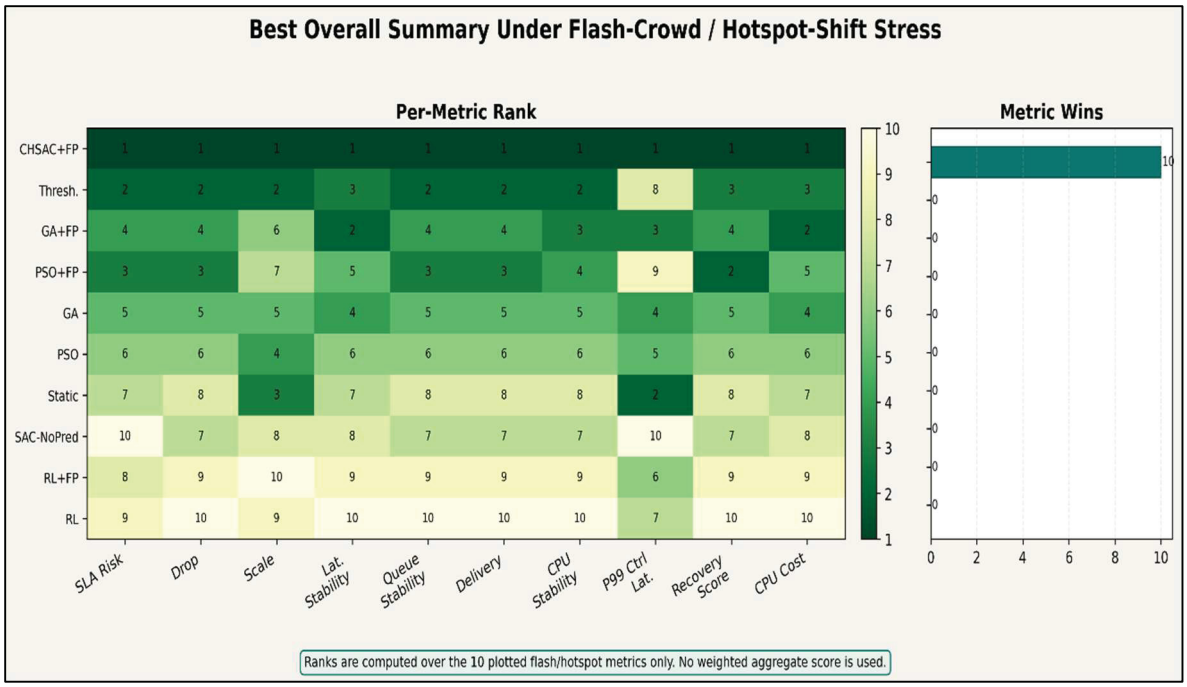


Figure 6.20 Overall performance across ten metrics under flash-crowd

Figure 6.21 confirms that the same pattern remains visible across repeated runs. In the multi-seed comparison, CH-SAC + Fusion Predictor is best on all three plotted metrics, with 0.0350 ± 0.0014 SLA Risk, 3.50 ± 0.14% Drop Rate, and 6.44 ± 0.07 Scale Overhead. The closest baseline is again Threshold, with 0.2438 ± 0.0065, 24.38 ± 0.65%, and 9.57 ± 0.24, respectively. The small error bars for CH-SAC show that its performance is not only strong, but also highly repeatable under bursty and shifting demand.

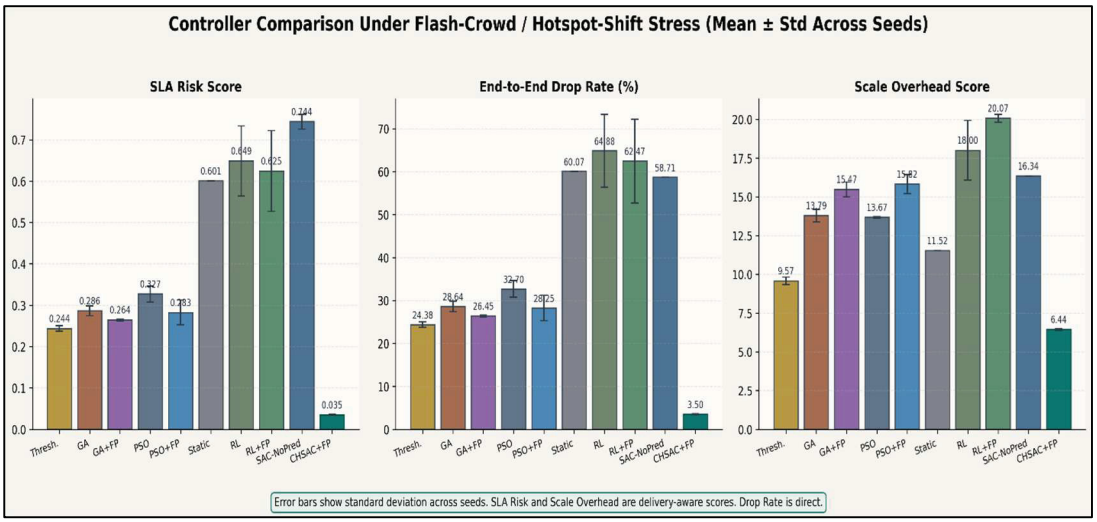


Figure 6.21 Multi-seed reliability comparison under flash-crowd stress

Figures 6.16–6.21 support a clear interpretation. Under flash-crowd / hotspot-shift stress, Threshold is the strongest classical baseline and GA + Fusion Predictor is the strongest predictor-enabled heuristic baseline, but both remain clearly behind CH-SAC + Fusion Predictor. The proposed method is best not only in reliability, but also in delivery, stability, recovery, efficiency, and repeated-run robustness.

### 6.2.4 Hotspot Dynamics and Demand Shifts

Figure 6.22 shows that the flash-crowd workload is both concentrated and shifting. Across 12 windows, the dominant node changes 6 times, with 3 unique nodes appearing as the top hotspot. The average top-node share is 0.27, while the average top-K share is 1.00, meaning that nearly all traffic remains concentrated within a very small hotspot set even though the leading node changes over time. This is important for control: the workload is not uniform, but neither is it fixed at one node. An effective controller must therefore focus on a small hotspot set while adapting as dominance shifts. This behaviour directly justifies prediction-guided focus rotation under flash-crowd stress.

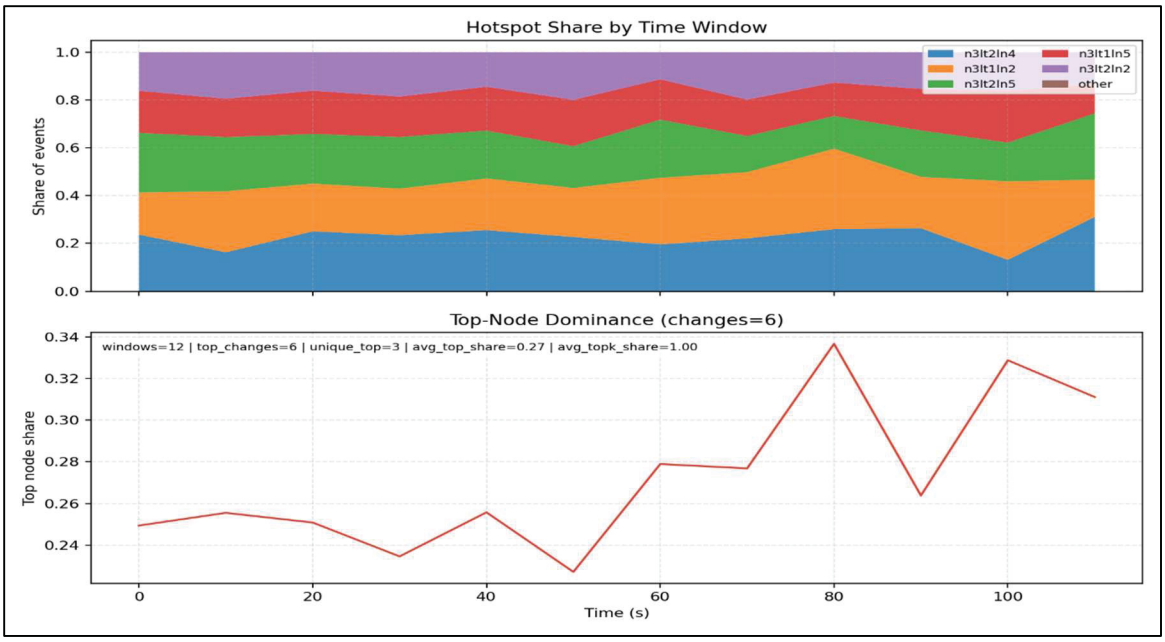


Figure 6.22 Flash-crowd hotspot dynamics and shifting dominance

### 6.2.5 Proactive Scaling and Congestion Prevention

Figure 6.23 provides the clearest behavioural explanation for the flash-crowd results. The predictor signal spikes early, before severe congestion becomes sustained in the non-predictive controller. At the same time, the predictive controller keeps queue pressure near zero and stabilises latency at a low level after the initial transient, whereas SAC without prediction remains in a congested regime with queue pressure around 0.05–0.09 and latency in the 1200–1600 ms range. The figure therefore supports an anticipatory-control interpretation: useful forecast information is available early enough for the controller to remain ahead of hotspot migration, even if the exact magnitude of each individual scaling action is not the main quantity shown by the plot.

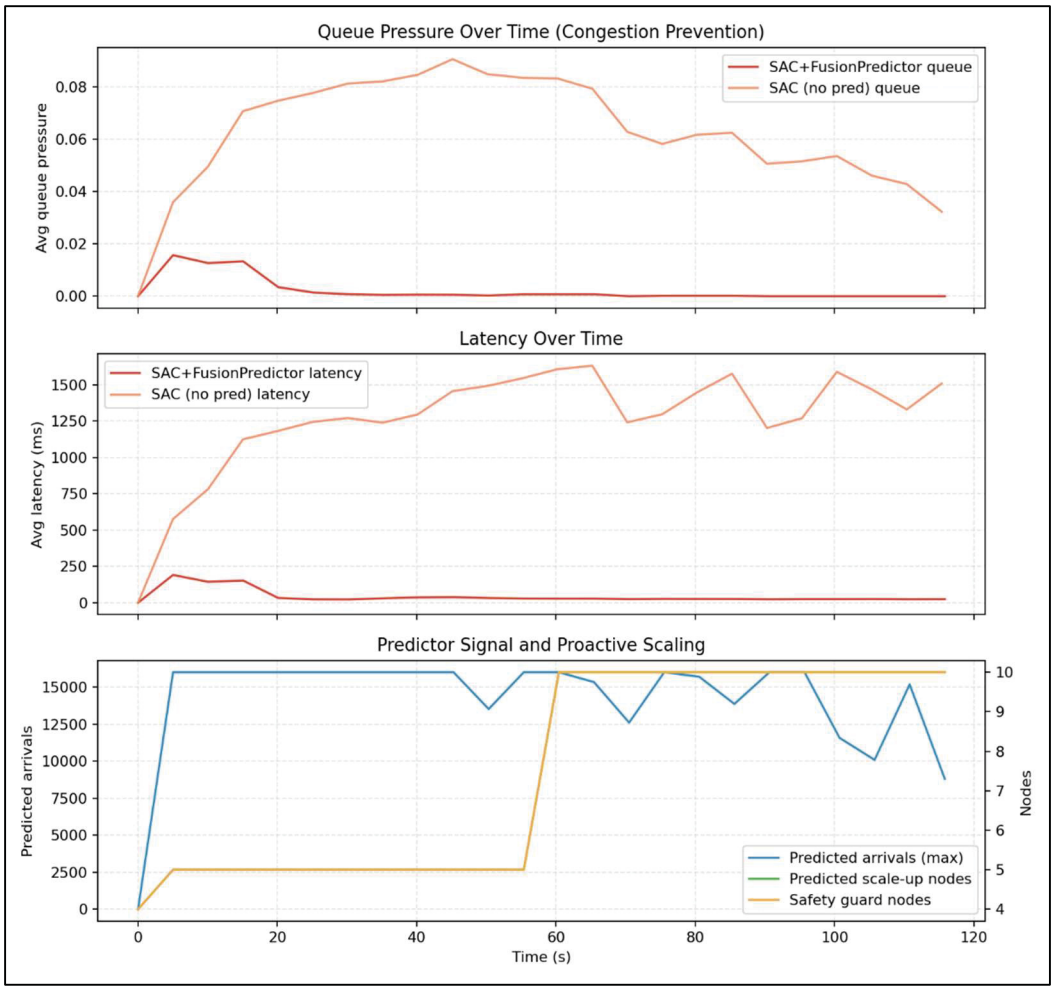


Figure 6.23 Flash-crowd prediction and congestion dynamics

## 6.3 Queue-Stress Evaluation

The queue-stress scenario tests whether the controller can prevent buffer buildup under sustained arrival pressure. Fog queues are restricted to 50, the workload is intensified with message-multiplier = 8, and each fog node is limited to one worker with 20 ms processing time. The experiment runs for 120 s under identical topology, workload, and random seed for all policies. Unlike CPU stress, this scenario is designed specifically to expose queue growth: once buffers begin to saturate, delayed control leads quickly to drops, high latency, and slow recovery.

### 6.3.1 Core QoS Performance

Figure 6.24 compares mean latency, P95 latency, mean queue pressure, and P95 queue pressure under queue stress. SAC + Fusion Predictor performs best on all four metrics, achieving 79.3 ms mean latency, 128.5 ms P95 latency, 0.0105 mean queue pressure, and 0.0418 P95 queue pressure. The strongest non-predictive baseline, Threshold, reaches 462.1 ms, 1311.9 ms, 0.0517, and 0.1397, while SAC (no prediction) and Baseline RL are substantially worse. These results show that prediction keeps the system out of the queue-growth regime rather than trying to drain queues after congestion has already formed.

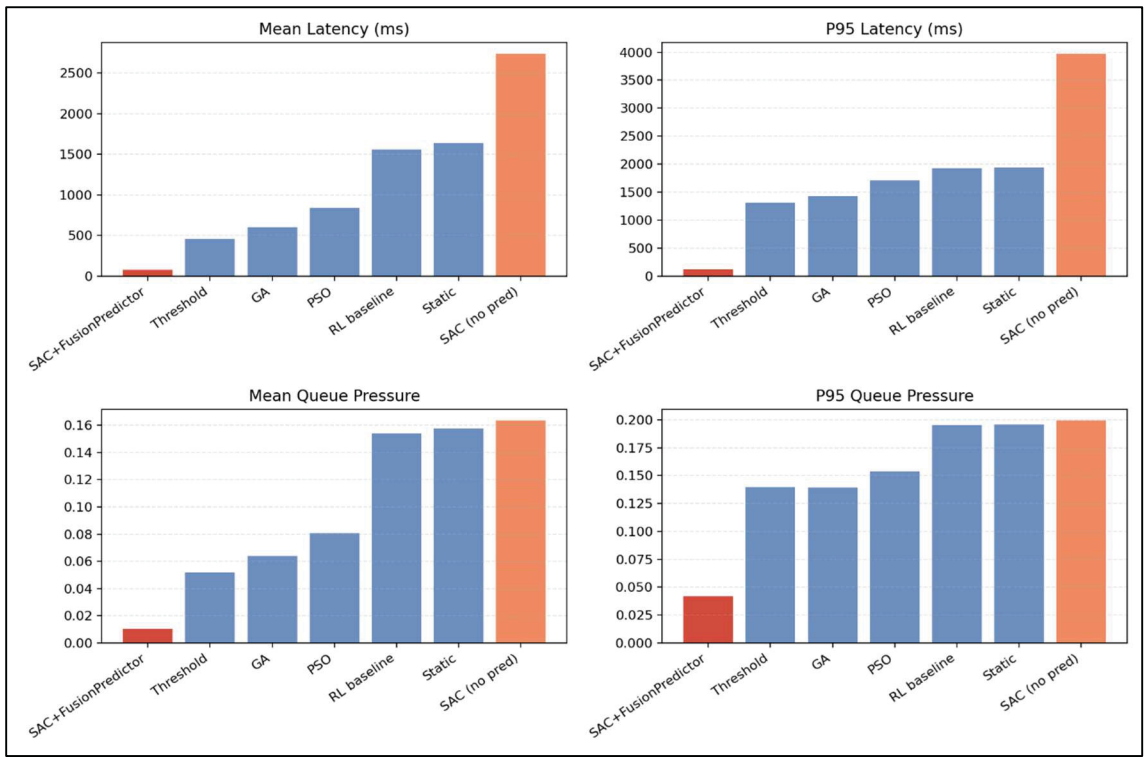


Figure 6.24 Core QoS performance under queue stress

### 6.3.2 Throughput and Work Conservation

Figure 6.25 shows that the latency gains are not achieved by sacrificing throughput. SAC + Fusion Predictor delivers 241,454 requests, compared with 192,729 for Threshold, 178,192 for GA, and 153,448 for PSO. The weaker learning baselines remain near the 50k–60k range.

Relative to the strongest baseline, the proposed controller improves throughput by about 25%. This confirms that predictive scaling improves both service quality and completed work.

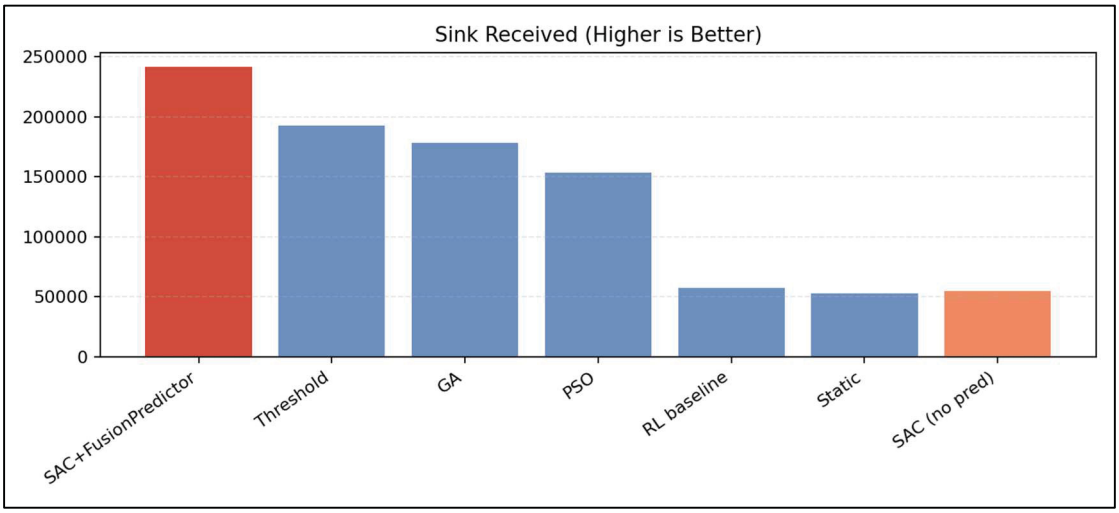


Figure 6.25 Throughput under queue stress

### 6.3.3 Reliability and Control Effort under Queue Stress

Figure 6.26 presents the main reliability and control-effort comparison under queue stress by reporting SLA Risk Score, End-to-End Drop Rate, and Scale Overhead Score. These metrics capture how well each controller prevents service degradation, how much traffic is actually lost, and how efficiently it converts adaptation into useful service under sustained backlog pressure.

CH-SAC + Fusion Predictor is best on all three panels, with SLA Risk = 0.0567, Drop Rate = 5.67%, and Scale Overhead = 3.28. The strongest baseline overall is Threshold, which records 0.2581 SLA Risk, 25.81% Drop Rate, and 16.09 Scale Overhead. Relative to Threshold, the proposed controller reduces SLA risk and drop rate by about 78%, while also reducing effective adaptation burden by nearly 80%. This gap is large enough to show a real change in control quality rather than a small calibration improvement.

The figure also shows that prediction helps the heuristic baselines, but only partially. GA + Fusion Predictor improves over plain GA, and PSO + Fusion Predictor improves over plain

PSO, yet both remain far behind CH-SAC in both reliability and effective control burden. Static offers an important counterexample: it achieves lower adaptation than several active baselines, but still performs very poorly on service outcomes. This confirms that under queue pressure, low actuation alone is not evidence of good control. A strong controller must both protect the workload and do so efficiently.

The main interpretation of Figure 6.26 is therefore straightforward. Under queue pressure, CH-SAC + Fusion Predictor loses much less traffic and reaches that result with much lower effective adaptation burden than every alternative. It is neither simply more aggressive nor simply more conservative. It is substantially better at turning control actions into useful service.

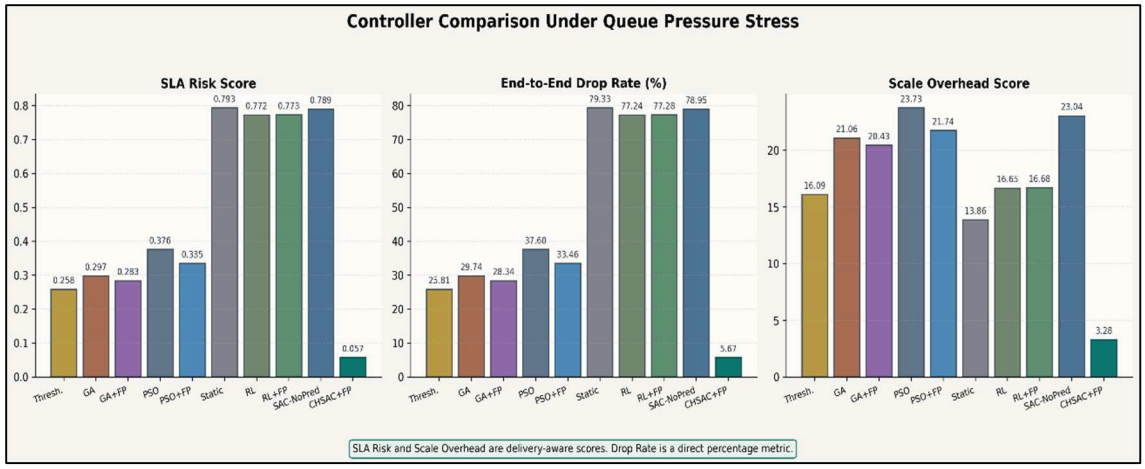


Figure 6.26 Reliability and control effort under queue stress

### 6.3.4 Prediction Accuracy and Temporal Dynamics

Figure 6.27 shows that the predictor tracks short-horizon demand well under queue stress, with correlation ≈ 0.83 and MAPE ≈ 0.20. The forecasts follow the timing and overall direction of demand changes closely, but peak amplitudes are visibly damped, especially for focus-maximum demand. The predictor should therefore be interpreted as a control-oriented early-warning signal rather than an exact peak-matching model.

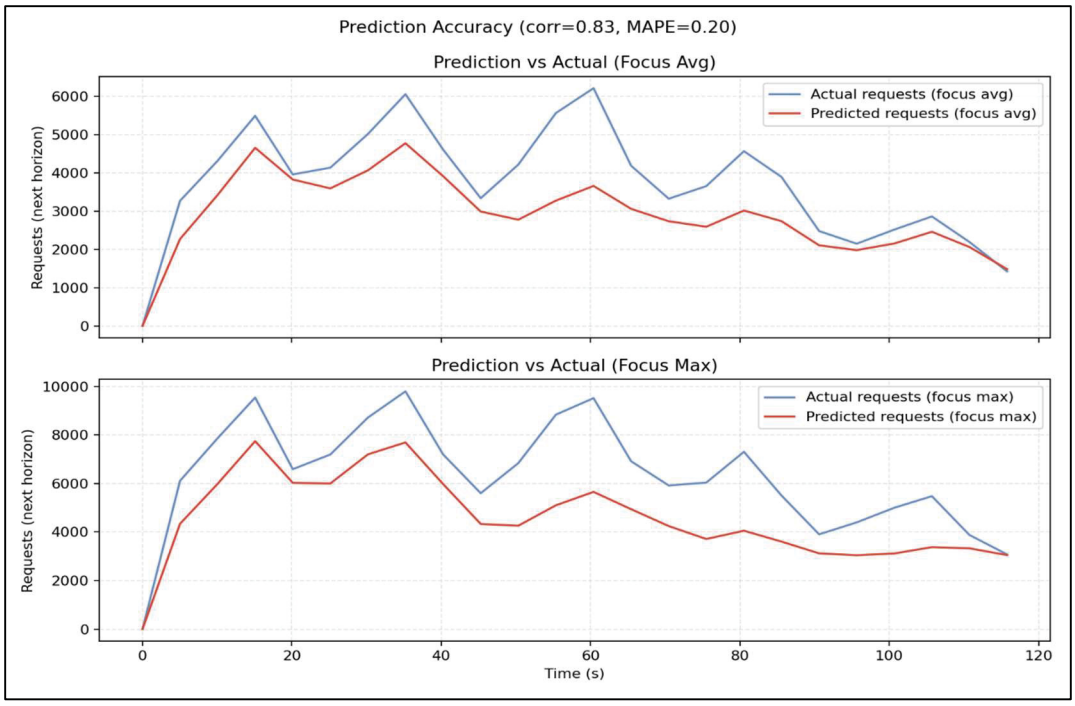


Figure 6.27 Prediction accuracy under queue stress

### 6.3.5 Time-Series Latency and Queue Pressure Dynamics

Figure 6.28 shows the system-level consequence of that forecasting signal. With prediction, latency peaks at 266.88 ms and falls to 51.63 ms by the end of the run, while queue pressure peaks at 0.05379 and decays to 0.00133. Without prediction, SAC enters persistent overload: latency rises to 3958.53 ms and remains above 3600 ms, while queue pressure peaks at 0.19814

and ends at 0.14123. Together, Figures 6.27 and 6.28 show that trend-level forecasting accuracy is sufficient to move the controller from reactive recovery to congestion prevention.

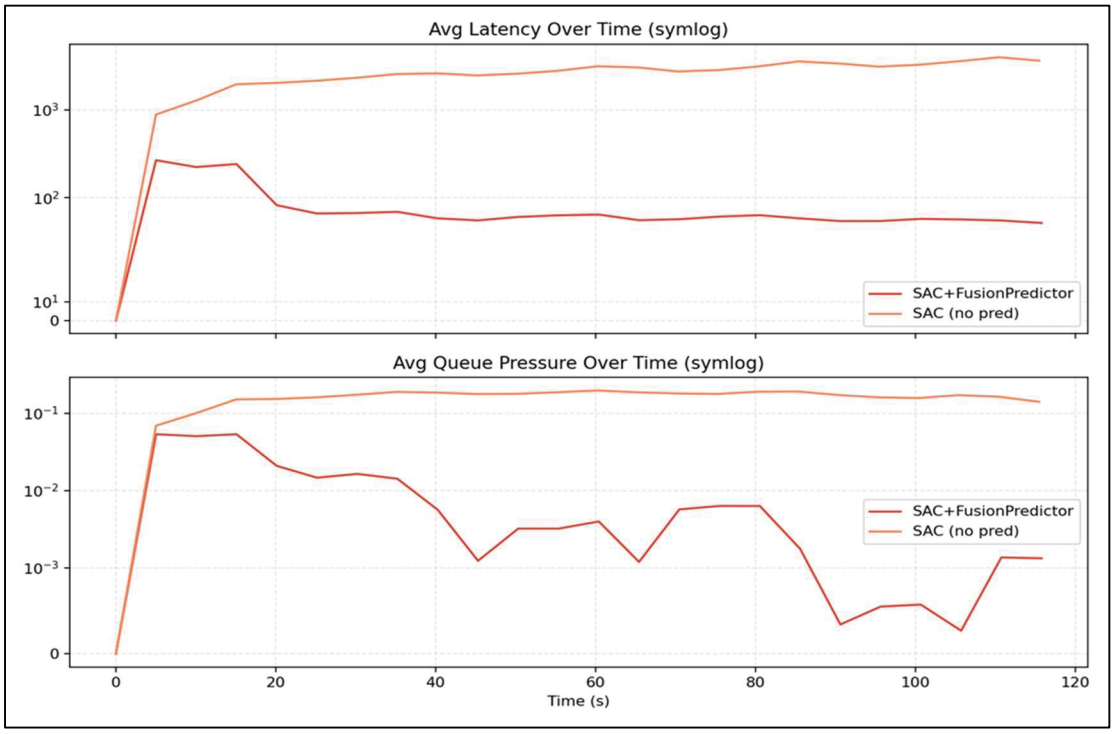


Figure 6.28 Latency and queue dynamics under queue stress

## 6.4 Relative Improvement over Every Baseline

Figure 6.29 reframes the queue-pressure comparison as relative improvement over every baseline. Each cell in the heatmap reports how much CH-SAC + Fusion Predictor improves over a given baseline on one metric, with positive values indicating better performance by the proposed method. The most important observation is that every cell is positive. CH-SAC improves over every baseline on every plotted queue-pressure metric.

Against Threshold, the proposed controller improves SLA Risk and Drop Rate by 78.0%, Scale Overhead by 79.6%, Latency Stability by 80.3%, Queue Stability by 63.7%, Delivery Ratio by 27.2%, CPU Stability by 83.0%, P99 Controller Latency by 72.2%, Recovery by 71.5%, and Useful CPU Cost by 29.6%. These are already large gains against the strongest classical

baseline. The same pattern remains visible against predictor-enabled baselines. Against GA + Fusion Predictor, CH-SAC still improves SLA Risk and Drop Rate by 80.0%, Scale Overhead by 84.0%, Latency Stability by 82.1%, Queue Stability by 65.2%, Delivery Ratio by 31.6%, CPU Stability by 86.9%, P99 Controller Latency by 72.7%, Recovery by 72.5%, and Useful CPU Cost by 32.7%. This is important because it shows that the advantage is not explained by prediction alone.

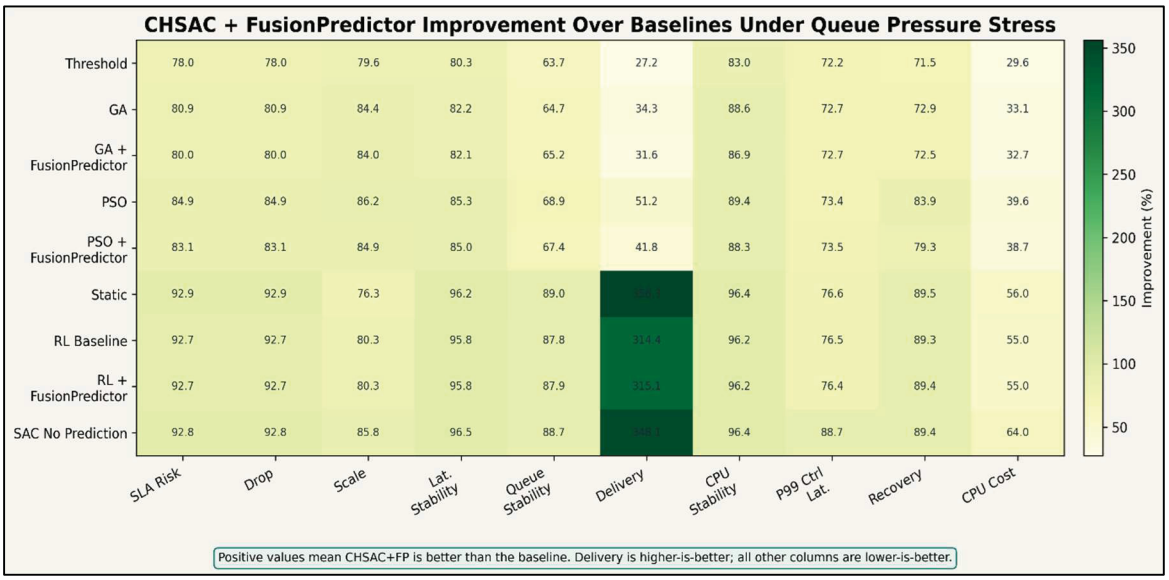


Figure 6.29 Relative improvement under queue stress

## 6.5 Resilience, Stability, and Overall Performance under Queue Stress

Figure 6.30 complements the heatmap by focusing on the most difficult queue-pressure regime: stressed-tail behaviour, queue unwinding, and useful CPU efficiency. CH-SAC + Fusion Predictor is again best on all three metrics, with P99 Controller Latency = 455.14 ms, Queue Recovery Score = 21.03, and Useful CPU Cost = 0.00435. The strongest baseline is generally Threshold, which reaches 1635.68 ms on P99 Controller Latency, 73.70 on Queue Recovery Score, and 0.00617 on Useful CPU Cost. Even this strongest baseline remains far behind the proposed controller.

The weaker controllers make the interpretation even clearer. Static, RL Baseline, RL + Fusion Predictor, and SAC No Prediction all perform very poorly once queue growth becomes persistent. SAC No Prediction is especially revealing because it reaches 4027.4 ms on P99 Controller Latency, showing that continuous control without predictive foresight can still fail badly in a queue-limited regime. The central conclusion of this section is that CH-SAC is not only better on average queue-stress behaviour. It also remains superior on the tail, on backlog recovery, and on the efficiency with which CPU effort is turned into useful delivered work. P99 Controller Latency is reported in milliseconds (ms), while Queue Recovery Score and Useful CPU Cost are dimensionless recovery and efficiency scores.

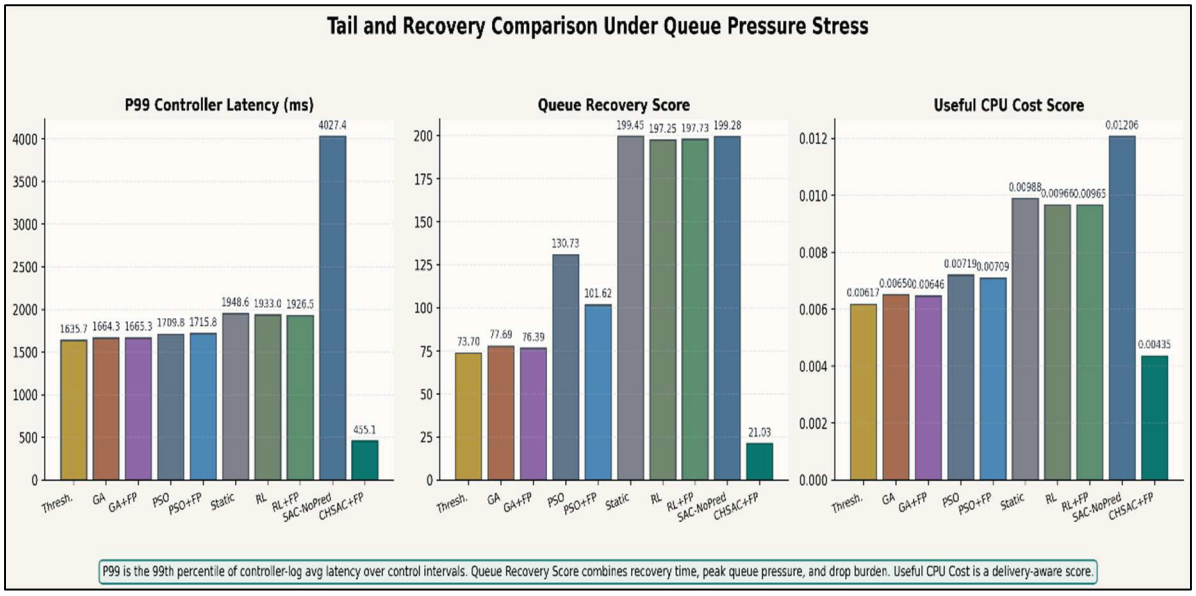


Figure 6.30 P99 latency, recovery score, and CPU cost under queue stress

### 6.5.1 Stability and Efficiency under Queue Stress

Figure 6.31 compares Latency Stability Score, Queue Stability Score, Delivery Ratio, and CPU Control Stability Score. CH-SAC + Fusion Predictor is best on all four metrics, with Latency Stability = 162.94, Queue Stability = 25.95, Delivery Ratio = 94.33%, and CPU Control Stability = 0.0684. Threshold is the strongest overall baseline, but it remains much worse at

825.30, 71.42, 74.19%, and 0.4023, respectively. These results show that CH-SAC is not only reducing drops. It is also keeping latency and queue behaviour much more stable over time while changing control decisions more smoothly than the baselines. Under queue pressure, low action variation by itself is not a sign of good control if the workload is not being served effectively. CH-SAC is strong on both stability and service outcome.

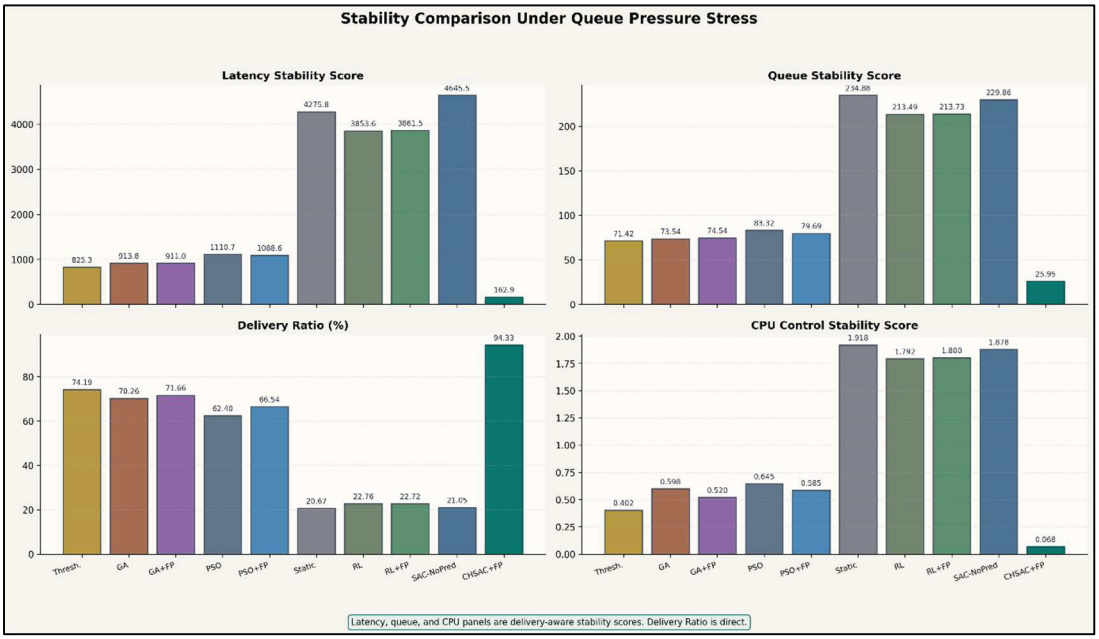


Figure 6.31 Stability and efficiency under queue stress

Figure 6.32 then provides the cleanest overall synthesis of the queue-pressure results. Instead of using a weighted composite score, it ranks each controller on the ten plotted metrics and counts the number of first-place finishes. CH-SAC + Fusion Predictor ranks first on all ten metrics, giving it 10 metric wins and an average rank of 1.0. Threshold is the strongest baseline with average rank = 2.1, followed by GA + Fusion Predictor at 3.5, GA at 4.2, PSO + Fusion Predictor at 5.3, and PSO at 6.3. RL Baseline, RL + Fusion Predictor, Static, and SAC No Prediction occupy the bottom of the ranking. This ordering is informative because it shows that prediction improves the heuristic baselines somewhat, but does not close the gap to CH-SAC. It also shows that adding prediction to the RL baseline does not materially change its overall weakness in this scenario.

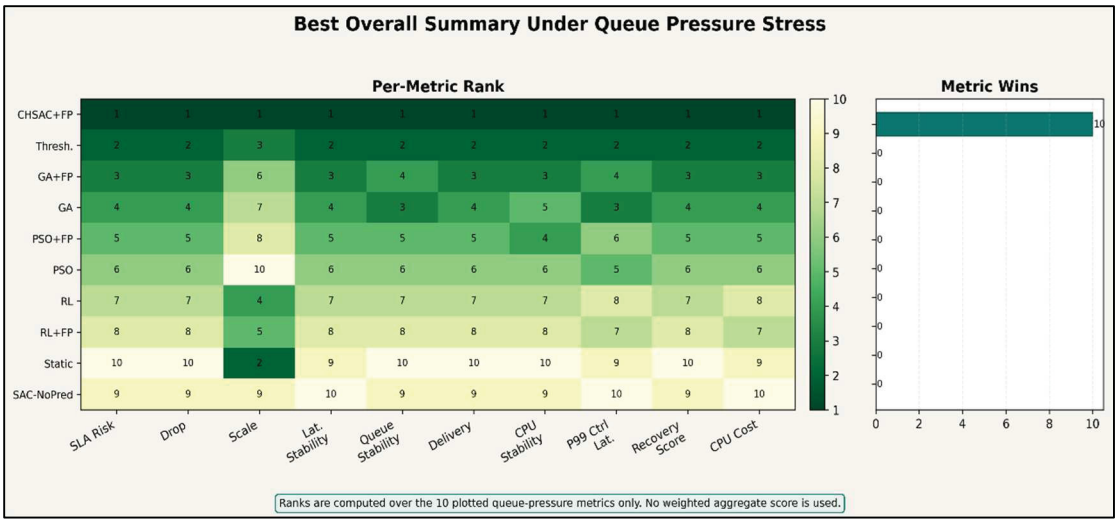


Figure 6.32 Best overall summary under queue-pressure stress

Figure 6.33 shows the multi-seed queue-pressure comparison which confirms that CHSAC + FusionPredictor is not only the best controller in the single-run results, but also the most repeatable one across the available seeds. Its mean SLA Risk, Drop Rate, and Scale Overhead remain substantially below all baselines, while its standard deviations remain small. This shows that the advantage of the proposed method under queue pressure is robust: it consistently preserves service quality, reduces traffic loss, and minimizes effective adaptation burden across repeated runs.

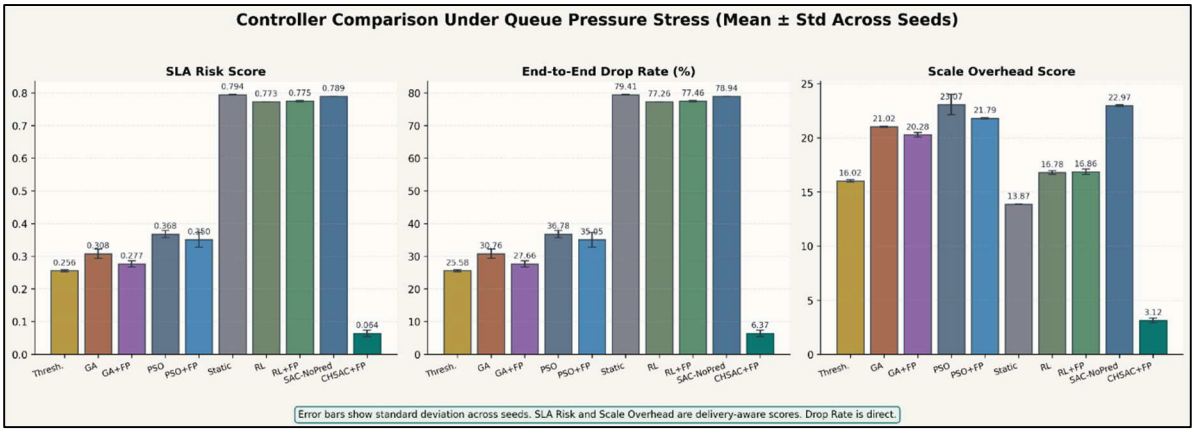


Figure 6.33 Multi-seed reliability comparison under Queue pressure stress

## 6.6 Constraint-Aware Control via Guard Mechanism

To improve SAC robustness under extreme load, a constraint-aware guard is added to the control loop. The guard monitors queue pressure, resource utilization, latency spikes, and drop ratio, and injects corrective scaling when safety thresholds are exceeded. This prevents the policy from remaining too conservative during rapid overload escalation and ensures that raw actor outputs are converted into safe executable actions.

The purpose of this section is interpretive rather than comparative. It explains how the guard changes the behaviour of the SAC family under stress and why the combination of guard-based safety and predictive guidance produces stronger control than reactive SAC alone. Section 6.6.1 examines this effect under CPU stress, while Section 6.7 then extends the same component-isolation analysis to queue stress.

### 6.6.1 SAC with Guard and Prediction under CPU Stress

This subsection revisits the CPU-stress ablation from the perspective of controller interpretation rather than primary behavioural analysis. It should therefore be read together with Section 5.10, where the same SAC-family ablation is reported in detail. The purpose here is not to duplicate the full all-controller comparison of Sections 6.1.2–6.1.5, but to isolate the roles of baseline SAC, the runtime guard, and the Fusion Predictor inside the SAC family.Baseline SAC collapses under CPU stress. Mean latency reaches 4823.0 ms, mean queue pressure rises to 0.171, drop rate reaches 80.9%, and completed throughput falls to 37,199 requests. This is the uncontrolled overload regime in which reactive SAC fails to prevent queue growth once compute capacity becomes the limiting factor.Adding the guard alone substantially improves survivability. Mean latency drops to 986.4 ms, mean queue pressure falls to 0.054, drop rate decreases to 22.1%, and throughput rises to 149,325 processed requests. The guard therefore acts as the main safety mechanism once overload becomes visible, preventing runaway queues and restoring partial service continuity. However, the behaviour remains fundamentally reactive because the intervention still begins after congestion signals have already appeared.Adding prediction on top of the guard produces a second and qualitatively different gain. Mean latency falls further to 117.0 ms, mean queue pressure to 0.007, and drop rate to 5.3%, while

throughput increases to 182,145 processed requests. The interpretation is straightforward: the guard preserves survivability, whereas prediction changes the timing of control so that scaling begins before the queue enters a self-reinforcing growth regime.

Figure 6.34 summarises this SAC-family CPU-stress ablation. It shows that the guard provides the essential safety benefit, while prediction supplies the decisive preventive gain that transforms SAC from a reactive controller into an anticipatory and safety-aware control system.

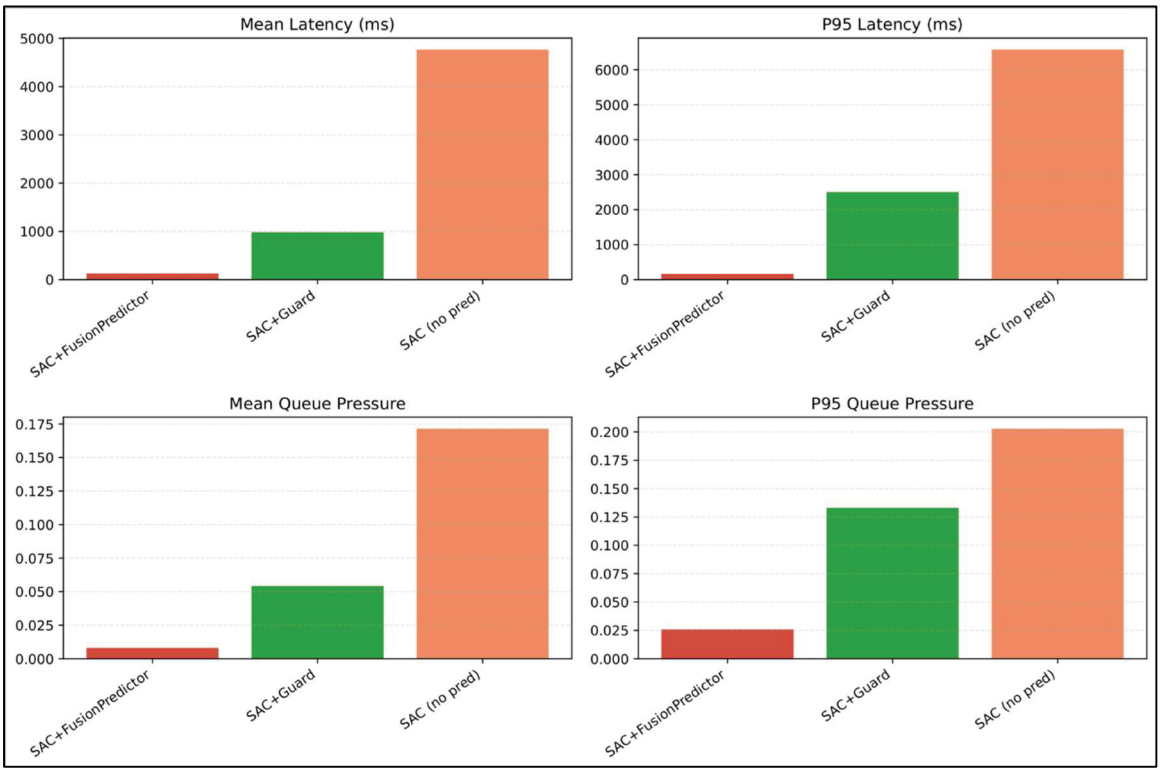


Figure 6.34 Guard and prediction effects on SAC under CPU stress

## 6.7 SAC Variant Analysis under Queue Stress

This section reports the queue-stress SAC-family variant analysis. Unlike Sections 6.3–6.5, which compare the proposed controller against all baseline families, the purpose here is to isolate the separate contributions of prediction and guarding within the SAC family.

### 6.7.1 Prediction Accuracy under Queue Stress

This subsection reports the queue-stress SAC-family variant analysis and should be interpreted separately from the full all-controller queue benchmark in Sections 6.3–6.5. Across the SAC-family variants, the predictive controller consistently outperforms the strongest non-predictive configuration on latency, queue behaviour, delivery, and recovery.Time-series behaviour explains why. With prediction, queue pressure rises only briefly and is then suppressed, keeping latency low throughout the run. Without prediction, SAC remains in a congested regime where backlog stays elevated and delay continues to grow. Prediction therefore improves the timing of control rather than merely increasing actuation.

Figure 6.35 shows that the predictor achieves correlation ≈ 0.83 and MAPE ≈ 0.20 under queue stress. Although the largest peaks remain somewhat damped, the model captures the timing and direction of demand change accurately enough to support proactive queue control. For this purpose, trend-level foresight is more important than exact reproduction of every maximum.

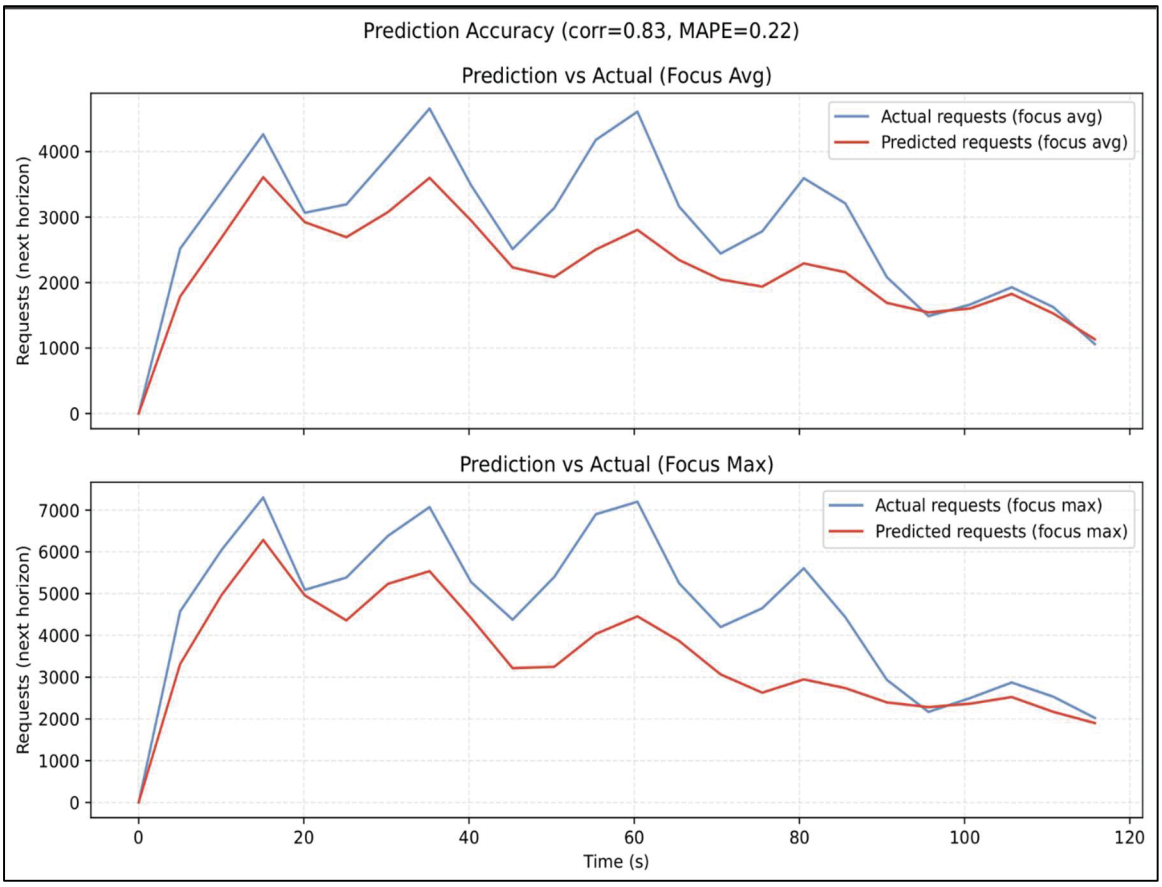


Figure 6.35 Prediction accuracy for SAC variants under queue stress

### 6.7.2 Resilience, Recovery, and Cost Trade-offs

Figure 6.36 shows that the predictive SAC variant achieves the lowest P99 latency and the fastest queue recovery among the SAC-family configurations. Recovery occurs on the order of about 10 s for the predictive controller, compared with roughly 60 s for SAC + Guard and about 110 s for SAC No Prediction. This stronger recovery is accompanied by purposeful adaptation effort: the controller uses somewhat more well-timed scaling effort in order to prevent deep congestion rather than paying a much larger recovery cost afterward.

The key point is that the predictive controller is not simply more aggressive. It intervenes earlier, which allows the system to remain in a lower-cost operating region. The result is a better

balance between stressed-tail delay, recovery speed, and effective control effort. P99 latency is reported in milliseconds (ms), recovery time is reported in seconds (s), and CPU cost is a dimensionless control-effort score.

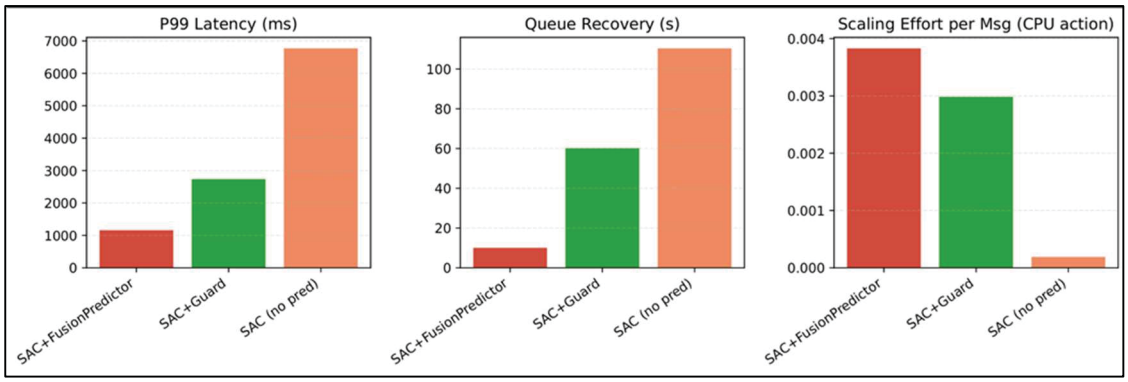


Figure 6.36 Tail, recovery, and CPU cost for SAC variants

### 6.7.3 Stability, Efficiency, and Throughput Analysis

Figure 6.37 shows the same pattern for stability and service completion. The predictive SAC variant has lower latency jitter and queue jitter, and a higher delivery ratio, than both guard-only and non-predictive SAC. Although SAC No Prediction can sometimes appear to have lower action variation, that mainly reflects weak adaptation rather than good control.

The relevant result is that the full predictive controller combines smoother behaviour with better service outcome. Under queue stress, that combination is more important than low actuation by itself, because the real objective is not merely to change resources slowly, but to keep the workload out of the backlog-growth regime while still completing service efficiently. Latency jitter is measured in milliseconds (ms), queue jitter is computed from normalized queue pressure, delivery ratio is reported as a percentage, and action variation is dimensionless.

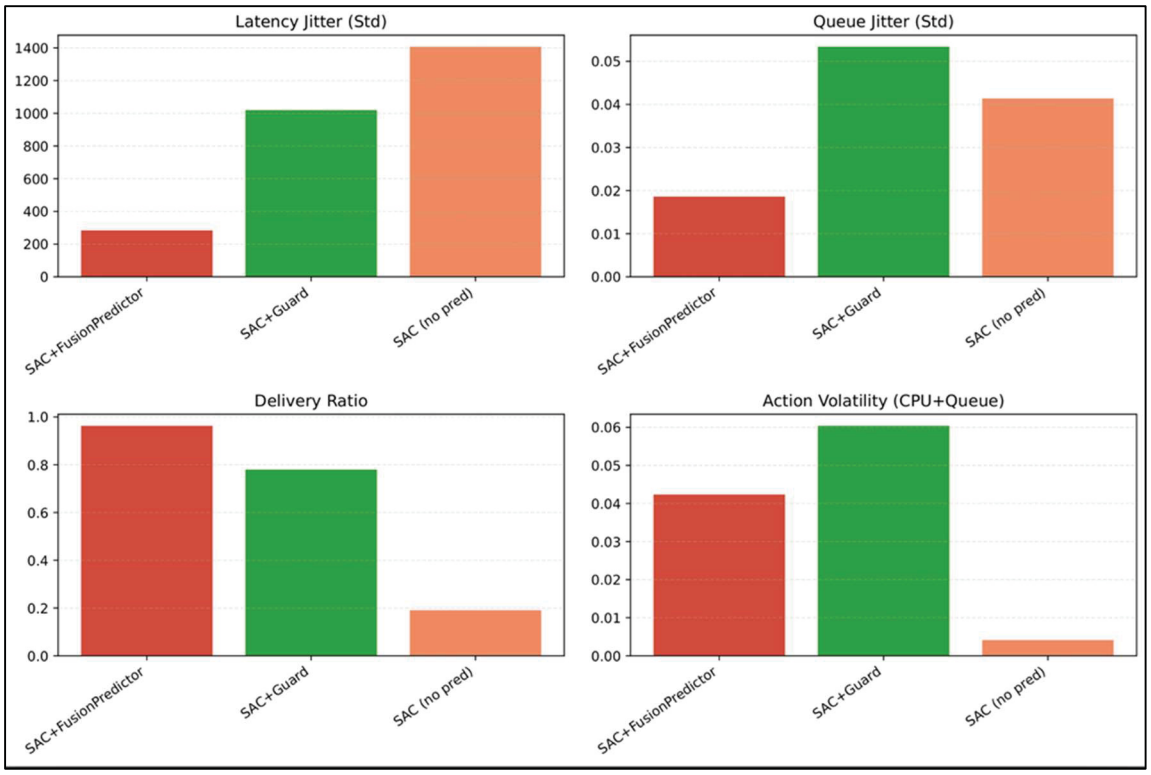


Figure 6.37 Stability, efficiency, and throughput for SAC variants

### 6.7.4 Proactive Congestion Prevention and Recovery

Figure 6.38 provides the clearest behavioural interpretation of the queue-stress SAC-family analysis. Predicted demand rises before severe congestion becomes sustained, and the controller responds by activating capacity earlier. As a result, SAC + Guard + Fusion Predictor keeps queue pressure near zero after the initial transient and stabilises latency at a low level. SAC No Prediction, by contrast, remains trapped in a congested regime with sustained queue buildup and rapidly growing delay.

Overall, the queue-stress variant analysis confirms the same component hierarchy seen in the CPU ablation. The guard improves robustness, but prediction is what shifts the controller from reactive safety enforcement to proactive congestion prevention.

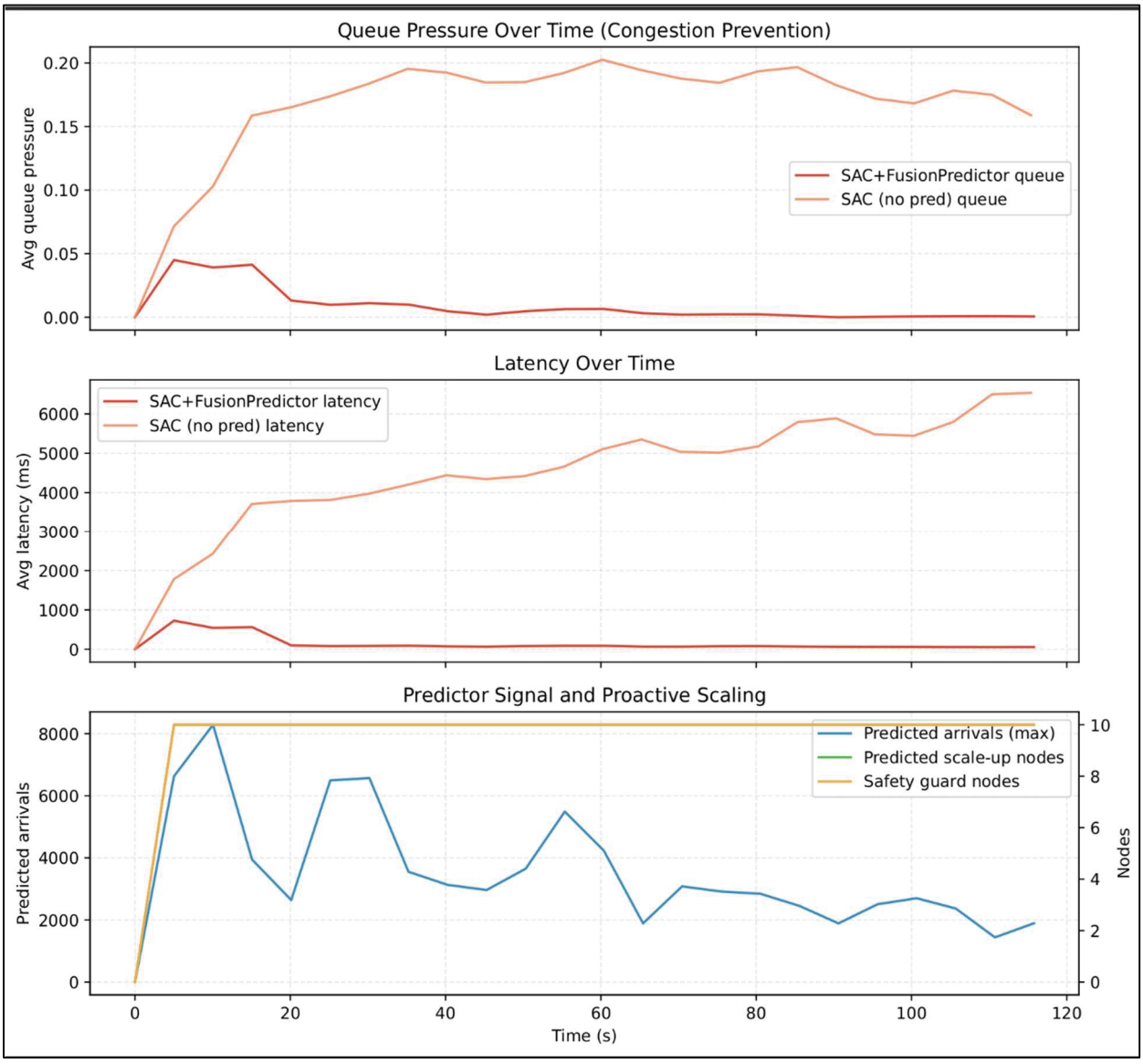


Figure 6.38 Proactive congestion control under queue stress

## 6.8 Equal-Workload Benchmark and Controller Comparison

### 6.8.1 Controller QoS Comparison

Figure 6.39 compares six controllers under the same normalised workload and SLA conditions. The proposed CH-SAC with Fusion Predictor achieves the highest QoS score at 72.53%,

followed by GA + Predictor at 69.86%. The remaining methods perform less effectively: GA reaches 52.51%, PSO 47.45%, Baseline SAC 37.41%, and Baseline RL 17.81%.

These ranking highlights three main findings. First, predictive support materially improves control quality, since both predictor-assisted methods outperform their non-predictive counterparts. Second, the gain does not come from SAC alone: the large gap between Baseline SAC and CH-SAC with Fusion Predictor shows the value of integrating forecasting, mobility-aware state information, and proactive control. Third, the proposed method provides the best overall balance between QoS and real-time decision-making. These results support the central claim of the thesis that short-horizon prediction combined with constrained continuous control improves fog-resource provisioning under mobility-driven load. The QoS score is reported as a percentage (%), where higher values indicate better overall service quality under the normalized workload.

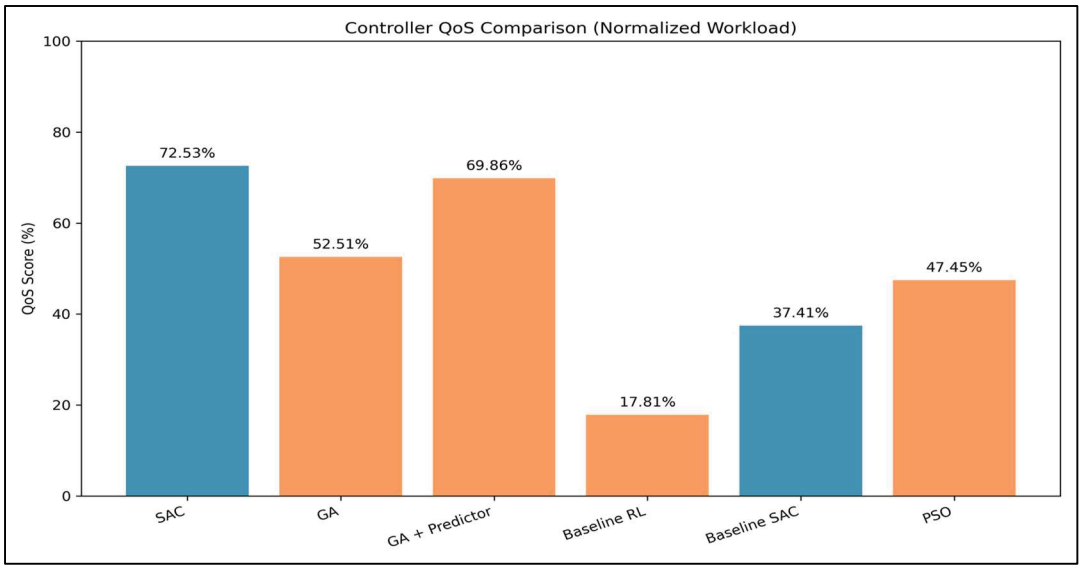


Figure 6.39 QoS comparison under normalised workload

## 6.9 Equal-Workload Comparison, Fairness, and Resilience

### 6.9.1 Composite Fairness and Resilience Analysis

Fog-resource controllers must balance fairness, meaning the equitable distribution of delay and service quality across nodes, with resilience, meaning the ability to restore acceptable performance quickly after a disturbance. Figure 6.40 presents a composite analysis of these trade-offs for eight strategies: CH-SAC with Fusion Predictor, Baseline SAC, Static, GA, GA + Fusion Predictor, PSO, Baseline RL, and RL + Fusion Predictor.The upper panel reports the Composite Fairness and Resilience Score on a 0–1 scale, where higher values indicate better overall performance. The lower panels show the component metrics used to construct this score: queue coefficient of variation (CV), latency Gini coefficient, and time to recovery (TTR). Lower values are desirable for all three raw component measures.The composite score is defined as:

$$\text{Composite} = 0.80\, f_{\text{resilience}} + 0.10\, f_{\text{fairness}} + 0.10\, f_{\text{smoothness}}, \quad (6.1)$$

where each $f_i$is normalised to the interval $[0, 1]$, with larger values indicating better performance after normalisation. Queue CV and latency Gini are inverted after normalisation so that lower dispersion corresponds to a higher component score. Resilience is normalised on a logarithmic scale to prevent extremely long recovery times from dominating the aggregate score. Smoothness measures the fraction of small resource adjustments, reflecting gradual rather than abrupt scaling behaviour.

Among all evaluated methods, CH-SAC with Fusion Predictor attains the highest composite score at 0.87 ± 0.01, indicating the strongest overall balance among resilience, fairness, and smoothness. A second tier comprising Static, Baseline SAC, and GA clusters around 0.74, while GA + Fusion Predictor reaches 0.49. The weakest methods are PSO, Baseline RL, and RL + Fusion Predictor, all of which fall below 0.30, indicating limited adaptability and poor recovery behaviour under dynamic conditions.

The lower panels help explain this ranking. CH-SAC with Fusion Predictor achieves the shortest recovery time, restoring acceptable performance in approximately 10 s, whereas Static requires around 100 s, GA + Fusion Predictor about 460 s, and the most reactive learning-based baselines extend into the $10^3$–$10^4$ s range. This fast recovery is the dominant reason for the superior composite score of the proposed controller. In fairness-related measures, rigid controllers such as Static and GA perform better because their allocations are more uniform, but this comes at the cost of reduced responsiveness. By contrast, the proposed controller deliberately concentrates resources in overloaded regions, accepting some inequality in exchange for much stronger resilience.

Overall, Figure 6.40 shows that CH-SAC with Fusion Predictor provides the strongest operational balance among fairness, smoothness, and resilience. Static and heuristic controllers may achieve more even load distribution, but they recover too slowly for highly dynamic mobility-driven fog environments. The proposed controller instead prioritises rapid restoration of service while maintaining controlled, non-oscillatory behaviour. The composite fairness and resilience score is normalized to [0,1], queue CV and latency Gini are dimensionless dispersion measures, and time to recovery (TTR) is measured in seconds (s).

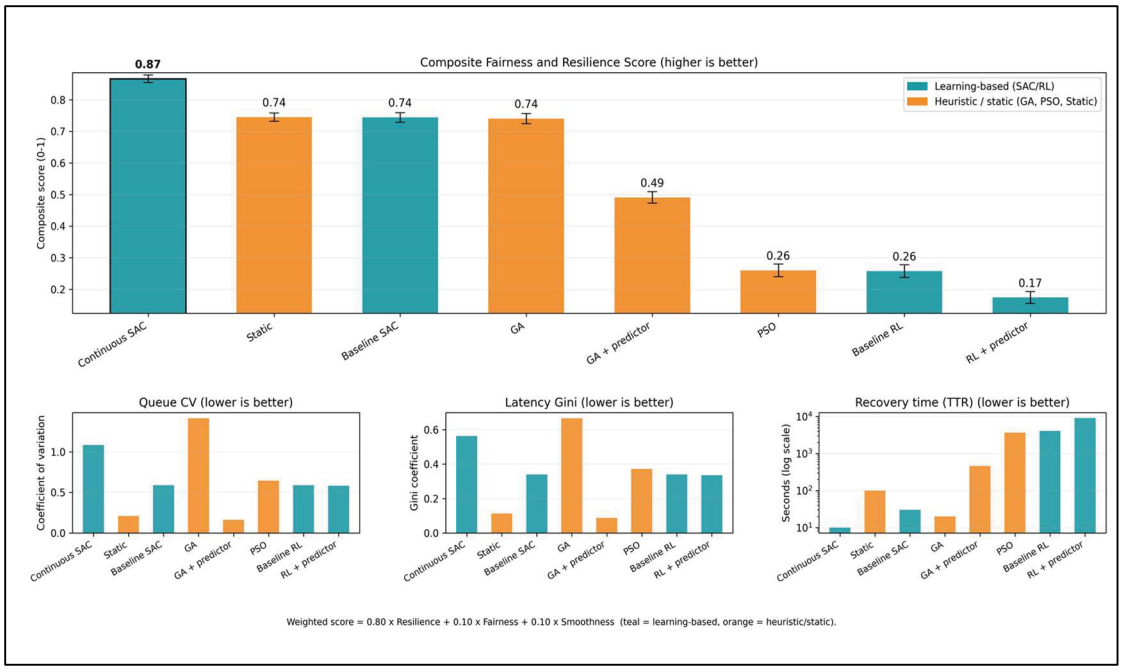


Figure 6.40 Fairness and resilience under equal workload

### 6.9.2 Latency Evaluation under Equal-Workload Conditions

Latency is a primary QoS indicator in fog environments. To compare controllers fairly, each method is evaluated under identical workload and mobility conditions, with the same topology, traffic intensity, and random-seed schedule. Because some controllers do not complete all assigned work within the simulation horizon, the evaluation uses a penalty-adjusted latency metric:

$$L_{effective} = L_{raw} + P(1 - completion\ rate) \tag{6.2}$$

where $L_{raw}$ is the mean observed latency, $P$ is a large penalty constant, and completion rate is the fraction of tasks successfully completed. This preserves the relative ordering of controllers while penalising methods that leave substantial work unfinished.

### 6.9.3 End-to-End Latency

Figure 6.41 reports the penalty-adjusted end-to-end latency, measured from message emission at the user device to reception at the sink node. CH-SAC with Fusion Predictor achieves the lowest latency at approximately 757 ms, corresponding to roughly a 25% improvement over Baseline SAC (about 1,002 ms) and substantially lower latency than Baseline RL (1,658 ms) and GA (1,540 ms). The proposed controller also outperforms GA + Fusion Predictor (981 ms) and Baseline RL + Fusion Predictor (1,096 ms), indicating that prediction alone is insufficient unless it is integrated into a continuously adaptive control architecture.

Although the Static policy appears competitive under equal-workload conditions, it lacks the ability to adapt when mobility concentration shifts spatially. In such situations, fixed placements lead to longer effective paths and local queue growth. CH-SAC with Fusion Predictor, by contrast, continuously reallocates capacity in response to predicted load movement, thereby maintaining the lowest overall delay.

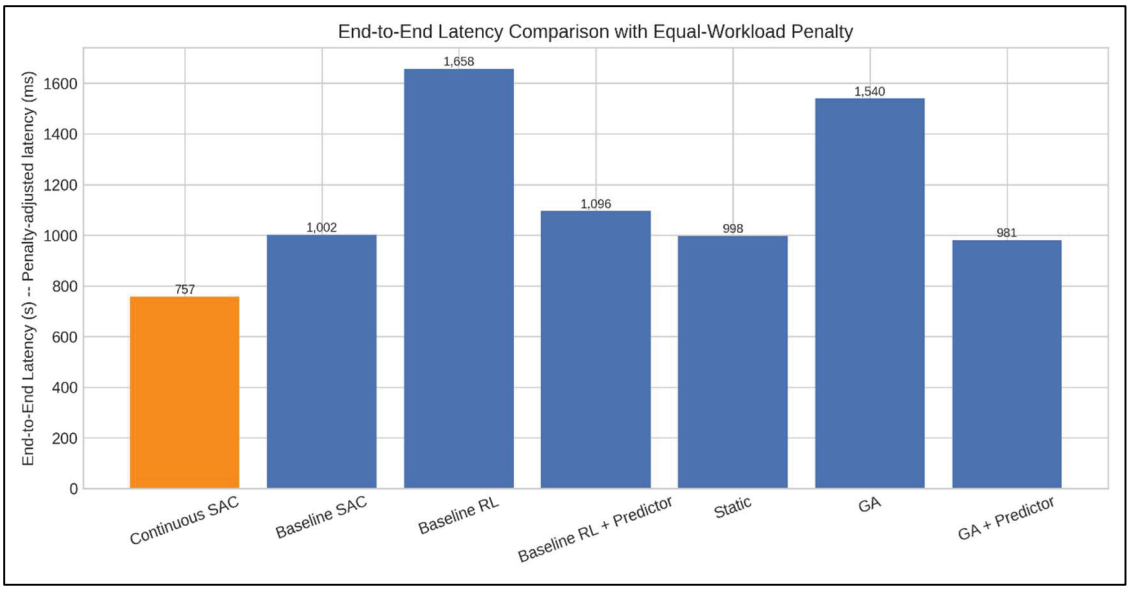


Figure 6.41 Penalty-adjusted end-to-end latency under equal workload

### 6.9.4 Ingress Latency

Figure 6.42 reports the corresponding ingress latency, defined as the delay between task generation and arrival at the first fog node. The proposed controller again achieves the best result at approximately 752 ms. By comparison, Baseline SAC incurs roughly 2,194 ms and Baseline RL around 3,204 ms, while GA and GA + Fusion Predictor remain above the proposed method at approximately 1,229 ms and 979 ms, respectively. These results reinforce the interpretation that the proposed controller is more effective at placing computation close to expected demand and reducing the routing and queueing delays accumulated before fog processing begins.

Across both end-to-end and ingress latency, the proposed controller consistently delivers the shortest delays and the highest completion rates. These gains arise from three interacting mechanisms: predictive foresight supplied by the Fusion Predictor, continuous online adaptation through CH-SAC, and smooth continuous control of resource levels. Although the absolute reduction from approximately 1.0 s to 0.76 s may appear modest, it is operationally significant because it increases the number of requests meeting their deadlines and reduces exposure to long-tail delay escalation.

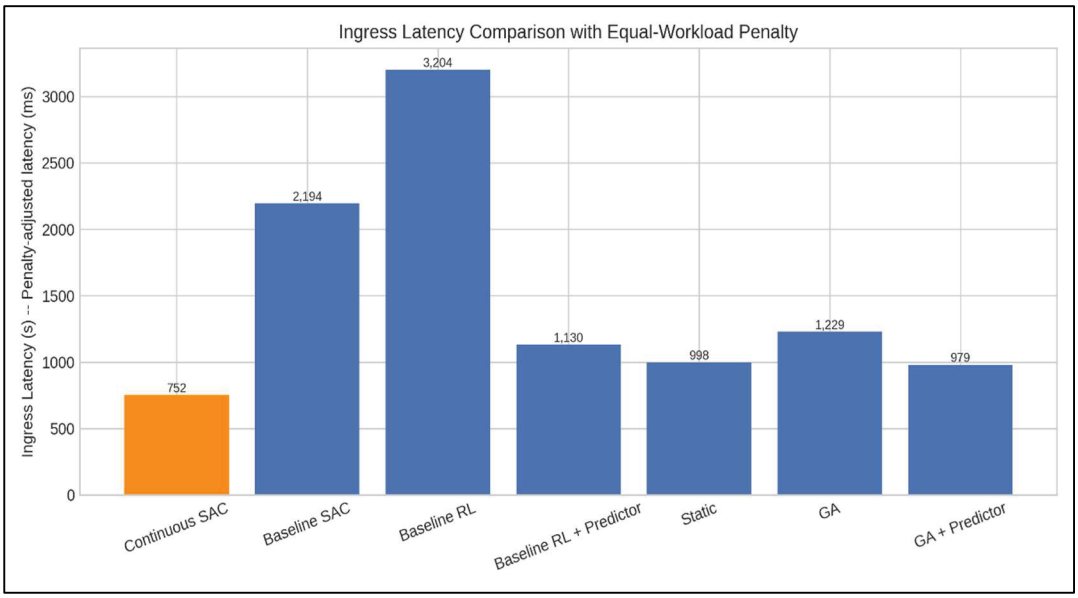


Figure 6.42 Ingress latency under equal-workload conditions

### 6.9.5 Numeric QoS Reference Table

Table 6.2 lists the same normalised-workload QoS scores shown graphically in Figure 6.39 and is included primarily as a compact numeric reference. The ranking is unchanged: CH-SAC + Fusion Predictor performs best, followed by the strongest predictor-enabled heuristic baseline, while the reactive RL baselines remain weakest. Because Figure 6.39 already provides the main interpretation, Table 6.2 should be read as a numeric complement rather than as a second full discussion.

Table 6.2 QoS comparison under normalised workload

| **Controller** | **QoS Score (%)** |
|---|---|
| CH-SAC + Fusion Predictor | 72.53 |
| GA + Fusion Predictor | 69.86 |
| GA | 52.51 |
| PSO | 47.45 |
| Baseline SAC | 37.41 |
| Baseline RL | 17.81 |

## 6.10 Generalisation and Modelling Transparency

Although the evaluation uses a real-world urban mobility trace derived from the Beijing T-Drive dataset, the methodology itself is dataset-agnostic. The analysis relies on relative timing events message emission, processing, and reception timestamps recorded by the simulation and emulation environments. As a result, the same evaluation pipeline can be applied to alternative mobility traces or synthetic stress scenarios without structural modification.

All latency values are obtained by instrumenting each stage of the service path from sensor to sink, ensuring that queueing, routing, and processing delays are all captured explicitly. Simulation parameters, controller configurations, and metric definitions are documented so that the reported results can be independently reproduced and verified.

## 6.11 Adaptation and Recovery under Controlled Disturbance

### 6.11.1 Metrics and Disturbance Scenario

Figure 6.43 summarizes the adaptation capabilities of the evaluated controllers using two complementary indicators: time to recovery (TTR) and excess violation area. These metrics quantify how quickly and how effectively each controller restores acceptable QoS after a sudden mobility-driven disturbance.

The first metric, time to recovery, is defined as the interval between the onset of the disturbance and the first time at which the smoothed end-to-end latency remains continuously below the SLA threshold for at least 10 s. The second metric, excess violation area, measures the cumulative area above the SLA threshold during the recovery period:

$$A_{exc} = \int_{t_{dist}}^{t_{rec}} \max(0, L(t) - L_{SLA})\, dt \qquad 6.3)$$

where $L(t)$is the smoothed average end-to-end latency, $L_{SLA}$is the SLA threshold, $t_{dist}$is the disturbance onset, and $t_{rec}$is the recovery time.

All algorithms are evaluated under identical conditions: the same fog topology, the same Beijing mobility trace, the same workload profile, the same SLA target, and the same per-decision time budget. The disturbance scenario simulates a sudden mobility-induced overload by combining a temporary workload spike with a mobility surge concentrated over a compact set of hotspot nodes. This creates a controlled but severe overload episode in part of the topology while leaving the rest of the network under nominal conditions.

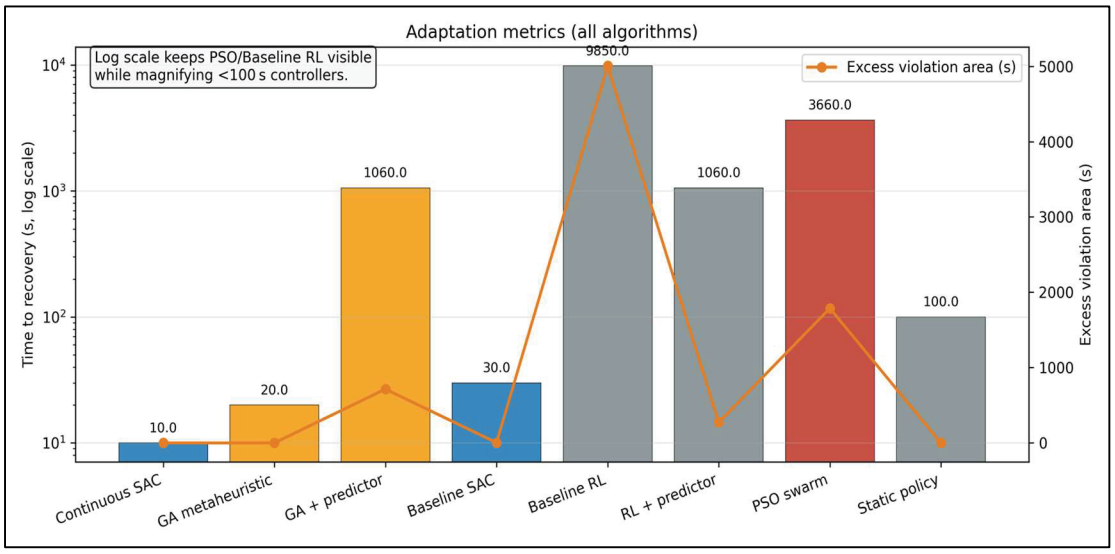


Figure 6.43 Adaptation and recovery under controlled disturbance

### 6.11.2 Quantitative Results and Interpretation

Under this controlled disturbance, CH-SAC with Fusion Predictor exhibits the fastest and cleanest recovery, with TTR ≈ 10 s and an almost negligible excess violation area. The next strongest methods are GA at approximately 20 s and Baseline SAC at approximately 30 s, followed by Static at 100 s. More reactive or computationally heavy methods perform substantially worse: GA + Fusion Predictor requires about 1,060 s, PSO around 3,660 s, Baseline RL nearly 9,850 s, and RL + Fusion Predictor roughly 1,060 s.

These differences reveal the same structural pattern seen in the latency and QoS results. The proposed controller benefits from the combination of fused short-horizon prediction, continuous online adaptation, and smooth continuous action selection. It can therefore react to

disturbances before they fully propagate into prolonged SLA violations. Heuristic optimisers, by contrast, suffer from recomputation overhead, while purely reactive RL baselines lack sufficient foresight to prevent long-lasting degradation.

The excess-violation results reinforce this interpretation. Slow controllers accumulate large regions of SLA excess because they remain above the threshold for long periods, whereas CH-SAC with Fusion Predictor keeps both recovery time and integrated violation small. In mobility-driven fog systems, where transient overload can rapidly cascade through the topology, this ability to restore acceptable performance within a few seconds is operationally critical.

Taken together with the equal-workload latency and QoS results, the adaptation analysis supports a central claim of the thesis: embedding the Fusion Predictor inside a continuously learning CH-SAC controller shifts fog-resource management from reactive stabilisation to proactive resilience, enabling fast recovery and sustained QoS under highly dynamic operating conditions.

### 6.11.3 Why CH-SAC with Fusion Predictor Recovers Faster

The superior recovery behaviour of CH-SAC with Fusion Predictor arises from the interaction of three mechanisms. First, the Fusion Predictor provides short-horizon foresight about rising demand. Second, CH-SAC updates its policy continuously during execution, allowing it to track drift in workload and mobility patterns. Third, the controller operates in a continuous action space, which enables smooth and low-amplitude corrective actions rather than abrupt resource jumps. The original equal-workload analysis explicitly attributes the latency gains of continuous SAC to these same three interacting mechanisms: forecast-driven foresight, continuous online adaptation, and smooth continuous control.

This makes the proposed controller fundamentally different from static or meta-heuristic baselines. GA and PSO may produce reasonable allocations offline, but they do not adapt continuously during deployment. Baseline RL and Baseline SAC can adapt online, but without fused predictive signals they remain closer to a catch-up regime, reacting after congestion has already started to form. By contrast, CH-SAC with Fusion Predictor combines online adaptation with anticipatory scaling, which is why it recovers more quickly and with less accumulated violation area.

## 6.12 Prediction Accuracy

This section briefly recaps the queue-stress prediction evidence in order to clarify why predictive guidance changes the SAC control regime. It does not introduce a new controller comparison. Instead, Figures 6.44 and 6.45 summarise the temporal congestion pattern and the demand-tracking accuracy that support proactive queue-stress control.

### 6.12.1 Time-Series Congestion Dynamics

Figure 6.44 compares the time evolution of average latency and queue pressure for SAC + Fusion Predictor and SAC No Prediction under queue stress. The two controllers operate in clearly different regimes. With prediction, latency remains bounded after the initial transient and queue pressure decays toward near-zero values. Without prediction, both latency and queue pressure rise sharply and remain high, indicating that the controller becomes trapped in a congested state. This difference shows why reactive control fails under queue stress: once queues begin to grow, delayed scaling becomes costly and recovery is slow. By contrast, the predictive controller stabilises the system before buffers enter sustained growth. The result is not just lower peaks, but a different operating regime in which congestion is largely prevented rather than drained afterward.

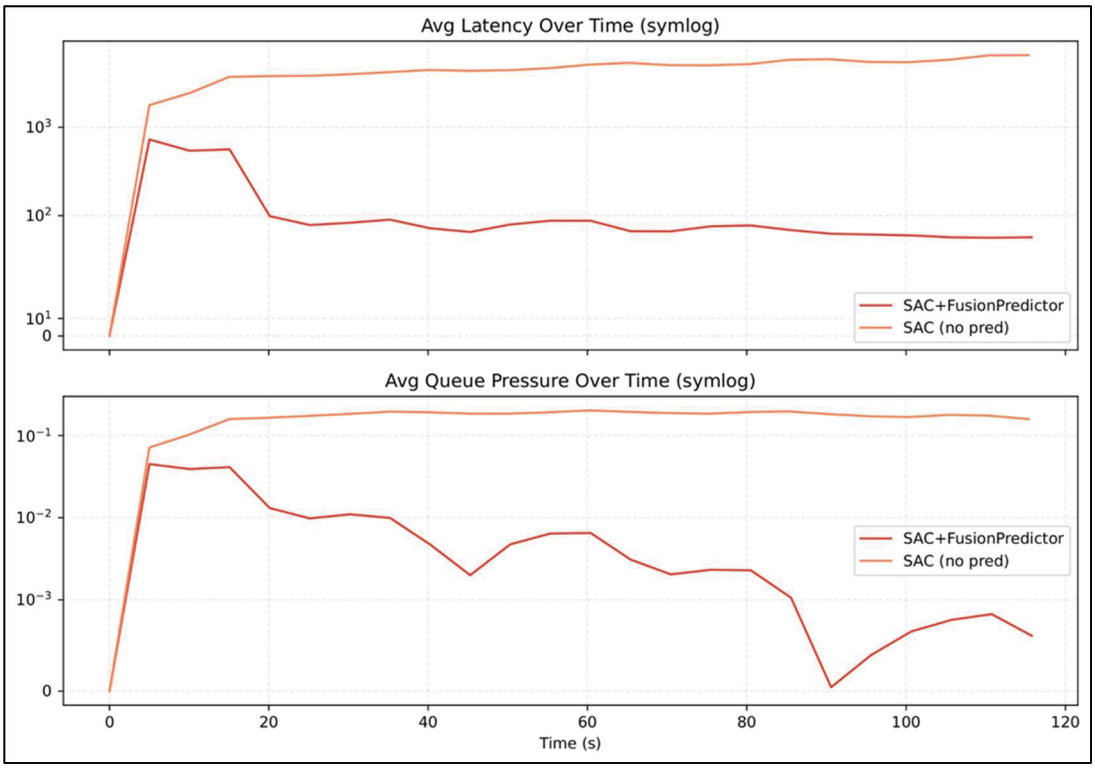


Figure 6.44 Congestion dynamics over time under queue stress

### 6.12.2 Predictor Accuracy and Demand Tracking

Figure 6.45 compares predicted and actual request volumes for both focus-average and focus-maximum demand. The predictor achieves correlation ≈ 0.83 and MAPE ≈ 0.20, indicating strong short-horizon alignment in timing and overall shape. The main limitation is that the largest peaks are damped rather than reproduced exactly, especially for focus-maximum demand. For control, however, this level of accuracy is sufficient because the controller primarily needs early warning of rising load rather than exact reproduction of every peak magnitude.

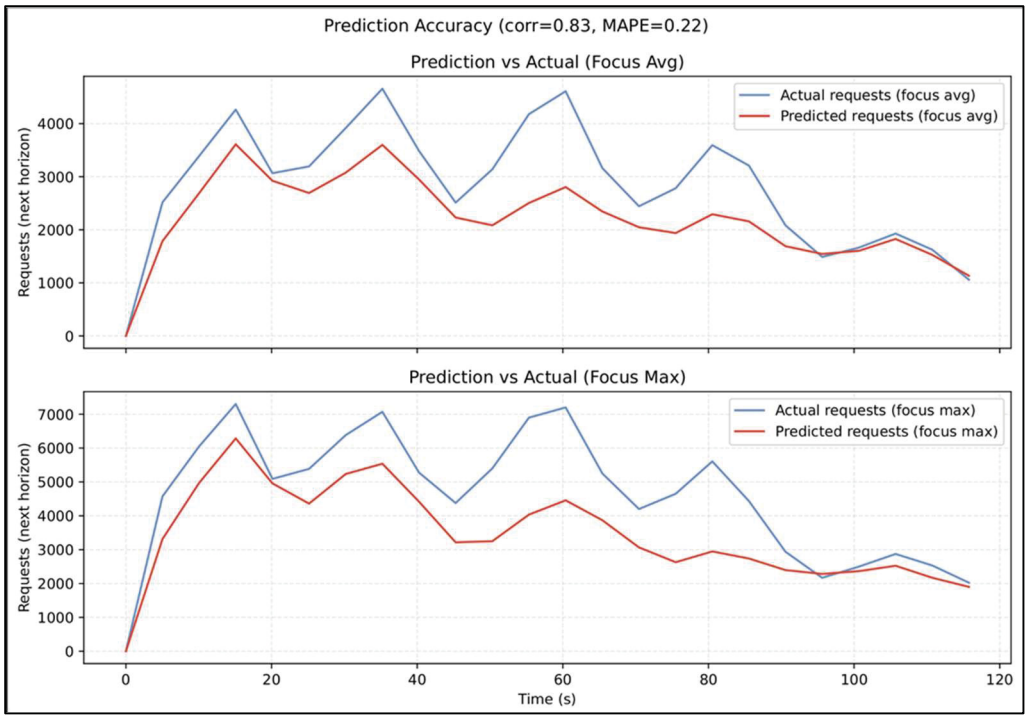


Figure 6.45 Prediction accuracy and demand tracking under queue stress

### 6.12.3 Why Prediction Changes Control Outcomes

Taken together, Figures 6.44 and 6.45 show why prediction changes control outcomes. Figure 6.44 demonstrates the system-level effect: with prediction, latency remains bounded after the initial transient and queue pressure decays toward near-zero values, whereas without prediction both metrics remain in a persistently congested regime. Figure 6.45 explains why this is possible: the controller receives short-horizon demand signals that are accurate in timing and overall scale even though peak magnitudes are damped. This is enough to shift the controller from reacting after queue growth becomes visible to acting before sustained congestion forms.

# CHAPTER 7

# DISCUSSION

## 7.1 Chapter Overview

This chapter synthesizes the findings developed in Chapters 5 and 6 and clarifies their meaning for proactive fog-resource management. While Chapter 5 analysed how the proposed framework behaves during execution and Chapter 6 quantified its performance under controlled stress scenarios, the present chapter focuses on interpretation rather than new primary results. It explains the main conceptual contribution of the thesis, compares the proposed framework with the principal baseline families, discusses the complementary roles of prediction, continuous control, and safety guarding, answers the research questions, and identifies the principal limitations and future directions of the work.

## 7.2 Discussion of the Main Findings

The main finding of this thesis is that proactive fog-resource management under mobility-driven demand requires more than reactive continuous-control reinforcement learning. The central contribution of CH-SAC with Fusion Predictor is a change in the timing of control. Rather than waiting for queue growth, latency spikes, or packet drops to reveal that overload has already formed, the proposed framework uses short-horizon predictive signals to provision capacity before congestion becomes dominant. This shift from reactive recovery to proactive congestion prevention is the most important conceptual result of the thesis.

A second key finding is that the value of prediction lies less in perfect point accuracy than in operationally useful foresight. In the behavioural analysis, the Fusion Predictor achieved a mean absolute error of 3.45 taxis and a Pearson correlation of 0.95, while proactive control was reflected in a 78.2% proactive-action share, a median lead time of 17.0 s, a proactive overload-prediction F1-score of 0.890, and an average precision of about 0.87. These results

show that the predictor is not an isolated monitoring tool. Instead, its forecasts are translated into earlier and better-timed scaling decisions that materially change how the controller behaves.

The behavioural evidence also shows that proactive control is spatially selective rather than uniformly distributed across the topology. Forecast-driven actions are concentrated in the busiest parts of the system, and CPU provisioning follows the spatial structure of predicted mobility demand. The framework therefore does not simply add more resources everywhere. It uses short-horizon prediction to decide where intervention is most likely to matter, which is especially important when hotspots migrate across the city. This confirms that the Fusion Predictor contributes both temporal foresight and spatial guidance.

A third major finding is that the importance of prediction depends on the stress regime. Under CPU stress, prediction is decisive because compute saturation develops with enough delay that early scaling can prevent queue growth before it becomes self-reinforcing. Under queue stress, this effect becomes even stronger because once buffers begin to saturate, delayed correction becomes expensive and slow. In the full queue-stress comparison, CH-SAC + Fusion Predictor achieves 79.3 ms mean latency and 241,454 completed requests while also attaining the strongest reliability and stability profile, including 5.67% drop rate and 94.33% delivery ratio. The predictive controller therefore keeps the system out of the queue-growth regime rather than attempting to drain queues after overload has already formed.

Under flash-crowd / hotspot-shift stress, the balance changes. Here, overload appears abruptly when demand becomes concentrated over a small subset of nodes. As a result, the guard provides the dominant immediate safety benefit, while prediction contributes mainly by improving hotspot tracking, capacity redistribution, and control smoothness. Even in this setting, the full framework remains strongest overall, achieving 41.65 ms mean latency and 162,285 completed requests. These results show that prediction remains useful under flash-crowd stress, but its role shifts from preventing delayed overload to improving spatial alignment under rapidly moving hotspots.

Another important finding concerns the relationship between adaptivity and stability. A common concern with learning-based control is that average gains may be achieved only by aggressive or oscillatory actuation. The present results point in the opposite direction. Under

queue stress, CH-SAC + Fusion Predictor not only attains the best latency and throughput, but also the best tail, recovery, and stability profile, including P99 controller latency of 455.14 ms, Queue Recovery Score of 21.03, Delivery Ratio of 94.33%, and the strongest Latency Stability, Queue Stability, and CPU Control Stability scores among all compared methods. The framework therefore does not merely act earlier; it acts earlier while preserving smoother and more reliable control behaviour than the baselines.

The thesis also provides a more nuanced view of fairness and efficiency. Static or rigid controllers can appear more equal because they allocate capacity more uniformly across nodes, but this uniformity comes at the cost of slower recovery and weaker responsiveness to localized overload. Under equal-workload comparison, CH-SAC with Fusion Predictor attains the strongest overall operational balance, with a composite fairness-and-resilience score of 0.87 ± 0.01, penalty-adjusted end-to-end latency of about 757 ms, the highest QoS score at 72.53%, and recovery in approximately 10 s. This indicates that modest short-term inequality in allocation is not a flaw but a rational trade-off in mobility-driven fog systems, where rapid restoration of service is often more important than perfectly even short-term distribution.

Finally, the discussion clarifies an important operational boundary. Several YAFS-based latency analyses show that once requests reach the fog-processing stage, local computation can become negligible relative to total end-to-end delay. This means that the proposed controller can optimize the controllable fog tier very effectively, while residual delay may still be dominated by transport, routing, and sink-side effects outside the controller's direct action space. This observation does not weaken the thesis contribution. Instead, it defines its scope more precisely and explains why the framework should be evaluated not by latency alone, but by the combined evidence of anticipation, congestion suppression, reliability, and recovery.

## 7.3 Comparative Interpretation and Practical Implications

The comparative results show that prediction must be integrated into the control architecture rather than merely attached to it. Predictor-augmented heuristic baselines do improve relative to their non-predictive versions, which confirms that short-horizon forecast information is intrinsically useful. However, they do not match the performance of CH-SAC with Fusion

Predictor. Under normalised workload, the proposed controller achieves the highest QoS score at 72.53%, followed by GA + Fusion Predictor at 69.86%, while GA, PSO, Baseline SAC, and Baseline RL remain substantially lower. This ranking indicates that the strongest gains arise when short-horizon foresight is combined with continuous online adaptation and smooth continuous action selection

### 7.3.1 Why the Proposed Framework Outperforms Baselines

The comparison with Baseline SAC provides the clearest evidence that the gain does not come from SAC alone, but from the integration of prediction and safety-aware control. In the full all-controller CPU comparison, CH-SAC + Fusion Predictor reduces mean latency from 4812.1 ms under SAC No Prediction to 212.6 ms, lowers mean queue pressure from 0.1716 to 0.0173, and increases completed work from 37,070 to 172,880. The SAC-family ablation then explains why this happens. Within that ablation, guard-only SAC improves survivability, while adding prediction on top of the guard reduces mean latency from 986.4 ms to 117.0 ms and increases throughput from 149,325 to 182,145. These are not contradictory results from the same benchmark table; they answer different questions. The full comparison shows that the proposed controller outperforms the broader baseline families, whereas the ablation isolates the separate roles of the guard and the predictor inside the SAC family. The practical interpretation is straightforward: prediction provides foresight, CH-SAC learns how strongly and how early to react to that foresight, and the guard preserves safe operation when overload escalates too quickly for learned control alone to remain sufficient.

## 7.4 Cross-Platform Synthesis and Discussion

A central strength of this thesis is that the proposed framework is evaluated through two complementary experimental layers: a YAFS-based trace-driven simulation layer and a Mininet-based emulation layer. These two layers serve different methodological purposes and should therefore be interpreted together rather than viewed as redundant repetitions of the same experiment.

The YAFS-based analyses provide fine-grained visibility into controller behaviour. They make it possible to examine forecast–action coupling, proactive versus reactive scaling, proactive headroom, overload-prediction quality, spatial distribution of proactive behaviour, lead-time distributions, and the temporal alignment between mobility demand and provisioning activity. In other words, the YAFS layer explains why the controller behaves as it does. It shows that the Fusion Predictor generates meaningful short-horizon signals, that CH-SAC converts these signals into anticipatory resource adjustments, and that proactive decisions are concentrated in the nodes and time windows where mobility-driven demand is expected to rise.

The Mininet-based analyses, by contrast, provide the primary evidence of end-to-end control effectiveness under controlled stress. Because Mininet explicitly enforces bandwidth, propagation delay, queueing, and per-node compute constraints, it exposes performance differences under more realistic congestion dynamics. This layer is therefore used to evaluate the proposed framework under the three principal stress regimes of the thesis: CPU stress, queue stress, and hotspot / flash-crowd stress. The Mininet results show that the proposed framework consistently reduces latency, suppresses queue pressure, lowers drop rates, increases throughput, and recovers more quickly than the baseline policies under tightly controlled comparison conditions.

Interpreted together, the two platforms reveal a coherent picture. The YAFS analyses show that the controller is genuinely predictive rather than merely reactive, while the Mininet results show that this predictive behaviour translates into measurable system-level gains under realistic bottlenecks. The simulation layer therefore provides mechanistic validation, whereas the emulation layer provides performance validation.

The combined evidence also clarifies the distinct roles of the major components of the framework. The Fusion Predictor provides temporal and spatial foresight, allowing the controller to anticipate where and when demand will intensify. CH-SAC transforms this foresight into adaptive, continuous scaling decisions. The constraint-aware guard provides a safety backstop that stabilises the system when overload signals escalate too quickly for learned control alone to remain sufficient. This division of roles is especially visible when the results are compared across stress regimes.

Under CPU stress, the dominant challenge is delayed compute saturation. In this regime, prediction is decisive because early scaling can prevent queue growth before it becomes self-reinforcing. Under queue stress, this effect becomes even stronger: once buffers begin to saturate, recovery becomes costly and slow, so anticipation is essential. Under hotspot / flash-crowd stress, the balance shifts somewhat. Because overload signals appear abruptly when demand concentrates on a small subset of nodes, the guard becomes the dominant mechanism for immediate safety, while prediction primarily improves hotspot tracking, capacity redistribution, and stability during spatial migration.

This cross-platform interpretation also explains why the proposed framework should not be assessed using a single metric alone. The strongest evidence for its value lies not merely in lower average latency, but in the combination of behavioural and quantitative improvements observed across both evaluation layers: proactive action timing, strong forecast–action correlation, low queue pressure, fast recovery, reduced drop rate, improved QoS stability, and stronger fairness–resilience trade-offs under equal-workload comparison.

Overall, the two evaluation layers support the same central conclusion from different perspectives. The YAFS results show that the controller has learned to behave as a proactive, mobility-aware system. The Mininet results show that this behaviour is operationally effective under realistic stress. Together, they provide strong evidence that CH-SAC with Fusion Predictor shifts fog-resource management from reactive stabilisation toward anticipatory and safety-aware control.

## 7.5 Answers to the Research Questions

The research questions posed in Chapter 1 can now be answered on the basis of the theoretical design, behavioural analysis, and quantitative evaluation presented in this thesis.

**RQ1: Can short-horizon mobility- and load-aware forecasting provide operationally useful predictions for fog resource management?**

The results show that it can. The proposed Mobility- and Load-Aware Fusion Predictor combines a Flow-GRU mobility head, a Holt-style offered-load demand head, and an adaptive fusion layer to produce forecasts that are sufficiently accurate and stable for real-time control. Across the evaluation, the predictor captures the timing and direction of demand changes,

preserves visibility of offered load under congestion, and provides early warning of overload conditions. Although prediction errors remain at the most extreme peaks, the forecasts are accurate enough to support effective proactive scaling.

**RQ2: Can predictive information be integrated into reinforcement-learning-based fog control in a way that improves decision quality?**

Yes. The integration of the Fusion Predictor into CH-SAC changes the controller from a reactive policy into a forward-looking one. Predicted arrivals are incorporated into the controller's observation space, used in focus-node prioritisation, and translated into proactive scaling decisions. The resulting behaviour shows that the controller does not merely respond faster; it acts earlier, allocating resources before queue growth becomes self-reinforcing. This shift from reactive recovery to proactive congestion prevention is one of the central findings of the thesis.

**RQ3: Does the proposed framework improve performance under dynamic and stress-driven fog conditions?**

Yes. Under the three principal stress regimes examined in the thesis, CPU stress**,** queue stress**,** and hotspot / flash-crowd stress**,** the proposed framework consistently outperforms the non-predictive and heuristic baselines. It reduces latency, suppresses queue pressure, lowers packet drops, improves throughput, shortens recovery time, and strengthens QoS stability. The results therefore show that prediction-aware, constraint-aware continuous control is more effective than static, heuristic, or reactive learning-based alternatives in mobility-driven fog environments.

**RQ4: What are the respective roles of prediction, learning, and safety constraints in the final framework?**

The evaluation shows that these components play complementary roles. The Fusion Predictor provides temporal and spatial foresight. CH-SAC converts that foresight into adaptive continuous provisioning decisions. The constraint-aware guard ensures safe operation when overload emerges too rapidly for learned control alone to remain sufficient. Their relative importance depends on the operating regime: prediction is especially decisive under CPU and queue stress, whereas under flash-crowd stress the guard provides the primary safety benefit and prediction improves hotspot tracking and stability. These findings confirm that the framework is not a

simple additive combination of modules, but an integrated predictive-control system whose components reinforce one another.

In summary, the research questions of the thesis are answered positively. Short-horizon fused prediction is operationally useful, its integration into CH-SAC produces genuinely proactive control, the resulting framework performs strongly under multiple stress regimes, and the combined roles of prediction, continuous control, and guard-based safety provide a coherent and effective solution for mobility-aware fog resource provisioning.

## 7.6 Applicability of the Framework to Healthcare

Although the present study is situated in intelligent transportation and urban mobility, the prediction-enhanced SAC framework is not inherently limited to taxi traces or road traffic. Its main components, a fog-based execution environment, a short-horizon forecasting module, and a guarded continuous Soft Actor–Critic controller represent a general architecture for dynamic, latency-sensitive, and spatially distributed workloads. Healthcare is a particularly relevant extension, since remote monitoring, hospital sensor networks, telemedicine, and pre-hospital emergency care all involve non-stationary demand, geographically dispersed endpoints, and strict QoS requirements.

In a healthcare setting, the same control loop could be preserved with a different domain mapping. Data-plane events would originate from patient monitors, wearables, ambulance equipment, and clinical applications rather than vehicles. Fog nodes would correspond to ward servers, hospital edge gateways, ambulance base stations, home gateways, or regional micro-data centres. Application workflows could include vital-sign streaming, anomaly detection, early warning score computation, ECG or EEG analysis, medical image pre-processing, and telemedicine signalling. As in the transportation case, these workflows could be modelled through message types with realistic compute, memory, and bandwidth requirements.

The forecasting module would also remain conceptually applicable, but its target would shift from mobility demand to clinical workload. Instead of predicting vehicle arrivals, the predictor could estimate near-term alarm rates in an intensive care unit, the number of active monitoring streams on a ward, the likelihood of emergency admissions, or the expected imaging traffic from satellite clinics. The stream-based design used in this work short-horizon demand

estimation with lightweight online updates remains suitable for such settings because healthcare workloads are also bursty and time-varying. Only the input features and prediction model would need to be adapted, while the controller-facing outputs could remain per-node demand estimates and uncertainty indicators.

The SAC controller is similarly domain-agnostic, but its observation space and reward function would require revision. In addition to utilisation, queue length, and latency statistics, the state could include workload criticality, the mix of ICU, emergency, and general-ward traffic, and uncertainty in short-term forecasts. The reward would need to reflect clinical priorities by imposing severe penalties on delayed or missed processing of critical alarms, while allowing greater tolerance for non-urgent analytics. Existing guardrails would also need to be strengthened through hard capacity floors for life-critical services, restricted actions for essential pipelines, and safe-RL mechanisms such as constrained or shielded control.

Such an extension would be relevant to several scenarios, including smart hospitals, chronic-patient remote monitoring, and pre-hospital emergency care. However, transfer to healthcare would require stricter safety, regulatory, and ethical safeguards than those typically needed in transportation. Human oversight, auditability, privacy protection, and secure data handling would be essential, particularly where online learning affects resource allocation decisions.

Overall, the framework appears generalisable to healthcare provided that realistic workload traces are available, application graphs and SLAs are redefined around clinical priorities, and stronger safety mechanisms are incorporated. Under those conditions, the same architectural principle, a short-term predictor coupled with a guarded continuous controller, could support proactive and resilient fog-resource management for distributed healthcare workloads.

## 7.7 Limitations

Despite the positive results reported in this thesis, several limitations bound the scope of the conclusions and should be considered when interpreting the findings.

First, the workload is derived from a single real-world mobility source, namely the Beijing T-Drive dataset. Although this dataset provides realistic urban dynamics, it represents one city and one specific operating context. The proposed framework has not yet been validated across multiple cities, seasons, or application domains with substantially different mobility structures.

Second, the evaluation assumes a fixed fog topology and controlled deployment setting. The tiled fog region, placement rules, and network structure provide a reproducible and analytically useful environment, but they do not capture the full heterogeneity of real fog infrastructures, where nodes may differ substantially in compute power, connectivity, and failure behaviour.

Third, although the thesis combines YAFS simulation and Mininet-based emulation**,** it does not include a production deployment on a physical fog testbed. As a result, the reported improvements do not yet reflect practical factors such as hardware failures, monitoring noise, orchestration delays, clock drift, or deployment-level control-plane failures.

Fourth, the experiments focus on a single workload family generated from mobility-driven transport-service interactions. The proposed framework has not been evaluated under multi-tenant conditions or under application mixes such as video analytics, industrial telemetry, or safety-critical V2X traffic, which may impose different latency and compute requirements.

Fifth, the controller operates on an aggregated state representation and a restricted focus-node abstraction. This design is necessary for tractable control, but it inevitably compresses some spatial and temporal detail. Fine-grained per-flow effects, rare local anomalies, and wider long-range dependencies may therefore be only partially represented in the state.

Sixth, the controller influences only part of the end-to-end service path. While CH-SAC with Fusion Predictor can effectively optimise fog-side provisioning, it does not directly control the full network path**,** backhaul capacity, or sink-side processing. Consequently, some end-to-end latency components remain dominated by infrastructure effects outside the controller's action space.

Seventh, the evaluation compares the proposed framework with a finite set of baselines. Although this includes static, threshold-based, heuristic, meta-heuristic, and learning-based methods, it is not an exhaustive comparison against all possible control paradigms. Alternative methods, including model-predictive control or multi-agent approaches, remain outside the scope of this work.

Finally, several implementation choices, such as the guard thresholds, fusion bounds, and reward trade-offs, are engineered and tuned for stable operation rather than exhaustively optimised. The reported results therefore demonstrate a strong and practical solution, but not necessarily a globally optimal one.

These limitations do not weaken the core contribution of the thesis. Rather, they define the conditions under which the reported findings should be interpreted and motivate the future research directions outlined below.

## 7.8 Future Work

Several directions emerge naturally from the results and limitations of this thesis.

A first priority is validation on heterogeneous and larger-scale fog infrastructures**.** Future work should evaluate the framework on multi-tier topologies with non-uniform node capacities, variable link characteristics, and more complex placement constraints. This would clarify how well the proposed predictor and controller generalise beyond homogeneous tiled environments.

The second direction is multi-application and multi-tenant control. Extending the framework to jointly manage diverse workloads with competing QoS requirements would bring it closer to real operational fog systems and would allow the study of fairness, prioritisation, and resource isolation under mixed traffic.

A third direction concerns richer network-level controllability**.** In the current thesis, the controller acts mainly through fog-side scaling and capacity allocation. Incorporating routing decisions, path-aware placement, or transport-aware congestion management could further reduce end-to-end latency and better address the infrastructure-dominated delay components identified in the evaluation.

A fourth direction is uncertainty-aware and longer-horizon forecasting**.** The Fusion Predictor already produces robust short-horizon forecasts, but future work could add explicit uncertainty estimation, confidence-aware action modulation, or multi-horizon planning to further improve control under abrupt or rare demand transitions.

A fifth direction is the development of distributed or multi-agent variants of CH-SAC. As fog topologies scale, decentralised or hierarchical learning could improve scalability and responsiveness by allowing local controllers to coordinate while preserving global performance objectives.

Finally, a crucial practical step is deployment validation on a physical testbed. Even a small-scale prototype with real edge devices, live telemetry, and a real orchestration layer would

provide strong evidence of operational feasibility and would expose issues not visible in controlled simulation and emulation.

These directions would extend the present work from a validated predictive-control framework toward a broader and more deployable class of intelligent fog orchestration systems.

# CONCLUSION

This thesis addressed the problem of proactive resource provisioning in mobility-aware fog computing environments, where demand is highly dynamic, spatially shifting, and prone to sudden overload. In such settings, purely reactive control often responds only after queue growth, latency escalation, and packet drops have already emerged, making recovery slower and less effective. The central objective of this work was therefore to determine how fog-resource management can be made anticipatory, adaptive, and safe under mobility-driven demand.

To address this problem, the thesis proposed CH-SAC with a Mobility- and Load-Aware Fusion Predictor. The framework combines a Flow-GRU mobility head, a Holt-style offered-load demand head defined over accepted plus dropped requests, and an adaptive congestion-aware fusion layer with explicit floor and cap safety bounds. These predictive signals are integrated into a Constrained Hybrid Soft Actor–Critic controller, while a guard mechanism monitors queue pressure, utilisation, latency spikes, and drop ratio to preserve safe operation when overload develops too rapidly for learned control alone to remain sufficient.

The evaluation was organized in two complementary experimental layers. A trace-driven YAFS layer was used for behavioural analysis, including forecast–action coupling, proactive timing, hotspot dynamics, and prediction quality. A Mininet-based fog emulation layer was used to assess end-to-end control effectiveness under CPU-stress, queue-stress, and flash-crowd / hotspot-shift conditions. Together, these layers provided both mechanistic validation and stress-based performance validation.

The results show that the proposed framework shifts fog-resource control from reactive congestion recovery to proactive congestion prevention. In the YAFS behavioural analysis, the Fusion Predictor achieved strong short-horizon accuracy, with MAE = 3.45 taxis and Pearson correlation $r \approx 0.95$, while the controller attained a proactive-action share of 78.2%, a median lead time of 17.0 s, and an overload-prediction F1-score of 0.890. In the Mininet stress evaluation, CH-SAC with Fusion Predictor achieved 79.3 ms mean latency and 241,454 completed requests under queue stress, and 41.65 ms mean latency with 162,285 completed requests

under flash-crowd / hotspot-shift stress. Across the two evaluation layers, the framework improved proactive provisioning, latency, congestion control, service reliability, and recovery behaviour relative to threshold-based, metaheuristic, baseline RL, and non-predictive SAC baselines. These gains were especially strong under CPU- and queue-stress conditions, where early anticipation is most valuable, while under abrupt flash-crowd stress the guard provided the dominant safety benefit and prediction improved spatial adaptation and control smoothness.

A major conclusion of the thesis is that prediction, continuous-control reinforcement learning, and runtime safety constraints play distinct but complementary roles. The Fusion Predictor provides short-horizon temporal and spatial foresight. CH-SAC transforms this foresight into adaptive continuous resource-scaling decisions. The guard ensures robustness and survivability when overload evolves too quickly for predictive learned control alone to guarantee safe operation. The value of the framework therefore lies not in any single module in isolation, but in the tight integration of these components within one coherent predictive-control architecture.

The thesis also clarifies the operational boundary of fog-resource control. Once requests reach the fog-processing stage, local processing delay is only one component of total end-to-end latency, and residual delay may still be influenced by transport, routing, and infrastructure effects outside the controller's direct action space. This does not weaken the contribution of the proposed framework; rather, it defines its scope more precisely and highlights network-level coordination as an important direction for future work.

In summary, this thesis demonstrates that CH-SAC with Fusion Predictor provides an effective and practical approach to proactive fog-resource provisioning in mobility-driven urban environments. By combining fused short-horizon forecasting, constraint-aware continuous-control reinforcement learning, and runtime safety monitoring, the framework improves not only performance under stress but also stability, resilience, and operational responsiveness. More broadly, the work shows that fog-resource management can be transformed from a reactive overload-response problem into a proactive, resilient, and QoS-oriented decision-making process.

## RECOMMENDATIONS

Based on the findings of this thesis, several recommendations can be made for the design, deployment, and future study of mobility-aware fog-resource management systems.

First, proactive resource provisioning should be preferred over purely reactive scaling in mobility-driven fog environments. The results of this thesis show that short-horizon predictive signals allow resources to be allocated before queue growth becomes dominant, which is especially important under CPU-stress and queue-stress conditions.

Second, system monitoring should represent true offered load rather than admitted traffic alone. In congested conditions, dropped requests hide part of the actual demand, and controllers that observe only accepted traffic risk underestimating system pressure. Future fog-control platforms should therefore track accepted requests, dropped requests, queue pressure, tail latency, and delivery ratio together.

Third, learning-based controllers intended for SLA-sensitive fog systems should always be supported by explicit runtime safety mechanisms. A guard layer that monitors queue pressure, utilisation, latency spikes, and drop behaviour should remain active during execution so that overload can be contained even when the learned policy reacts too conservatively or when demand changes abruptly.

Fourth, resource allocation should remain spatially selective rather than uniformly distributed across all fog nodes. Because mobility-driven demand is concentrated and shifts over time, control strategies should prioritise hotspot or focus nodes and apply bounded continuous adjustments where they are most needed.

Fifth, controller evaluation should not rely on mean latency alone. Future work and practical deployments should report tail latency, queue pressure, drop rate, delivery ratio, recovery time, action volatility, and fairness-resilience behaviour, and should validate performance under CPU-stress, queue-stress, and flash-crowd or hotspot-shift scenarios.

Finally, future extensions of this work should move toward heterogeneous, multi-tenant, and network-aware fog infrastructures. In particular, integrating routing-aware control,

uncertainty-aware forecasting, distributed or multi-agent coordination, and physical testbed validation would improve deployment realism and strengthen end-to-end QoS performance.

## APPENDIX A

## YAFS PSEUDOCODE

Algorithm-A I-1

YAFS trace-driven predictive control loop

```
Input:
    T-Drive mobility traces
    YAFS tiled fog topology
    application/service deployment
    Fusion Predictor
    CH-SAC controller
    Guard thresholds
    decision interval Δt
1: Load config and experiment parameters
2: Parse T-Drive traces into timestamped mobility events
3: Map taxi positions to fog tiles / nodes
4: Build YAFS topology and ensure connectivity
5: Deploy application services and initialize router + placement policy
6: Initialize Fusion Predictor, CH-SAC, replay buffer, logs
7: for simulation time t = 0 to T_end do
8:     mobility_events ← events arriving at time t
9:     update dynamic taxi gateways from mobility_events
10:    for each active taxi gateway do
11: Generate application message(s)
 12:        route message via AdvancedShortestPath
13:        execute through service module chain
14:        accumulate communication + queue + processing delay
15:        record completion or drop
16:    end for
17:    if t mod Δt = 0 then
18:        telemetry ← collect system metrics
              (latency, queue pressure, utilization, throughput, drops)
 19:        mobility_features ← derive
              (taxi counts, arrivals, departures, neighbour flows)
 20:        offered_load ← accepted_requests + dropped_requests
21:        prediction ← FusionPredictor.update(
              telemetry, mobility_features, offered_load)
22:        focus_nodes ← select nodes with highest current/predicted pres-
sure
23:        state ← build CH-SAC state from
              telemetry + mobility_features + prediction + focus_nodes
24:        action_raw ← CH-SAC.actor(state)
25:        action_safe ← Guard(action_raw, telemetry, safety_limits)
26:        apply action_safe through YAFS orchestration / placement layer
              (service-capacity adjustment)
```

```
27:        reward ← compute QoS/resource/stability reward
28:        store transition in replay buffer
29:        update actor/critic asynchronously
30:    end if
31:    append step logs
32: end for
33: continue short drain period for in-flight requests and queue tail
34: post-process logs into latency, queue pressure, throughput, drops, SLA,
recovery
35: generate figures
```

## APPENDIX B

## MININET PSEUDOCODE

Algorithm-A II-1

Mininet emulation predictive control loop

```
Input:
    T-Drive mobility traces
    Mininet graph from config.ini
    TCLink network settings
    fog host service parameters
      (workers, queue size, processing time, metrics interval)
    Fusion Predictor
    CH-SAC controller
    Guard thresholds
    decision interval Δt

1: Load Mininet configuration and graph
2: Create one switch-host pair per logical fog node
3: Create links with TCLink bandwidth/delay settings
4: Start Open vSwitch bridges in standalone mode
5: Launch fog-processing service on each host
6: Parse T-Drive traces into timestamped events
7: Map each event to nearest fog node
8: Initialize controller, predictor, logs
9: for each discrete time step t do
10:    requests_t ← mobility events scheduled for time t
11:    for each request r in requests_t do
12:        inject r at mapped node / gateway
13:        transmit r through Mininet network path
14:        deliver r to destination fog host
15:        if host queue is full then
16:            mark drop
17:        else
18:            enqueue r
19:            worker thread processes r for configured service time
20:            mark completion at sink
21:        end if
22:    end for
23:    if t mod Δt = 0 then
24:        telemetry ← collect host metrics
              (latency, queue pressure, utilization, arrivals, drops)
25:        mobility_features ← derive from mapped event stream
26:        offered_load ← accepted_requests + dropped_requests
27:        prediction ← FusionPredictor.update(
              telemetry, mobility_features, offered_load)
28:        focus_nodes ← select highest-pressure nodes
```

```
        state ← build state from telemetry + mobility + prediction
        action_raw ← CH-SAC.actor(state)
        action_safe ← Guard(action_raw, telemetry, safety_limits)
        enforce action_safe on hosts
          (workers / CPU-related capacity / queue-related controls)
        reward ← compute reward from QoS + efficiency + stability
Log decision and system state
    end if
end for
allow final drain interval
aggregate logs into throughput, latency, drop rate, delivery ratio, SLA, re-
covery
generate evaluation figures
```

# APPENDIX C

## LIST OF PUBLICATIONS

The following publications related to this thesis have been officially accepted:

**Beigali, S. and Kadoch, M. "Prediction-Enhanced Soft Actor–Critic for Proactive Mobility-Aware Resource Provisioning in Fog Computing."** Officially accepted by the International Symposium on Intelligent Computing and Networking 2026 (ISICN 2026). To be published by Springer and submitted to EI for indexing, 2026.

**Beigali, S. and Kadoch, M. "A Congestion-Aware Fusion Predictor for Proactive Fog Resource Provisioning Under Mobility Shifts and Flash Crowds."** Officially accepted by the International Symposium on Intelligent Computing and Networking 2026 (ISICN 2026). To be published by Springer and submitted to EI for indexing, 2026.

## LIST OF REFERENCES


Abdelghany, H. M. (2025). Dynamic Resource Management and Task Offloading Framework for Fog Computing. *Journal of Grid Computing*, *23*(2), 19. https://doi.org/10.1007/s10723-025-09804-7

Alsadie, D. (2024). A Comprehensive Review of AI Techniques for Resource Management in Fog Computing: Trends, Challenges, and Future Directions. *IEEE Access*, *12*, 118007–118059. https://doi.org/10.1109/ACCESS.2024.3447097

Bachiega, J., Costa, B., Carvalho, L. R., Rosa, M. J. F., & Araujo, A. (2023). Computational Resource Allocation in Fog Computing: A Comprehensive Survey. *ACM Computing Surveys*, *55*(14s), 1–31. https://doi.org/10.1145/3586181

Behera, S. R., Panigrahi, N., Bhoi, S. K., Sahoo, K. S., Jhanjhi, N. Z., & Ghoniem, R. M. (2023). Time Series-Based Edge Resource Prediction and Parallel Optimal Task Allocation in Mobile Edge Computing Environment. *Processes*, *11*(4), 1017. https://doi.org/10.3390/pr11041017

Canali, C., & Lancellotti, R. (2019). GASP: Genetic Algorithms for Service Placement in Fog Computing Systems. *Algorithms*, *12*(10), 201. https://doi.org/10.3390/a12100201

Ghafari, R., & Mansouri, N. (2025). Reinforcement learning-based solution for resource management in fog computing: A comprehensive survey. *Expert Systems with Applications*, *276*, 127214. https://doi.org/10.1016/j.eswa.2025.127214

Ghobaei-Arani, M., Souri, A., & Rahmanian, A. A. (2020). Resource Management Approaches in Fog Computing: A Comprehensive Review. *Journal of Grid Computing*, *18*(1), 1–42. https://doi.org/10.1007/s10723-019-09491-1

Goswami, A., Modi, K., & Patel, C. (2025). Evaluation of Optimization Algorithm for Application Placement Problem in Fog Computing: A Systematic Review. *Archives of Computational Methods in Engineering*. https://doi.org/10.1007/s11831-025-10227-6

Lera, I., & Guerrero, C. (2024). Multi-objective application placement in fog computing using graph neural network-based reinforcement learning. *The Journal of Supercomputing*, *80*(19), 27073–27094. https://doi.org/10.1007/s11227-024-06439-5

Nieto, G., de la Iglesia, I., Lopez-Novoa, U., & Perfecto, C. (2024). Deep Reinforcement Learning techniques for dynamic task offloading in the 5G edge-cloud continuum. *Journal of Cloud Computing*, *13*(1), 94. https://doi.org/10.1186/s13677-024-00658-0

Sarrafzade, N., Entezari-Maleki, R., & Sousa, L. (2022). A genetic-based approach for service placement in fog computing. *The Journal of Supercomputing*, *78*(8), 10854–10875. https://doi.org/10.1007/s11227-021-04254-w

Tang, M., & Wong, V. W. S. (2022). Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Transactions on Mobile Computing*, *21*(6), 1985–1997. https://doi.org/10.1109/TMC.2020.3036871