

UNIVERSITY OF PIRAEUS
DEPARTMENT OF DIGITAL SYSTEMS
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGIES

AI-Driven Resource and Service Management in the
Cloud-Edge Computing Continuum, with Emphasis
on Resource Optimization and Quality of Experience.

Efterpi G. Paraskevoulakou

Doctoral Thesis

Piraeus, 2026

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

Διαχείριση πόρων και υπηρεσιών βασισμένη στην
τεχνητή νοημοσύνη στο
νεφροϋπολογιστικό-αχροδικτυακό περιβάλλον, με
έμφαση στη βελτιστοποίηση πόρων και ποιότητας
εμπειρίας.

Ευτέρπη Γ. Παρασκευουλάκου

Διδακτορική Διατριβή

Πειραιάς, 2026

UNIVERSITY OF PIRAEUS

DEPARTMENT OF DIGITAL SYSTEMS

**AI-Driven Resource and Service Management
in the Cloud-Edge Computing Continuum
with Emphasis on Resource Optimization and Quality of Experience**

Doctoral Thesis Presented

by **Efterpi Paraskevoulakou**

in Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

ADVISORY COMMITTEE:

Professor Dimosthenis Kyriazis

Professor Christos Doulkeridis

Assistant Professor Orestis Telelis

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Διαχείριση πόρων και υπηρεσιών βασισμένη στην τεχνητή νοημοσύνη στο
νεφοϋπολογιστικό-ακροδικτυακό περιβάλλον, με έμφαση στη
βελτιστοποίηση πόρων και ποιότητας εμπειρίας.

Διδακτορική Διατριβή

που υποβλήθηκε από την Ευτέρπη Γ. Παρασκευουλάκου

για την εκπλήρωση των απαιτήσεων

απόκτησης Διδακτορικού Διπλώματος

ΣΥΜΒΟΥΛΕΥΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Καθηγητής Δημοσθένης Κυριαζής

Καθηγητής Χρήστος Δουλκερίδης

Επίκουρος Καθηγητής Ορέστης Τελέλης

UNIVERSITY OF PIRAEUS
DEPARTMENT OF DIGITAL SYSTEMS

**AI-Driven Resource and Service Management in the Cloud-Edge Computing
Continuum, with Emphasis on Resource Optimization and Quality of
Experience.**

Doctoral Thesis Presented
by **Efterpi G. Paraskevoulakou**
in Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

APPROVED BY:

Dimosthenis Kyriazis, Professor, University of Piraeus

Christos Doulkeridis, Professor, University of Piraeus

Orestis Telelis, Assistant Professor, University of Piraeus

Andriana Prentza, Professor, University of Piraeus

Michail Filippakis, Professor, University of Piraeus

Andreas Menychtas, Assistant Professor, University of Piraeus

Georgios Kousiouris, Associate Professor, Harokopio University of Athens

Piraeus, 2026

To my grandfather Giorgos.

Acknowledgments

The PhD journey is not an easy task. I could never have imagined that I would be at this point, writing down the research I have carried out over all these years. I can confirm that a PhD is not for everyone, and I say this not because of the level of intelligence required.

Beginning a PhD is easy, finishing one requires a robust character, a willingness to sacrifice social life and ordinary human activities, and an unwavering commitment to see it through. Holding a PhD degree gives you something unique, something beyond the technical expertise presented in these pages. The most profound thing I have gained from this journey is something no algorithm can quantify: wisdom. Wisdom, and the ability to think in the right direction — not only within the domain of my expertise, but also in how to face challenging problems with patience, strong will, and the determination never to give up.

I am deeply grateful to my supervisor, Professor Dimosthenis Kyriazis, for giving me this priceless gift: the resilience to find direction independently, navigating the complexity and uncertainty that research demands without a guiding hand. I truly appreciate him for being such a tremendous mentor, and I am very happy to have him as part of my life.

I would also like to thank Professor Michael Filippakis for supporting me throughout these years and empowering me with the will to complete my PhD. It has been an honour to know him.

Special thanks go to my best friend and colleague Aggeliki Panagopoulou, who always knew exactly how I was feeling on those days when the pressure was at its highest. Her support was invaluable throughout this journey, and I truly owe her a

great deal. Her moral support and the flexibility she offered me in my professional life were crucial in allowing me to complete this research.

I would not forget my parents, Giorgos and Georgia. I am very lucky to have such loving and supportive parents, who stood by me every step of the way.

This thesis is devoted, above all, to my grandfather, who left this world in 2022. His absence created a void that words cannot fill, but his memory became the quiet force that drove me forward. In my hardest moments (especially when I submitted the second revision of my first journal), the thought of him — and of what I wanted to prove, to everyone, to him, and above all to myself — gave me the strength to continue and to finish what I had started. I hope, wherever he is, he is proud.

Grandfather, this one is for you.

Abstract

Keywords: edge computing, service placement, AI-driven resource management, Quality of Service (QoS), Quality of Experience (QoE), serverless computing, functions-as-a-service, incremental learning, federated learning, reinforcement learning, artificial intelligence, multi-objective optimization

Edge computing has emerged as a promising solution for delivering services that demand low latency, high bandwidth, and stringent privacy requirements across numerous data- and compute-intensive applications. A major challenge in the edge computing paradigm is service placement, where the objective is to dynamically select edge nodes for deploying and executing services, thereby improving application Quality-of-Service (QoS) and maximizing resource utilization. Nevertheless, the heterogeneous nature of edge computing resources, diverse application requirements, limited computational resource capacity, stringent user demands, and the overall dynamically changing conditions of the ecosystem constitute a multi-objective optimization problem, resulting in the need for adaptive optimization techniques. On the other hand, Artificial Intelligence (AI) has emerged as a revolutionary paradigm, exploiting data to learn hidden patterns and complex relationships, producing recommendations and predictions for the future behavior of the domain to which it is applied. This thesis investigates the dynamic service placement problem in edge-cloud continuum applications, considering the heterogeneity of the edge system, limited computing resources, and

user satisfaction levels, with the goal of dynamically providing optimal placement schemes for services running across the edge-cloud continuum. Two main approaches are introduced: service placement considering the service itself, and service placement considering the edge resources. The first method is an AI-based approach that leverages historical telemetry data derived from services deployed in the edge ecosystem under the serverless paradigm, recommending the optimal chain of functions-as-a-service to minimize service response time. The second method presents a decentralized, AI-driven three-tier solution comprising three complementary learning-based techniques — Incremental Learning (IL), Federated Learning (FL), and Reinforcement Learning (RL) — that collectively consider edge resources and diverse, dynamic user demands to place services in a manner that balances the trade-off between computational capacity and user satisfaction. The results demonstrate that both methods can effectively determine optimal service placement schemes in a dynamic manner, achieving significant improvements in resource utilization, service response time, and user Quality-of-Experience (QoE) compared to baseline standalone approaches, while maintaining adaptability under varying workload conditions and heterogeneous infrastructure configurations. This thesis also provides potential directions for future research that could build upon the findings of the current thesis.

Περίληψη

Λέξεις-κλειδιά: ακροδικτυακή υπολογιστική, δυναμική κατανομή υπηρεσιών, διαχείριση πόρων βασισμένη σε τεχνητή νοημοσύνη, ποιότητα Υπηρεσιών, Ποιότητα Εμπειρίας, μοντέλο “χωρίς διακομιστή”, λειτουργία-ως-υπηρεσία, αυξητική μάθηση, ομοσπονδιακή μάθηση, ενισχυτική μάθηση, τεχνητή νοημοσύνη, πολυ-κριτηριακή βελτιστοποίηση

Η παρούσα διδακτορική διατριβή διερευνά το πρόβλημα της δυναμικής κατανομής υπηρεσιών σε ακροδικτυακές-νεφροϋπολογιστικές υποδομές (Edge Computing service placement), λαμβάνοντας υπόψη την ετερογένεια του συστήματος, τους περιορισμένους υπολογιστικούς πόρους και τον βαθμό ικανοποίησης χρηστών, με στόχο την παροχή βέλτιστων σχημάτων τοποθέτησης υπηρεσιών σε δυναμικό περιβάλλον. Στην παρούσα διατριβή εισάγονται δύο προσεγγίσεις βασισμένες στη Τεχνητή Νοημοσύνη: (α) κατανομή υπηρεσιών προσεγγίζοντας την ίδια την υπηρεσία και (β) κατανομή υπηρεσιών προσεγγίζοντας τους ακροδικτυακούς πόρους. Η πρώτη μέθοδος αποτελεί μια προσέγγιση που αξιοποιεί ιστορικά δεδομένα φόρτου εργασίας της υποδομής από υπηρεσίες αναπτυγμένες στο ακροδικτυακό οικοσύστημα με χρήση του μοντέλου “χωρίς-διακομιστή” (serverless), προτείνοντας το καλύτερο συνδυασμό συνόλων “λειτουργίας-ως-υπηρεσία” (functions-as-a-service) με σκοπό την ελαχιστοποίηση του χρόνου απόκρισης ως προς την απάντηση του τελικού χρήστη. Η δεύτερη μέθοδος παρουσιάζει μια ολοκληρωμένη αποκεντρωμένη, τριών ε-

πιπέδων λύση, που συνδυάζει τρεις τεχνικές μάθησης — αυξητική μάθηση (Incremental Learning), Ομοσπονδιακή Μάθηση (Federated Learning) και Ενισχυτική Μάθηση (Reinforcement Learning) — οι οποίες λαμβάνουν υπόψη τους ακροδικτυακούς πόρους και τις ποικίλες, δυναμικές απαιτήσεις χρηστών, τοποθετώντας τις ακροδικτυακές υπηρεσίες με τρόπο που εξισορροπεί τους περιορισμένους υπολογιστικούς πόρους της ακροδικτυακής υποδομής και την ικανοποίηση των χρηστών ως προς την απόκριση. Τα αποτελέσματα δείχνουν ότι και οι δύο μέθοδοι μπορούν αποτελεσματικά να προσδιορίσουν βέλτιστα σχήματα κατανομής υπηρεσιών με δυναμικό τρόπο, επιτυγχάνοντας σημαντικές βελτιώσεις στην αξιοποίηση πόρων, τον χρόνο απόκρισης υπηρεσιών και την Ποιότητα Εμπειρίας των χρηστών σε σχέση με μεμονωμένες προσεγγίσεις, διατηρώντας παράλληλα προσαρμοστικότητα υπό μεταβαλλόμενες συνθήκες φόρτου εργασίας και ετερογενείς υποδομές. Η διατριβή παρέχει επίσης κατευθύνσεις για μελλοντική έρευνα που θα μπορούσε να οικοδομηθεί πάνω στα ευρήματα της παρούσας εργασίας.

Table of Contents

Acknowledgments	xv
Abstract	xvii
Περίληψη (Abstract in Greek)	xix
Table of Contents	xxi
List of Figures	xxvii
List of Tables	xxxiii
Acronyms	xxxvii
1 Introduction	1
1.1 Motivation	8
1.2 Objectives	10
1.3 Knowledge gap and research questions	11
1.4 Research Contributions	16
1.5 Outline of dissertation	19
2 Background: edge-cloud continuum and service placement	23
2.1 Edge computing paradigm	24
2.1.1 Multi-access Edge Computing (MEC)	25
2.1.2 Fog Computing	27
2.1.3 Cloudlet Computing	27
2.1.4 Comparative analysis and suitability for resource-constrained edge deployments	28
2.2 Cloud-to-edge continuum architecture	29
2.3 Service placement fundamentals	30
2.4 Orchestration frameworks and technologies	37

2.4.1	Containers as the ultimate deployment unit	38
2.4.2	Kubernetes and its edge-oriented derivatives	39
2.4.3	Service meshes for microservice management	40
2.4.4	ETSI MEC orchestration framework	40
2.4.5	Infrastructure monitoring and observability	41
2.4.6	The continuum gap	42
2.5	Optimization in decentralized systems	43
2.5.1	Classical optimization methods and their limitations	44
2.5.2	Machine learning for workload prediction	45
2.5.3	Federated learning for privacy-preserving collaborative opti- mization	46
2.5.4	Reinforcement learning for sequential placement decisions	47
2.5.5	Serverless computing and FaaS as a placement abstraction .	48
2.5.6	Shifting towards closed-loop AI-driven orchestration	49
3	Research Problem and Challenges	51
3.1	Problem definition	52
3.1.1	Infrastructure model	53
3.1.2	Application model	54
3.1.3	Deployment pattern and constraint taxonomy	55
3.1.4	Placement Strategy Taxonomy	56
3.1.5	The two dimensions of the placement problem	57
3.2	General formulation of the service placement problem	58
3.2.1	Notation and definitions	58
3.2.2	Placement function	59
3.2.3	Constraint formulation	59
3.2.4	Objective function	60
3.2.5	Complexity and resolution strategies	62
3.3	Resource heterogeneity and scalability	63
3.4	Latency and Quality of Service (QoS) constraints	65

3.5	Dynamic and uncertain environments	67
4	Related Work	71
4.1	Serverless computing and ML/DL workload deployment	72
4.1.1	Machine learning as a serverless service	73
4.1.2	Serverless frameworks for ML/DL pipelines	73
4.1.3	Edge Artificial Intelligence (AI) and serverless architectures	75
4.2	Application placement in edge–cloud environments	77
4.2.1	Orchestration mechanisms	77
4.2.2	Application placement strategies	79
4.2.3	Approaches to service placement by using the serverless paradigm	81
4.3	Summary and research gaps	82
5	An AI-driven service-centric approach to dynamic service placement via the serverless paradigm	85
5.1	Motivation and service-centric perspective	87
5.2	Proposed approach: ML-FaaS	93
5.2.1	Offline phase	96
5.2.2	Online phase	103
5.2.3	Overcoming serverless constraints via container-extended run-times	105
5.3	AI-based function regulator for approaching the service placement problem	106
5.3.1	Problem formulation	106
5.3.2	Proposed architecture	107
5.4	Evaluation	112
5.4.1	Experimental setup	112
5.4.2	Offline phase evaluation results	113
5.4.3	Online phase and pipeline performance	115
5.4.4	Function partitioning and regulator results	118

5.4.5	Development effort estimation via the constructive cost model	119
5.5	Summary	122
6	A Closed-loop, three-tier AI-driven orchestration approach for dynamic resource-centric service placement	125
6.1	Motivation and resource-centric perspective	126
6.2	System model and problem formulation	128
6.2.1	Optimization problem	130
6.3	Core functionality of the three-layer AI-based approach	131
6.3.1	Conceptual Architecture	131
6.3.2	Data Collection and Analysis	134
6.3.3	Training phase	139
6.3.4	Incremental learning layer	140
6.3.5	Federated learning layer	141
6.3.6	Reinforcement learning layer	142
6.4	Evaluation	146
6.4.1	Experimental setup	146
6.4.2	Training models comparison	147
6.4.3	Federated learning results	150
6.4.4	Resource overhead of the decentralized AI agents	151
6.4.5	Reinforcement learning results	152
6.4.6	Placement inference results	154
6.4.7	Latency–resource utilization trade-off analysis	160
6.5	Summary	161
7	Conclusions and Future Directions	165
7.1	Critical reflection on the proposed approaches	166
7.1.1	The service-centric approach	166
7.1.2	The resource-centric approach	167
7.2	Synthesis of contributions	169
7.3	Future research directions	172

7.3.1	Extensions of the service-centric approach	173
7.3.2	Extensions of the resource-centric approach	174
7.3.3	Integration and joint optimization	175
7.3.4	Broader research challenges	176
	Bibliography	179

List of Figures

5.1	Overview of the proposed two-phase Machine Learning (ML)-Function as a Service (FaaS) architecture, illustrating the offline data analytics and model training phase, and the online serverless function pipeline realized on Apache OpenWhisk, with container-extended runtimes overcoming platform code-size constraints.	95
5.2	Proposed data analytics exploitation path for the offline phase of the ML-FaaS approach, depicting the sequential progression from dataset collection and Exploratory Data Analysis (EDA) through dimensionality reduction, feature engineering, data enrichment, model training, and evaluation, culminating in the selection of the Keras Multilayer Perceptron (MLP) as the production inference model. . .	96
5.3	Class distribution of the fraud detection dataset, illustrating the severe imbalance between legitimate and fraudulent e-commerce transactions, with fraudulent observations comprising only 3.5% of the total dataset.	98
5.4	Nullity matrix visualizing the patterns of missing values across a representative sample of the dataset's features. Grey shading indicates data presence, and white indicates absence. The high proportion of missing values across multiple feature columns necessitates systematic imputation before model training.	99

5.5 Distribution of fraudulent transactions across weekdays, revealing a statistically elevated concentration of fraud on Saturday, Sunday, and Friday. This temporal pattern is attributable to the reduced responsiveness of financial institutions and law enforcement during weekend periods, and motivates the extraction of weekday-derived features during the feature engineering step. 100

5.6 Response time distributions across the four candidate pipeline configurations (2, 3, 4, and 5 functions), illustrating the non-monotonic relationship between pipeline cardinality and end-to-end response time. The 5-function pipeline is excluded from further analysis due to response time divergence. 108

5.7 Conceptual architecture of the AI-based Function Regulator, comprising the Staging Distribution Process, the Hyperparameter Regulator, and the Trainer/Orchestrator components. 109

5.8 Classification performance of the evaluated ML/Deep Learning (DL) models across data augmentation strategies, measured in terms of recall (sensitivity) and Area Under the Curve (AUC). 116

5.9 Average per-action and end-to-end pipeline response time across 10 re-executions of the 3-function ML-FaaS pipeline under warm execution conditions. Sub-figures show: (a) first function — data transformation and imputation, (b) second function — normalisation, (c) third function — ML/DL model inference, (d) full end-to-end pipeline. The cold-start invocation (iteration 1, orange bar) is annotated separately in each panel. The dashed line indicates the average warm execution time across iterations 2–10. 117

6.1	Conceptual architecture of the proposed three-layer AI-driven orchestration mechanism across the edge–cloud continuum, illustrating the Users Layer, Infrastructure Layer, and Control Plane with the Incremental Learning (IL), Federated Learning (FL), and Reinforcement Learning (RL) components.	132
6.2	Complete pipeline flow with threshold-based escalation across the IL, FL, and RL layers, illustrating the event-driven activation conditions governing layer transitions.	133
6.3	Pairwise scatter plot matrix of the five normalized telemetry metrics collected from edge nodes during application runtime: Central Processing Unit (CPU) usage, memory usage, normalized network receive rate (<i>Net_RX_norm</i>), normalized network transmit rate (<i>Net_TX_norm</i>), and normalised system load (<i>Load_1m_norm</i>). Diagonal panels show the marginal distribution of each metric, off-diagonal panels show bivariate scatter plots between metric pairs. The absence of discernible linear structure across all pairs confirms the presence of complex non-linear temporal dependencies, motivating the adoption of neural network-based forecasting architectures.	138
6.4	Correlation heatmap of the collected telemetry metrics, revealing low or near-zero pairwise correlations indicative of complex non-linear temporal dependencies in edge node resource dynamics.	140
6.5	Evolution of Root Mean Square Error (RMSE), Mean Squared Error (MSE), and R^2 for CPU resource prediction across successive iterations during the incremental learning process, including the trigger points of the federated aggregation mechanism.	148
6.6	Evolution of RMSE, MSE, and R^2 for memory resource prediction across successive iterations during the incremental learning process, including the trigger points of the federated aggregation mechanism.	148

6.7	Evolution of MSE for CPU resource prediction across successive iterations, with indication of the federated aggregation trigger points and the corresponding post-aggregation error correction behavior.	150
6.8	Evolution of MSE for memory resource prediction across successive iterations, with indication of the federated aggregation trigger points. The post-aggregation stabilisation of RMSE values confirms the positive impact of the FL mechanism on prediction accuracy and robustness under non-stationary workload conditions.	151
6.9	Reward convergence comparison (left) and normalized loss evolution (right) of the evaluated RL algorithms over 10,000 training episodes.	153
6.10	Comparison of RL models in terms of success rate (percentage of episodes exceeding a reward threshold of 1,000) and correct scaling decisions.	153
6.11	Radar chart comparing all evaluated models — Random Forest, Decision Trees, Logistic Regression, Naïve Bayes, MLP, and PPO — across the complete set of performance metrics encompassing precision, recall, and F-measure (F1)-score for all placement action categories and overall accuracy. The PPO polygon consistently occupies the outermost region across all axes, confirming its uniformly superior performance.	156
6.12	Recall comparison across all evaluated models by placement action category: migrate_node_0, migrate_node_1, migrate_node_2, and scale. PPO achieves perfect or near-perfect recall across all categories, confirming its superior sensitivity to the full spectrum of placement decision requirements.	157

6.13	Precision comparison across all evaluated models by placement action category: <code>migrate_node_0</code> , <code>migrate_node_1</code> , <code>migrate_node_2</code> , and <code>scale</code> . PPO achieves perfect or near-perfect precision across all migration categories. The zero precision of Random Forest for <code>migrate_node_1</code> highlights the systematic generalization failures of static classifiers under heterogeneous placement conditions.	158
6.14	Quantitative evaluation of the PPO model's placement decisions: (left) proportion of cases where the geographically closest node was not selected and the reasoning behind such decisions, (centre) migration versus scaling decision accuracy, (right) distribution of resource load reduction Δ for non-closest-node selections.	160

List of Tables

1.1	List of publications related to the contributions of this thesis.	18
2.1	Comparative overview of edge computing paradigms.	29
2.2	Comparative overview of Cloud Computing and Edge Computing. .	31
4.1	Consolidated comparison of related work across the serverless ML/DL deployment and edge–cloud application placement dimensions. Limitations are assessed relative to the research gaps addressed by the contributions of Chapters 5 and 6.	84
5.1	Evaluation results of the offline phase across ML/DL model families and data augmentation strategies. The best-performing configuration is highlighted in bold.	115
5.2	Average response time of the ML-FaaS pipeline tested under two hardware configurations, demonstrating the independence of response time from allocated computational resources.	118
5.3	Benchmarking of different task groupings during the function partitioning procedure. Average response time is reported over 30 executions per configuration.	119
5.4	Forecasting performance of the evaluated DL architectures (Long Short-Term Memory (LSTM), Bi-directional LSTM, CNN) in terms of RMSE across the three candidate pipeline configurations, together with the optimal hyperparameter combinations identified by the Hyperparameter Regulator.	120

6.1	Representative sample of telemetry data collected from edge nodes during application runtime, illustrating the five normalized metrics used as inputs to the IL forecasting models	136
6.2	Average MSE, RMSE, and R^2 across three independent experimental runs and all worker nodes, for each evaluated DL architecture. . . .	148
6.3	Comparison of the attention-enhanced LSTM and GRU architectures across key performance indicators for resource-constrained edge deployment.	149
6.4	Training configuration of the PPO-based RL agent as implemented in the proposed orchestration approach.	154
6.5	Representative subset of the 100 evaluation scenarios, with node resource metrics, contextual information, and ground-truth placement labels.	154
6.6	Performance comparison across supervised ML models, classical heuristic baselines, and the PPO-based RL agent across all placement decision categories.	155
6.7	Latency–resource trade-off performance comparison between the PPO-based RL agent and classical heuristic placement strategies. Lower values of J indicate better joint optimization of latency and resource efficiency.	161

List of Algorithms

1	AI-based Function Regulator for optimal serverless pipeline composition	110
2	Three-layer AI-driven orchestration pipeline for addressing the service placement problem	134
3	Event-driven RL-based edge application placement layer	146

Acronyms

6G Sixth Generation Mobile Network

ACO Ant Colony Optimization

AI Artificial Intelligence

ALS Amyotrophic Lateral Sclerosis

ANN Artificial Neural Network

API Application Programming Interface

AUC Area Under the Curve

AWS Amazon Web Services

BD Big Data

CDN Content Delivery Network

CNN Convolutional Neural Network

COCOMO Constructive Cost Model

CPLX Product Complexity

CPS Cyber-Physical System

CPU Central Processing Unit

DAG Directed Acyclic Graph

DL Deep Learning

DQN Deep Q-Network

DRL Deep Reinforcement Learning

DS Data Science

DSML Domain-Specific Modelling Language

EDA Exploratory Data Analysis

ETSI European Telecommunications Standards Institute

EWC Elastic Weight Consolidation

F1 F-measure

FaaS Function as a Service

FL Federated Learning

FPGA Field-Programmable Gate Array

GA Genetic Algorithm

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

IF Impact Factor

IIoT Industrial Internet of Things

IL Incremental Learning

ILP Integer Linear Programming

IoE Internet of Everything

IoT Internet of Things

LAN Local Area Network

LLM Large Language Model

LSTM Long Short-Term Memory

MAAC Multi-Agent Actor-Critic

MARL Multi-Agent Reinforcement Learning

MDP Markov Decision Process

MEC Multi-access Edge Computing

ML Machine Learning

MLP Multilayer Perceptron

MRI Magnetic Resonance Imaging

MSE Mean Squared Error

PCA Principal Component Analysis

PM person-months

PPO Proximal Policy Optimization

PSO Particle Swarm Optimization

QoE Quality of Experience

QoS Quality of Service

RAM Random Access Memory

RL Reinforcement Learning

RMSE Root Mean Square Error

SA Simulated Annealing

SLOC source lines of code

SMOTE Synthetic Minority Oversampling Technique

SPP Service Placement Problem

VM Virtual Machine

WAN Wide Area Network

Chapter 1

Introduction

Chapter structure

This Chapter is constructed as follows:

- **Section 1.1 - Motivation**, This section outlines the rationale for this research and discusses its relevance and potential impact.
- **Section 1.2 - Objectives**, outlines the goals and objectives of the research.
- **Section 1.3 - Research questions**, presents the principal questions that the study seeks to address.
- **Section 1.4 - Research contributions**, explains the contributions made by the research.
- **Section 1.5 - Outline of dissertation**, presents a structural overview of the dissertation.

The rise of the edge computing paradigm emerged as a response to the inherent limitations of traditional cloud computing, which relies on centralised data centres to handle all computation and storage, requiring the transfer of large volumes of data across the network and resulting in increased latency and reduced responsiveness [Cao et al., 2020]. In this context, edge computing has been formally de-

defined as a paradigm in which substantial computing and storage resources, variously referred to as cloudlets, micro datacenters, or fog nodes, are deployed at the periphery of the network, in proximity to mobile devices or sensors, to minimise the communication overhead associated with centralised processing [Satyanarayanan, 2017].

Although edge computing represents a transformative paradigm that brings computation closer to end users, it does not replace the cloud computing model. Rather, from the perspectives of network infrastructure, business operations, application deployment, and intelligent processing, the two paradigms are intended to coexist, complement one another, and evolve in a coordinated manner, thereby contributing more effectively to the digital transformation of industries.

The foundations of this shift towards distributed computation can be traced back to the late 1990s, when the rapid growth of the internet exposed critical limitations in centralised content delivery architectures. Content Delivery Network (CDN)s emerged during this period as a response to internet congestion, with Akamai founded in 1998 by MIT researchers, pioneering the concept of distributing content across geographically dispersed servers positioned closer to end users, rather than relying on a single central server. This foundational principle of proximity-based content delivery laid the groundwork for the later evolution of edge computing. Over time, CDNs expanded beyond the caching of static assets such as images and scripts, progressively incorporating video delivery, dynamic content handling, and security functionalities [Douch et al., 2022].

Edge computing encompasses several architectural paradigms, each addressing specific deployment contexts and requirements which are explicitly described in Chapter 2. **Multi-access Edge Computing (MEC)**, formerly known as Mobile Edge Computing, was standardized by the European Telecommunications Standards Institute (ETSI) [Hu et al., 2015], which defined five key characteristics: on-premises isolation, proximity, low latency, location awareness, and network context infor-

mation. These properties extend earlier Edge Computing architectures [Ghaseminya et al., 2025] by incorporating wireless access and heterogeneous virtualization technologies. **Fog Computing**, [Yi et al., 2015] as a decentralized paradigm in which heterogeneous, ubiquitous devices communicate and cooperate to perform storage and processing tasks without third-party intervention, emerged as an extension of Cloud Computing to support the proliferation of Internet of Things (IoT) devices and real-time, low-latency applications [Peter, 2015]. **Cloudlet Computing** [Satyanarayanan et al., 2009], represents a lightweight, proximity-based cloud extension that bridges the gap between resource-constrained mobile devices and distant cloud infrastructure, by enabling computational offloading through dynamically formed, virtualized local resources [Bahtovski and Gusev, 2014].

Currently, edge computing is strongly associated with the IoT, in which billions or even trillions of network-connected devices, situated at the periphery of the Internet, enable innovative applications that are transforming numerous aspects of both traditional industrial production and everyday life. These edge IoT devices encompass a wide variety of sensors and embedded chips with diverse capabilities, produced by a broad range of manufacturers, and serve as the foundation for a growing number of applications in domains such as smart homes, smart healthcare, smart transportation, smart buildings, and smart cities. [Pan and McElhannon, 2018]

The IoT has progressively evolved alongside the shift towards the digital and automated era, giving rise to a broader conceptualisation known as the Internet of Everything (IoE), which extends beyond mere device-to-device connectivity to encompass the interconnection of people, processes, data, and things. It is precisely this richer notion of connectivity that lies at the heart of the IoE and constitutes its fundamental source of value [Miraz et al., 2015]. The widespread proliferation of smart devices and their exponential growth in connectivity have consequently led to an unprecedented explosion in data generation [Weissberger, 2014].

This surge in data generation introduced a new unifying concept that brought together a wide range of knowledge domains, shaping two prominent areas of research and practice: Data Science (DS) and Big Data (BD). BD is commonly characterised by three defining dimensions, known as the three V's: volume, velocity, and variety [Cappa et al., 2021]. Volume refers to the massive amounts of data continuously collected from people's actions and choices, particularly through the widespread use of online applications, velocity denotes the speed at which data is generated and must be processed, and variety relates to the heterogeneity of data sources and formats. These characteristics have driven both the research community and industry to develop advanced analytical frameworks capable of processing such data at scale and delivering informed, automated decision-making processes [Raban and Gordon, 2020].

Such analytical frameworks constitute the foundation of DS, which encompasses a set of principles, problem definitions, algorithms, and processes for extracting non-obvious and useful patterns from large datasets [Donoho, 2017]. Many of the constituent elements of DS have been developed in closely related fields, such as ML and data mining, terms that are often used interchangeably with DS, in the literature. The commonality across these disciplines lies in their shared focus on improving decision-making through systematic data analysis. However, while data science draws extensively from these fields, it is broader in scope. ML focuses specifically on the design and evaluation of algorithms for extracting patterns from data, whereas data mining generally concerns the analysis of structured data, often with an emphasis on commercial applications. DS, encompasses both of these considerations while extending further to address a wider range of challenges related to the entire data lifecycle. ML is best considered as a subset of AI. ML emerged as a foundational discipline within the broader landscape of data science, enabling systems to extract meaningful patterns from data through exposure to training datasets and iterative feedback mechanisms. In its early form, ML required programmers to manually identify and engineer the relevant features

that would allow a model to distinguish between classes — a process that, while effective, was inherently time-consuming and heavily dependent on domain expertise [Bini, 2018]. As datasets grew larger and computational power increased, these limitations became increasingly apparent, particularly in high-dimensional problems where the extraction of predictive features was neither straightforward nor scalable. This drove the development of more sophisticated learning architectures, most notably deep learning, which eliminated the need for manual feature engineering by enabling systems to autonomously learn hierarchical representations directly from raw data. The progressive maturation of these techniques, combined with the exponential growth in available data and advances in parallel computing, gradually gave rise to Artificial Intelligence AI as a broader and more encompassing paradigm. Unlike traditional ML, which focuses narrowly on pattern recognition and prediction, AI aspires to replicate higher-order cognitive functions, including reasoning, decision-making, and autonomous adaptation, positioning it as a transformative force across scientific, industrial, and societal domains.

Specifically, AI has a great impact in the healthcare domain, because the large amounts of data generated from sensors, special machines, domain expert reports, images such as Magnetic Resonance Imaging (MRI)s, medical exams, and patient surveillance have contributed to mitigating the expansion of fatal diseases. AI-driven methods for early detection of diseases such as pancreatic cancer [Kerner et al., 2021] or Amyotrophic Lateral Sclerosis (ALS) [Kitaoka et al., 2025] can increase sensitivity, decrease false-positive rates, and provide personalized risk assessment, ultimately enabling earlier interventions and improving patient outcomes. Digital twins [Katsoulakis et al., 2024], which mimic human biological behavior, shifted towards a more AI-driven approach that serves as a powerful tool to assist healthcare experts by generating artificial samples for optimized clinical trials and new experiments.

Moreover, AI has demonstrated significant potential in industrial and manufac-

turing environments, where for instance robotic systems leverage advanced methods such as RL [Kaelbling et al., 1996] to acquire cognitive capabilities and optimize a wide range of operational tasks. These include action and motion planning [Meyes et al., 2017], energy efficiency optimization [Yan and Zhang, 2021], and autonomous path planning [Bhuiyan et al., 2023].

The same applies to the financial sector, where AI-based recommendation systems leveraging sentiment analysis have been introduced to support stock market predictions and investment decision-making [Mohan et al., 2019]. Similarly, in the education domain, AI-powered virtual assistants and intelligent learning platforms have transformed traditional pedagogical approaches by enabling personalized, adaptive, and scalable learning experiences [Vacalopoulou et al., 2024].

In the transportation sector, the emergence of autonomous driving systems represents one of the most disruptive AI-driven innovations, integrating computer vision, deep learning, and real-time decision-making to enhance road safety and traffic efficiency [Grigorescu et al., 2020]. Collectively, these advancements across diverse domains underscore the transformative role that AI has played in reshaping modern society and redefining the way humans interact with technology.

The adoption of AI has also expanded significantly across several emerging fields in communication and service provisioning. In particular, AI has been increasingly leveraged to optimize resources in Cloud and Edge Computing environments, addressing critical challenges such as resource allocation, task scheduling, and energy efficiency in alignment with sustainable computing principles. Consequently, it is notable that AI has driven extensive research efforts toward the development of more intelligent and adaptive management frameworks for such heterogeneous and complex infrastructures. Looking ahead, AI and ML are anticipated to have a transformative impact on 6G networks, addressing the primary challenge of managing the substantial increase in connected devices and data traffic [Zhu et al., 2023] while optimizing network resources, such as bandwidth and computing power,

to ensure efficient network operation [Chataut et al., 2024]. Beyond resource management, AI-driven cybersecurity mechanisms can continuously monitor network traffic, detect anomalies, and proactively respond to emerging security threats [Salem et al., 2024], collectively reinforcing the transformative role of AI in the evolution toward next-generation intelligent communication infrastructures.

As the digital landscape progresses toward full automation, deploying intelligent applications at the edge layer has become a fundamental requirement. By bringing intelligence closer to the point of need, edge computing minimizes the reliance on long-distance communication between clients and servers, thereby improving reliability and ensuring continuous operation even under intermittent cloud connectivity, a property that benefits not only time-critical applications but also general-purpose services serving end users. This architectural shift yields critical benefits, including reduced latency and bandwidth consumption, improved connectivity, more efficient network operation and service delivery, enhanced scalability, faster access to external computational and storage resources, stronger security guarantees, greater data privacy, and improved system reliability. Nevertheless, the deployment and management of applications at the edge layer introduce unique challenges that continue to be an active area of investigation within the research community which can be tackled by using AI.

The main challenges arising in the edge computing paradigm can be summarized as follows:

- **Programmability:** The heterogeneous nature of edge devices hinders the runtime execution of applications, introducing significant difficulties in developing software that can be seamlessly deployed across edge environments. Unlike Cloud Computing, where the underlying infrastructure remains transparent to the user, edge deployments require explicit consideration of device-specific constraints and capabilities [Shi et al., 2016]
- **Data Abstraction:** The diversity of data generated at the edge, spanning mul-

multiple formats and representations, necessitates the adoption of a common data model to fully exploit data usability. This includes determining which data should be collected and processed, and standardizing its format, so as to eliminate redundant computations and avoid unnecessary consumption of computational resources [Sheikh et al., 2025]

- **Privacy and Security:** Retaining computation at the edge, rather than transmitting data to remote servers, constitutes an effective approach to mitigating privacy risks. Nevertheless, several challenges persist, including limited community awareness of edge security vulnerabilities, ambiguity over data ownership, and the absence of lightweight, resource-efficient security mechanisms capable of operating on constrained edge devices within highly dynamic environments [Alwarafy et al., 2020].
- **Resource Management:** This challenge is arguably the most multifaceted, as it encompasses several interconnected sub-challenges. Specifically, resource management at the edge layer involves, among others, the optimization of limited computational resources [Hong and Varghese, 2019], service placement [Zheng et al., 2024], [Supraja et al., 2025], task scheduling [Alizadeh et al., 2020], service offloading and migration [Wang et al., 2018], and edge data caching [Liu et al., 2020b]. These challenges collectively motivate the research community and industry to propose novel technologies, innovative frameworks, and intelligent models aimed at ensuring efficient, reliable, and scalable edge operation.

1.1 Motivation

Given that the edge layer is composed of resource-constrained, decentralized devices yet is expected to support increasingly advanced microservice-based applications, service placement is tightly coupled with the broader imperative for ef-

fective orchestration of edge–cloud infrastructures. Orchestration constitutes a vital operational layer in edge–cloud systems, encompassing the coordinated management of workloads, including services and their constituent components, deployed across distributed environments [Mahmoudi et al., 2018]. It comprises three closely interrelated dimensions: service orchestration, workload orchestration, and resource orchestration [Svorobej et al., 2020]. Modern orchestration frameworks aim to accelerate service delivery, simplify optimization, and reduce operational cost [Nygren et al., 2010] through container-based application deployment, continuous monitoring of compute and network resources, and dynamic adaptation to application-level requirements. Effective orchestration further necessitates continuous system health monitoring, vertical and horizontal scaling to accommodate fluctuating demands, and the deployment of intelligent scheduling and load-balancing mechanisms, all of which must operate proactively to guarantee high availability, reliability, low latency, and scalability.

Within this orchestration context, service placement emerges as a critical sub-problem, defined as the strategic allocation of software components, storage mechanisms, and associated configuration files across edge nodes in accordance with specific computational tasks [Tymplalexix et al., 2025]. In modern edge environments, effective service placement is essential to minimizing latency, optimizing bandwidth utilization, and accommodating user mobility across decentralized, resource-constrained infrastructure. Nevertheless, it remains an inherently complex challenge, as it is subject to NP-hard combinatorial complexity, dynamic workload variations, high resource heterogeneity, stringent energy constraints, and considerable service migration overhead.

Considering the rapid growth of data generated by decentralized edge devices, combined with the increasing complexity of deployed services, has intensified the necessity for AI-driven orchestration mechanisms capable of dynamically coordinating service placement decisions so as to jointly optimize resource utilization

and satisfy user requirements. The evolution of AI has reinforced this imperative, as ML and DL methods now play a central role in anomaly detection, predictive maintenance, and intelligent load balancing. With the rise of Edge Intelligence [Yu et al., 2020], AI models executed at the network edge provide predictive insights into system behavior, user activity, and application demands — capabilities essential for dynamic resource allocation and for sustaining strong QoS and Quality of Experience (QoE).

1.2 Objectives

The primary objective of this thesis is to exploit telemetry data retrieved from heterogeneous and diverse environments, such as Cloud infrastructures and Edge networks, toward the development of DL models capable of efficiently managing the behavior of such environments by delivering an effective AI-driven orchestration mechanism that addresses the dynamic service placement problem. To this end, this thesis presents two main approaches designed to achieve this goal.

More specifically, this thesis presents two AI-driven management methods, each approaching the dynamic service placement problem in heterogeneous edge–cloud infrastructures from a distinct perspective. Both methods aim to dynamically allocate services across edge nodes in a manner that balances the trade-off between the computational resources required to sustain service execution at the edge layer and the concurrent satisfaction of user requirements, namely the delivery of service responses within acceptable latency bounds.

To evaluate the proposed methods, both are assessed in distinct application contexts, however, their design ensures broad applicability across diverse use-case scenarios. The first approach relies on the serverless computing model to determine the optimal composition of the functions-as-a-service constituting a deployed service. By collecting telemetry data from the service’s runtime execution,

this approach employs an AI-driven mechanism to identify the optimal function chain set capable of delivering service responses within acceptable latency bounds. Consequently, this approach addresses the service placement problem by exploiting service-level execution characteristics to balance the trade-off between resource consumption and user satisfaction. The second method approaches the same problem from a resource-centric perspective, collecting, analyzing, and modeling (in real-time) telemetry data derived from the heterogeneous edge nodes responsible for service execution. Based on this analysis, a seamless three-layer AI-driven orchestration mechanism is proposed to dynamically allocate services across edge nodes, thereby balancing the same fundamental trade-off between computational resource utilization and the fulfillment of user QoS requirements.

Regarding the core of both approaches, several DL models have been evaluated and comparatively assessed. Furthermore, in the context of the second method, the evaluation is extended to include a comparative analysis against heuristic-based approaches, ensuring a comprehensive and fair assessment of the proposed method's effectiveness.

1.3 Knowledge gap and research questions

Despite the growing body of research leveraging AI for edge–cloud orchestration, the dynamic service placement problem in heterogeneous edge environments remains insufficiently addressed. The majority of existing approaches treat placement either as a single-stage decision problem or as a static optimization addressed through heuristic-based models. These works either adopt a reactive scheduling strategy that responds exclusively to instantaneous telemetry observations, or rely on a monolithic learning module trained offline or in a centralized fashion, without the capacity to continuously adapt to the non-stationary conditions that characterize real-world edge deployments. Considering the surveyed approaches collectively, while each addresses individual dimensions of the placement problem

with varying degrees of effectiveness, a unified mechanism that jointly optimizes the allocation of services across heterogeneous edge nodes while simultaneously satisfying resource constraints and user QoE requirements in a continuously adaptive, closed-loop manner under real-world dynamic conditions remains an open and insufficiently addressed research challenge.

A promising deployment paradigm that has gained considerable traction in both research and industry as a means of tackling the dynamic service placement problem is the serverless model, realized through FaaS, wherein applications are decomposed into fine-grained function pipelines. Unlike traditional Virtual Machine (VM)-based approaches, the serverless model abstracts the underlying infrastructure and allocates resources exclusively upon service invocation, thereby eliminating idle resource consumption and naturally accommodating the unpredictable workload behavior that is endemic to edge environments. These properties render the serverless paradigm particularly well-suited for edge nodes, where computational and memory resources are scarce and must be managed with precision. Exploiting the serverless model for dynamic service placement offers a principled approach to balancing the trade-off between resource utilization and user satisfaction, ensuring that services are allocated across edge nodes in a manner that maintains resource consumption within acceptable bounds while delivering responses within latency requirements. Nevertheless, existing approaches largely treat individual functions as isolated execution units, overlooking the performance implications of inter-function composition. Specifically, the optimal partitioning of an ML or DL workflow into a pipeline of serverless functions, a decision that directly governs end-to-end response time and, consequently, user QoE — remains an open problem that is not systematically addressed by current orchestration frameworks targeting dynamic service placement.

In practice, the dynamic nature of the placement problem is further compounded by the heterogeneous composition of edge infrastructures, which exhibit diver-

gent node capabilities, continuously shifting workload patterns, user mobility, and privacy and bandwidth constraints that fundamentally preclude centralized training on raw telemetry data. Collectively, these characteristics expose a significant gap in the design of closed-loop, AI-driven orchestration systems simultaneously capable of: (a) dynamically determining the optimal service composition at the function level to minimize response latency while preserving the resource efficiency afforded by the serverless model, (b) forecasting near-future resource dynamics locally across heterogeneous edge nodes to enable proactive placement decisions, (c) enhancing prediction robustness through collaborative learning without necessitating the exchange of raw telemetry data, and (d) exploiting global system observability to support coordinated, mobility-aware service placement across the edge–cloud continuum.

This thesis tries to answer the following research questions:

RQ1: "Can the dynamic service placement problem in heterogeneous edge environments be effectively addressed through AI-driven orchestration mechanisms that outperform conventional optimization-based approaches under non-stationary workload conditions?"

This question targets the fundamental limitation of existing approaches, which treat service placement as a static or single-stage problem. Conventional optimization-based models, whether exact, heuristic, or meta-heuristic, fail to capture the temporal evolution of workload patterns and node availability across heterogeneous edge infrastructures. This thesis demonstrates that AI-driven, closed-loop orchestration mechanisms, continuously updated on runtime telemetry, can effectively address the dynamic service placement problem, outperforming conventional optimization-based baselines in allocating services across heterogeneous edge nodes while jointly satisfying resource utilization and QoE constraints.

RQ2: "Can an AI-driven approach that considers the service itself, by exploiting the serverless FaaS paradigm, constitute a resource-efficient and QoE-aware so-

lution for dynamic service placement at the edge?”

This question evaluates whether modeling the service’s internal composition, rather than solely the underlying node resources, provides a viable and effective placement strategy. By decomposing services into fine-grained function pipelines and applying AI-driven techniques to determine the optimal function partitioning at runtime, this thesis demonstrates that the first proposed approach, by exploiting AI-driven mechanisms within the serverless FaaS paradigm, can minimize end-to-end response latency while preserving resource efficiency — establishing it as a principled and QoE-aware approach to dynamic service placement in resource-constrained edge environments.

RQ3: “From a complementary perspective, can an AI-driven approach that targets the edge node resources directly be considered a resource-efficient and QoE-aware solution for dynamic service placement at the edge?”

Whereas RQ2 approaches the placement problem from the service composition perspective, this question investigates the resource-centric viewpoint, in which telemetry data collected from heterogeneous edge nodes is analyzed and modeled to drive placement decisions. This thesis demonstrates that an AI-driven approach that continuously monitors, forecasts, and responds to node-level resource dynamics can achieve efficient service allocation across heterogeneous edge infrastructures, maintaining resource consumption within acceptable bounds while delivering service responses that satisfy user QoE requirements.

RQ4: “Given that the service placement problem is considered NP-hard, can fully AI-driven orchestration mechanisms produce more effective and proactive placement decisions compared to traditional heuristic-based models?”

This question addresses the computational intractability of the service placement problem and evaluates whether AI-driven approaches can surpass the solution quality and responsiveness of well-established heuristic methods such as genetic

algorithms, Particle Swarm Optimization, and linear programming. This thesis demonstrates that AI-driven mechanisms, by learning from real-time telemetry, can produce placement decisions that are not only competitive with heuristic baselines in terms of solution quality, but also significantly more proactive, as they anticipate workload shifts and resource exhaustion before constraint violations occur, rather than reacting to them after the fact.

RQ5: “Considering that the edge ecosystem is composed of decentralized devices hosting heterogeneous services, do the proposed AI-driven approaches adequately address scalability, user mobility, and security — the major challenges of edge computing — while simultaneously exploiting global system observability for coordinated placement decisions?”

This question evaluates the broader applicability and robustness of the proposed AI-driven approaches with respect to the three principal challenges of edge computing. This thesis demonstrates that the proposed approaches address scalability through the invocation-based resource model of the serverless paradigm and through distributed, collaborative learning mechanisms that do not require centralized raw data aggregation. User mobility is accommodated by incorporating spatial demand variability and mobility patterns into the placement decision process. Security and privacy are preserved by ensuring that collaborative learning operates exclusively on model updates rather than raw telemetry, thereby respecting the bandwidth and privacy constraints inherent to decentralized edge deployments.

The primary objective of this thesis is to investigate whether end-to-end AI-driven orchestration mechanisms can effectively address the dynamic service placement problem in heterogeneous edge–cloud environments, with the dual aim of optimizing computational resource utilization and satisfying user QoE requirements. To achieve this, the thesis presents two complementary methods that approach the problem from distinct perspectives: the first targets the service composition

level by exploiting the serverless FaaS paradigm, while the second targets the edge node resource level by collecting and modeling runtime telemetry data. In order to evaluate the significance of both methods, they will be assessed in diverse and realistic deployment scenarios, measuring placement performance in terms of resource efficiency, response latency, and user satisfaction. Furthermore, the proposed AI-driven approaches will be directly compared against well-established heuristic-based models, providing the necessary evidence to determine whether learning-based mechanisms constitute a more effective and proactive alternative for addressing the inherent complexity of the placement problem. Finally, the broader applicability of the proposed methods will be examined with respect to the principal challenges of edge computing, scalability, user mobility, and security, evaluating whether the proposed solutions maintain their effectiveness as these conditions are introduced and varied across experimental scenarios.

1.4 Research Contributions

The research conducted and presented in this thesis can be organized into four key contributions, addressing the dynamic service placement problem in heterogeneous edge–cloud environments from two complementary perspectives.

RC1: An AI-driven Serverless Function Regulator for QoE-aware Service Composition

Addressing **RQ1** and **RQ2**, this thesis presents a novel AI-driven method that approaches the dynamic service placement problem from the service composition perspective by exploiting the serverless FaaS paradigm. Moving beyond standard intelligent resource adaptation, the proposed approach, referred to as the AI-based Function Regulator, automatically determines the optimal number and grouping of serverless function blocks composing an ML/DL pipeline, with the objective of minimizing end-to-end response time and maximizing user QoE. By collecting

and analyzing runtime telemetry across candidate pipeline configurations, the approach resolves the open problem of optimal function partitioning, which existing orchestration approaches have systematically overlooked.

RC2: A Closed-loop, Three-tier AI-driven Orchestration approach for Dynamic Resource-centric Service Placement

Addressing **RQ1** and **RQ3**, this thesis presents a decentralized, multi-tier AI-based orchestration approach that addresses the placement problem from the resource perspective. The proposed solution integrates three tightly coupled operational tiers: (i) lightweight node-level incremental learning agents that continuously adapt to streaming telemetry data under concept drift using Elastic Weight Consolidation, (ii) an event-triggered, asynchronous Federated Learning mechanism that activates upon predictive performance degradation, improving model robustness without raw data exchange, and (iii) a centralized RL agent based on Proximal Policy Optimization (PPO) that formulates placement as a Markov Decision Process (MDP) and jointly optimizes scaling and migration decisions by considering both resource availability and user proximity.

RC3: Design and Implementation of Service Components for Dynamic Placement in Edge–Cloud Infrastructures

Addressing **RQ3** and **RQ4**, both approaches are realized as fully operational service components and evaluated in realistic edge–cloud deployment scenarios. The serverless-oriented approach is assessed in the context of ML/DL inference pipelines, while the resource-centric approach is evaluated across 100 heterogeneous placement scenarios. In both cases, the proposed components are designed to be generalizable and applicable across diverse use-case scenarios beyond the specific evaluation contexts presented in this thesis.

RC4: Comparative Evaluation of the Proposed AI-driven approaches against Heuristic and Supervised Learning Baselines

Table 1.1: List of publications related to the contributions of this thesis.

Title	Authors	Venue	Contribution
ML-FaaS: Toward Exploiting the Serverless Paradigm to Facilitate Machine Learning Functions as a Service	E. Paraskevoulakou , D. Kyriazis	IEEE Transactions on Network and Service Management, 2023 (Impact Factor (IF): 5.4)	RC1, RC3, RC4
A Three-Tier AI-based Approach for Dynamic Application Placement in Cloud-Edge Environments	E. Paraskevoulakou , C. Symvoulidis, P. Karamolegkos, G. Kousiouris, D. Kyriazis	Future Generation Computer Systems, 2026, Elsevier (IF: 6.1)	RC2, RC3, RC4
Leveraging the Serverless Paradigm for Realizing Machine Learning Pipelines across the Edge-Cloud Continuum	E. Paraskevoulakou , D. Kyriazis	2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN 2021), IEEE	RC1, RC3
[BEST PAPER AWARD] Enhancing Cloud-based Application Component Placement with AI-driven Operations	E. Paraskevoulakou , J.D. Totow Tom-Ata, C. Symvoulidis, D. Kyriazis	2024 14th IEEE Annual Computing and Communication Workshop and Conference (CCWC 2024)	RC3
Dynamic Resource Allocation on the Edge: A Causal and Contextually-Aware Machine Learning Approach	C. Symvoulidis, E. Paraskevoulakou , A. Kiourtis, A. Mavrogiorgou, D. Kyriazis	2024 10th Intelligent Systems Conference (IntelliSys 2024), Springer, Cham	RC2, RC3, RC4
Polymorphic Cloud Application Design Assisted by Open Source Software Classification	J.D. Totow Tom-Ata, K. Kritikos, M.A. Di Girolamo, E. Paraskevoulakou , C. Symvoulidis, D. Kyriazis	2023 5th IEEE International Communication Engineering and Cloud Computing Conference (CECCC 2023)	RC3
A Framework for Cognitive, Decentralized Container Orchestration	R.C. Sofia, J. Salomon, S. Ferlin-Reiter, et al., E. Paraskevoulakou , D. Kyriazis	IEEE Access, 2024	RC2, RC3
Towards Scalable Federated Container Orchestration: The CODECO Approach	R.C. Sofia, J. Salomon, R. Carroll, et al., E. Paraskevoulakou , D. Kyriazis	arXiv preprint, 2026	RC2, RC3

Addressing **RQ4** and **RQ5**, the significance of both proposed approaches is rigorously evaluated through direct comparative analysis. The serverless-oriented approach is benchmarked across multiple pipeline configurations, demonstrating that the optimal composition is not trivially determined by the number of functions alone. The resource-centric approach is compared against supervised ML baselines and well-established heuristic placement methods, including First-Fit and Best-Fit Bin Packing, demonstrating significantly improved placement accuracy, resource efficiency, and QoE satisfaction. This comparative evaluation provides the necessary empirical evidence to validate the superiority of the proposed AI-driven approaches over conventional methods for addressing the NP-hard dynamic service placement problem.

In addition, Table 1.1 demonstrates how the publications are mapped to the aforementioned research contributions.

1.5 Outline of dissertation

The remainder of this dissertation is organized in the following way:

- **Chapter 2** presents the theoretical and technical groundwork for the research contributions developed throughout this thesis. It surveys the architectural landscape of edge computing and the cloud-to-edge continuum, analyzes the service placement problem and its NP-hard complexity, reviews the orchestration frameworks and container-based technologies that govern service lifecycle management across distributed edge–cloud infrastructures, and examines the principal families of optimization methods applicable to decentralized edge systems — from classical mathematical programming and meta-heuristic approaches through ML-based workload prediction, FL, RL-based placement, and the serverless FaaS paradigm as a placement abstraction.
- **Chapter 3** presents the precise problem context within which the research contributions of this thesis are situated. The dynamic service placement problem is formally characterized and formulated as a multi-objective optimization problem, and the key technical challenges that render classical solutions insufficient, including resource heterogeneity, latency and QoS constraints, and the non-stationary nature of edge deployments are systematically identified and analyzed.
- **Chapter 4** surveys the state of the art across the two research dimensions that underpin this thesis. It first reviews the intersection of serverless computing and ML/DL workload deployment, covering FaaS-based frameworks and the open problem of function pipeline composition, and then examines the application placement problem in edge–cloud environments, spanning classical optimization methods, orchestration mechanisms, and state-of-the-art learning-based approaches. For each dimension it systematically identifies

the limitations of existing solutions that motivate the contributions of Chapter 5 and Chapter 6.

Chapter 5 presents the first proposed method, which approaches the dynamic service placement problem from the service composition perspective by exploiting the serverless FaaS paradigm. The Chapter is structured as follows: (i) motivation and service-centric perspective, establishing the rationale for the FaaS-based placement approach and the limitations of exclusively infrastructure-centric views, (ii) the ML-FaaS proof-of-concept, covering the offline data analytics and model training phase and the online serverless function pipeline, (iii) the proposed AI-based Function Regulator solution, detailing its core components and the automated pipeline composition optimization mechanism, (iv) a performance evaluation spanning the quality of the offline ML/DL classification model, the end-to-end response time of the online serverless pipeline, and the effectiveness of the Function Regulator in identifying the optimal composition, including a comparative analysis across candidate DL models, and, (v) a quantitative assessment of the practical advantages of the serverless deployment approach from both a technical and operational perspective.

Chapter 6 presents the second proposed method, which addresses the dynamic service placement problem from the resource-centric perspective via a closed-loop, three-layer AI-driven orchestration solution. The Chapter is structured as follows: (i) motivation and resource-centric perspective, establishing the rationale for the infrastructure-level placement, (ii) the system model and problem formulation, defining the multi-objective optimization objective governing placement decisions under dynamic edge conditions, (iii) a detailed description of the proposed three-layer solution and its core components, (iv) a comprehensive performance evaluation across all three layers with comparative analysis against supervised ML baselines and

classical heuristic placement strategies, and (ν) a quantitative assessment of the joint latency–resource optimization advantage of the proposed approach over deterministic placement approaches.

- **Chapter 7** summarizes the doctoral dissertation and its key contributions. The Chapter is structured as follows: (*i*) a critical reflection on the extent to which the proposed approaches address the dynamic service placement problem in heterogeneous edge environments, (*ii*) a synthesis of the answers to the research questions defined in Chapter 1, and (*iii*) an outline of the open research challenges and future directions derived from the present work, identifying promising avenues for extending the proposed approaches toward next-generation intelligent edge–cloud orchestration systems.

Chapter 2

Background: edge-cloud continuum and service placement

Chapter Structure

This Chapter is constructed as follows:

- **Section 2.1 – Edge computing paradigm**, introduces the fundamental concepts and architectural paradigms of edge computing, covering MEC, Fog Computing, and Cloudlet Computing, discussing their key properties and suitability for resource-constrained deployments.
- **Section 2.2 – Cloud-to-edge continuum architecture**, presents the architectural evolution from centralized cloud infrastructures towards the decentralized edge–cloud continuum, discussing how the two paradigms coexist and complement one another to support heterogeneous service deployments.
- **Section 2.3 – Service placement fundamentals**, defines the service placement problem in edge–cloud environments, discussing its NP-hard complexity, key challenges including resource heterogeneity, user mobility, and migration overhead, and its tight coupling with effective orchestration.
- **Section 2.4 – Orchestration frameworks and technologies**, reviews the key

dimensions of orchestration — service, workload, and resource — along with modern orchestration frameworks and container-based technologies that enable dynamic, scalable service management across distributed edge–cloud infrastructures.

- **Section 2.5 – Optimization in decentralized systems**, examines traditional and AI-driven optimization approaches applied to decentralized edge systems, analyzing the limitations of classical methods and surveying the state of the art in learning-based mechanisms for adaptive and proactive placement decision-making.

This chapter establishes the theoretical and technical foundation upon which the research contributions of this thesis are built. The architectural landscape of edge computing is first surveyed, followed by an analysis of the cloud-to-edge continuum, the service placement problem, the orchestration frameworks that govern service lifecycle management, and the principal families of optimization methods applicable to decentralized edge–cloud systems. The research problem and related work that build upon this foundation are developed in Chapters 3 and 4 respectively.

2.1 Edge computing paradigm

As established in Chapter 1, the proliferation of IoT devices and the exponential growth of data generated at the periphery of the network have exposed fundamental limitations of the traditional centralized cloud computing model. In particular, the linear growth of cloud computing capabilities has proven insufficient to meet the multi-source data processing demands of massive edge data, the network bandwidth and transmission speed have reached a bottleneck due to large-scale user access and long-distance data transmission, and the outsourcing of user data to remote cloud servers introduces significant risks to security and privacy [Zheng

et al., 2024].

These limitations have collectively motivated the emergence of edge computing as a novel paradigm that enables computation, storage, and intelligent service provision in close proximity to the source of data, in collaboration with centralized cloud infrastructure [Khan et al., 2019].

Edge computing is not a monolithic concept but rather encompasses several architectural paradigms, each designed to address specific deployment contexts, application requirements, and resource constraints. The principal edge computing paradigms are discussed in the following subsections: MEC, Fog Computing, and Cloudlet Computing, covering their defining properties, architectural characteristics, and suitability for resource-constrained edge deployments.

2.1.1 Multi-access Edge Computing (MEC)

MEC, formerly known as Mobile Edge Computing, was standardized by the European Telecommunications Standards Institute (ETSI) as a decentralized computing architecture in which storage, signal processing, control, and computation are distributed in an efficient and logical manner between the data source and the cloud [Shah et al., 2018].

Rather than relying on a centralized cloud infrastructure, MEC extends the cloud computing model by bringing its capabilities directly to the network edge, thereby significantly reducing end-to-end latency and alleviating the processing burden imposed on remote cloud servers, which would otherwise have to handle massive volumes of data under constrained bandwidth conditions [Shah et al., 2018]. This paradigm is particularly motivated by the inability of conventional cloud computing to satisfy three fundamental requirements of modern mobile and IoT applications: mobility support, location awareness, and ultra-low latency [Khan et al., 2019]. The ETSI formally defined four key characteristics of MEC: (a) on-premises

isolation, whereby MEC platforms can operate independently from the rest of the network, (b) proximity to end users providing low latency for delay-sensitive applications, (c) location awareness through the exploitation of low-level signaling for information sharing among edge-distributed devices, and (d) network context information, which enables intelligent, context-aware service delivery [Abbas et al., 2017]. These properties extend earlier edge computing architectures by incorporating wireless access and heterogeneous virtualization technologies.

From an architectural perspective, MEC systems are modelled as hierarchical structures in which user equipment (UE) connects to fog access points (F-APs), which in turn interconnect with one another and with the cloud [Shah et al., 2018]. This layered architecture is also a key enabler of Fog Radio Access Networks (F-RANs), which are considered essential components of 5G mobile network architectures. In F-RANs, remote radio heads (RRHs) and UEs are equipped to perform local signal processing, cooperative radio resource management, and distributed storage — functions that substantially reduce the processing burden on centralized base-band unit (BBU) pools and minimize fronthaul traffic [Shah et al., 2018]. Consequently, F-RANs built upon MEC principles can deliver high spectral and energy efficiency, low latency, and reliable service continuity for diverse IoT applications, including smart healthcare, smart transportation, vehicular connectivity, and smart city systems.

A key distinction between MEC and Fog Computing lies in the location of intelligence: whereas Fog Computing situates intelligence within the local area network, MEC embeds it directly within the edge devices themselves [Shah et al., 2018]. This architectural distinction renders MEC especially well-suited for resource-constrained edge deployments in which latency-critical services must be executed as close as possible to the data source, without dependence on intermediate network layers. The primary open challenges associated with MEC include seamless execution handover as mobile users traverse coverage areas, security and privacy pro-

tection during task offloading, intelligent edge caching strategies for frequently requested content, and real-time data processing scalability as the number of interconnected devices continues to grow exponentially [Shah et al., 2018].

2.1.2 Fog Computing

Fog Computing was introduced by Cisco as a paradigm that extends the cloud computing model toward the edge of the network, empowering applications and services to run directly on billions of smart connected devices [Vaquero and Roderomero, 2014]. It has been specified as a scenario in which a large number of heterogeneous, ubiquitous, and decentralized devices communicate and potentially cooperate among themselves and with the network to perform storage and processing tasks without the intervention of third parties [Abbas et al., 2017].

Fog Computing emerged primarily as a response to the proliferation of IoT devices and the demands of big data analytics, functioning as an extension of cloud computing designed for real-time, low-latency applications [Mehdipour et al., 2019]. Its distributed architecture enables real-time data analytics at the edge of the network, reduces transmission overhead, and provides location-aware, context-sensitive services to mobile users [Khan et al., 2019]. The limits of the Fog Computing paradigm are deliberately flexible, allowing implementations that range from IoT device clusters to cloud service integration, making it applicable towards diverse domains including smart manufacturing, healthcare, and intelligent transportation [Songhorabadi et al., 2023].

2.1.3 Cloudlet Computing

The Cloudlet concept was first proposed by Satyanarayanan et al. [Satyanarayanan et al., 2009] as a lightweight, proximity-based cloud extension designed to bridge the gap between resource-constrained mobile devices and distant cloud infras-

structure. A cloudlet is defined as a trusted, resource-rich computing cluster or server that is well-connected to the Internet and made available to nearby mobile devices, offering both computational and storage resources through dynamically formed, virtualized local environments.

Unlike fixed cloud infrastructure, cloudlets can be dynamically instantiated on any device within a local area network with available resources, without requiring permanent installation near a wireless access point. Cloudlets operate as discoverable, generic, stateless servers located in single-hop proximity to mobile devices and can function independently of cloud services when required. By leveraging the full potential of virtual machines, cloudlets facilitate computational offloading for mobile devices, enabling the execution of resource-intensive and latency-sensitive applications at the network edge without incurring the high latency penalties of remote cloud processing [Shaukat et al., 2016].

2.1.4 Comparative analysis and suitability for resource-constrained edge deployments

Despite their architectural differences, the paradigms described above share a common objective: bringing computation and intelligence closer to the source of data in order to reduce latency, minimize bandwidth consumption, and improve the overall quality of service for end users. Table 2.1 provides a comparative overview of the paradigms with respect to their key properties and primary application domains.

Among these paradigms, MEC and Fog Computing are particularly well-suited to resource-constrained edge deployments due to their invocation-based resource allocation models, support for heterogeneous device ecosystems, and intrinsic location awareness. Cloudlet Computing offers a more targeted solution for single-hop, high-performance offloading scenarios. As this thesis addresses the dynamic

Table 2.1: Comparative overview of edge computing paradigms.

Property	MEC	Fog Computing	Cloudlet
Standardization body	ETSI	Cisco / OpenFog	Carnegie Mellon
Proximity to user	Very high	High	Very high
Latency	Ultra-low	Low	Ultra-low
Mobility support	High	High	Moderate
Location awareness	Yes	Yes	Yes
Resource constraints	High	High	Moderate
Infrastructure model	Decentralized	Decentralized	Dynamic
Primary application domain	5G, V2X, AR	IoT, Smart city	Mobile offloading

service placement problem in heterogeneous edge–cloud environments, the architectural properties of these paradigms — particularly their resource heterogeneity, decentralized topology, and workload variability — constitute the primary operational context within which the proposed AI-driven approaches are designed and evaluated.

2.2 Cloud-to-edge continuum architecture

Cloud computing and edge computing are not mutually exclusive paradigms but rather architecturally complementary approaches that address fundamentally different dimensions of modern decentralized computing [Cisco,]. Although both paradigms share common objectives in terms of data storage and processing, they differ substantially across a number of critical parameters — including the physical location of computation, response time, processing volume, scalability, and real-time analytical capability — and it is precisely these differences that define their respective roles within the cloud-to-edge continuum [Al-Dulaimy et al., 2024].

From an architectural standpoint, cloud computing adopts a centralized model in which all computation and storage are performed at remote data centers, whereas edge computing operates through a distributed architecture that positions processing resources close to the data source [Abd Alnabe and Zeebaree, 2024]. This distinction has direct implications for performance: cloud-based processing intro-

duces inherent latency due to the distance between the data source and the processing infrastructure, rendering it unsuitable for time-critical applications that require immediate responses.

Edge computing, by contrast, enables real-time analysis and autonomous decision-making directly at the network periphery, leveraging the contextual and spatial proximity of edge devices to IoT data sources [Bagher et al., 2021]. However, these performance advantages come at a cost: edge devices operate under stringent hardware constraints, offering limited computational power, restricted scalability, and micro-scale data storage capacity compared to the virtually unbounded resources available in cloud environments [Ali-Eldin et al., 2021]. Cloud infrastructure, on the other hand, excels in workloads that demand substantial computing power, extensive long-term data storage, advanced analytics over large datasets, and enterprise-scale scalability — capabilities that edge nodes cannot replicate [Matetić et al., 2022]. Furthermore, while edge deployments offer stronger data protection by keeping sensitive data locally and minimizing transmission over public networks, cloud infrastructure remains more mature and broadly adopted, underpinned by a well-established ecosystem of services and applications [Mansouri and Babar, 2021]. Together, these complementary strengths and limitations — summarized in Table 2.2 — establish the fundamental rationale for the cloud-to-edge continuum: rather than selecting one paradigm over the other, modern distributed systems require an integrated architectural model in which workloads are dynamically allocated across both tiers according to their latency, resource, privacy, and scalability requirements.

2.3 Service placement fundamentals

One of the key challenges in edge–cloud computing infrastructures is dynamic service placement [Taleb et al., 2025, Sfaxi et al., 2024]. The placement of each service may be replicated simultaneously across multiple edge servers, and each server

Table 2.2: Comparative overview of Cloud Computing and Edge Computing.

Parameter	Cloud Computing	Edge Computing
Architecture	Centralized computing architecture	Decentralized computing architecture
Response Time	Reduced speed	Increased speed
Computing Power	Greater power, ensuring efficient processing	Reduced power, limited data processing
Scalability	High scalability, facilitating enterprise growth	Restricted scalability due to resource limits
Network / Data Protection	Less secure due to data transmission	More secure, data retained locally
Data Storage Quantity	Big data storage	Micro data storage
Volume of Data Processed	Big data processing	Limited quantity of data
Data Processing Location	Processes data in the cloud, distanced from source	Processes data at the edge, near the source
Data Analytics Performance	High efficiency due to advanced computing power	Decreased efficiency due to hardware limits
Real-Time Analysis	Cannot provide real-time analysis	Provides instant real-time results
Decision-Making	Centralized decision-making	Autonomous real-time decisions at device level
Maturity Level	Mature, with broad established applications	Still evolving, with continuous developments
Annual Costs	Higher operational cost	Lower operational cost
Use Cases	Large enterprises: flexibility, agility	Autonomous vehicles, smart agriculture, surveillance

can host multiple services subject to its available resources [Malazi et al., 2022]. All services are additionally maintained in the cloud as a fallback tier, ensuring that requests can always be fulfilled even when edge computing resources are inadequate [Hong and Varghese, 2019]. When a service request arrives at an edge server from a mobile user, the server responds instantly if the requested service is locally hosted. Otherwise, the request is either directed to a neighbouring edge server that hosts the service or offloaded to the cloud data centre via the backhaul

network. Alternatively, the edge server may initiate on-demand service instantiation locally before processing the request [Xu et al., 2018].

A fundamental distinction in the service placement literature is between static and dynamic approaches. Static placement strategies assume that user distributions, workload intensities, and resource availability remain stable over time, and therefore compute a fixed deployment configuration that is not revisited during system operation [Maia et al., 2020]. While this assumption simplifies the optimization problem considerably, it is rarely tenable in real edge environments, where user mobility, fluctuating request rates, and time-varying resource contention continuously alter the conditions under which placement decisions were originally made. Dynamic placement strategies, by contrast, treat the deployment configuration as a continuously adjustable control variable, re-evaluating and updating service locations in response to observed or predicted changes in system state [Maia et al., 2020, Kazmi et al., 2024].

Modern edge applications further complicate the placement problem through their microservice-based architecture. Rather than deploying monolithic services, contemporary applications are decomposed into sets of interdependent microservices whose execution dependencies are represented as Directed Acyclic Graphs (DAGs) [Liu et al., 2019]. Each node in the DAG corresponds to a specific functional component, and directed edges encode communication or data-flow dependencies between components. Placement decisions must therefore account not only for the resource requirements of individual microservices but also for the inter-service communication latency introduced when dependent components are distributed across distant edge nodes. Consolidating all microservices of an application onto a single node minimizes inter-service latency but risks resource saturation, distributing them across multiple nodes improves load balance but increases network communication overhead [Kazmi et al., 2025, Maia et al., 2020]. This structural complexity, combined with the heterogeneity of edge infrastructure — where

nodes range from resource-constrained base stations to more capable core nodes and ultimately the cloud, each with different capacities, costs, and proximity to end users — renders the placement problem significantly harder than its cloud counterpart.

A closely related and often overlooked dimension of the placement problem is load distribution. Because an individual edge node may lack sufficient resources to handle the full request volume directed at a given application, the incoming load must be spread across multiple nodes that each host a replica of that application [Maia et al., 2020]. Critically, placement and load distribution decisions are mutually constrained: requests can only be forwarded to nodes that currently host the relevant application, meaning that an optimal strategy requires their joint optimization rather than treating them as independent sub-problems [Maia et al., 2020]. Furthermore, application load is inherently dynamic, varying across both space and time as users move through the network or alternate between active and inactive states [Maia et al., 2020]. This temporal variability necessitates continuous re-evaluation of placement and distribution decisions to sustain acceptable service performance, yet frequent reconfigurations are not cost-free: relocating a service instance incurs a migration delay proportional to the size of the application state being transferred, which can itself degrade response times during the transition period [Maia et al., 2020].

Formally, the service placement problem can be modelled as an optimization problem over binary decision variables, where each variable encodes whether a given service is deployed at a given edge node, and a corresponding routing variable determines which node serves each user request. Finding the optimal placement for a microservice-based application across a heterogeneous edge environment is a complex combinatorial problem, given the exponentially large number of possible deployment configurations. This formulation is recognized as NP-hard, and remains one of the most critical and challenging open problems in the field [He

et al., 2018, Bahrami et al., 2023].

This computational intractability rules out exact solvers such as Integer Linear Programming (ILP), except in small-scale, tightly constrained scenarios, motivating the use of approximate, heuristic, and learning-based methods for realistic deployments [Taleb et al., 2025]. Among the classical deterministic heuristics, Bin Packing approaches, most notably *First-Fit* and *Best-Fit*, model service placement as the assignment of variable-sized workloads into capacity-constrained nodes. First-Fit assigns each service to the first node with sufficient remaining capacity, while Best-Fit targets the node whose remaining capacity most closely matches the service’s resource demand, thereby minimizing fragmentation. Although both variants are computationally lightweight and easily implementable in real-time deployment scenarios, their greedy, single-pass nature makes them susceptible to suboptimal global configurations, particularly under heterogeneous resource profiles and dynamic workload conditions [Coffman et al., 2013]. These limitations directly motivate the comparative evaluation presented in this thesis, wherein the proposed AI-driven frameworks are benchmarked against First-Fit and Best-Fit Bin Packing baselines.

Moreover, because the service placement objective functions involve trade-offs between application QoS requirements, privacy constraints, and available resources, there is generally no single optimal solution but rather a set of Pareto-optimal configurations that cannot be improved in one dimension without degrading another [Petrucchi et al., 2025]. To illustrate, a latency-sensitive application may benefit from having all its component services co-located on the same edge node to eliminate inter-service network delay, however, if a single node cannot provide the required resources, the placement model must distribute services across multiple nodes while simultaneously satisfying both the latency target and any applicable data locality or privacy constraints [Ahmadvand and Foroutan, 2025].

The objective is typically to optimize a composite utility metric subject to con-

straints on computational capacity, storage, bandwidth, and service deadlines [Somesula et al., 2023]. In practice, however, service placement is inherently a multi-objective problem: performance dimensions such as end-to-end latency, inter-server proximity, load balance across nodes, and data privacy requirements are mutually conflicting, and gains along one axis routinely degrade performance on others [Keshavarz Haddadha et al., 2024]. For instance, co-locating all microservices of an application on a single edge node reduces inter-service communication latency, but may cause resource saturation and increased queuing delays under high demand [Kazmi et al., 2025]. Similarly, minimizing deadline violations by aggressively migrating services to follow user mobility tends to increase migration costs and temporarily elevate response times during the transfer, while retaining services in place risks latency degradation as the gap between a user and its serving node grows [Maia et al., 2020]. This three-way tension between service quality, resource cost, and migration overhead is a defining characteristic of the dynamic placement problem.

The cooperative dimension of service placement becomes particularly pronounced in Pervasive Edge Computing (PEC) networks, where multiple MEC servers collaborate to deliver seamless offloading services to highly mobile users [Ning et al., 2020]. In such settings, a service must be dynamically migrated across PEC servers to follow user movement throughout the entire offloading pipeline, from raw data upload to result download. This introduces a fundamental migration trade-off: relocating a service incurs transmission overhead, whereas retaining it at a distant server risks backhaul interruption and increased latency. Critically, because multiple mobile users share the same pool of edge resources, their placement configurations become inter-dependent — the decision of whether and where to migrate a service for one user directly affects the resource availability for others [Ning et al., 2020]. This inter-dependence, combined with the difficulty of predicting long-term user mobility patterns, renders greedy or myopic placement policies inadequate and motivates the need for online algorithms capable of balancing cur-

rent service execution cost against future migration cost without requiring a priori knowledge of user trajectories.

Given the problem's NP-hard nature and considering the multi-objective structure of real deployments, a diverse range of algorithmic approaches has been proposed to tackle the challenge. The most well-established meta-heuristic models are as follows:

- **Ant Colony Optimization (ACO)** models the placement search as a collective path-finding process, offering a strong balance between solution exploration and exploitation [Zarei et al., 2025, Huang et al., 2020].
- **Particle Swarm Optimization (PSO)** treats candidate placements as particles navigating a solution space guided by individual and collective experience, and has demonstrated particular effectiveness in minimizing request waiting times and decision latency [Rodríguez et al., 2021].
- **Genetic Algorithm (GA)** evolves a population of placement configurations through selection, crossover, and mutation operators, and consistently achieves strong performance in minimizing inter-server distance and limiting the number of servers involved in a placement — properties directly relevant to privacy-sensitive applications [Zheng et al., 2024].
- **Simulated Annealing (SA)** explores the solution space by probabilistically accepting suboptimal moves in early stages, gradually reducing this acceptance as the search converges, making it well-suited to deadline-aware and energy-constrained scenarios [Mahjoubi et al., 2024].
- **Artificial Bee Colony (ABC)** partitions the search effort among specialized agent roles — employed bees, onlookers, and scouts — to balance local refinement with broader exploration, and has been shown to maintain low average inter-service latency [Zhou et al., 2023].

On the other hand, AI-based approaches are particularly well suited to the ser-

vice placement problem, as they operate as function approximators capable of learning complex input-output mappings directly from data, without requiring an explicit analytical formulation of the underlying optimization landscape [Keshavarz Haddadha et al., 2024]. These methods are discussed in detail in Section 2.5.

Despite the extensive catalogue of algorithms proposed for the service placement problem, it is important to emphasise that service placement does not operate in isolation, but is deeply embedded within a broader orchestration stack that coordinates the full lifecycle of services across heterogeneous edge–cloud infrastructures. Effective orchestration must simultaneously manage resource provisioning, service scaling, fault recovery, and policy enforcement, all of which are directly conditioned on where and how services are placed at any given moment [Chiang et al., 2023]. As a consequence, a placement decision that fails to account for its downstream effects on resource availability, inter-service communication paths, or QoS guarantees risks cascading into system-wide performance degradation that reactive scaling alone cannot fully address.

Therefore, an orchestration framework that treats placement as a static, one-time configuration step is inherently unable to respond to the dynamic conditions — including user mobility, load fluctuations, and node failures — that characterize real edge deployments. This tight coupling between placement and orchestration motivates the need for AI-driven approaches capable of jointly optimizing placement decisions within a broader resource and service management framework, which constitutes the central focus of this thesis.

2.4 Orchestration frameworks and technologies

The deployment and lifecycle management of applications across edge–cloud continuum environments depends on a layered stack of orchestration technologies

that collectively address containerisation, workload scheduling, microservice communication, standardised management interfaces, and infrastructure observability. Each technology layer solves a specific subset of the orchestration challenge, and understanding their architectural assumptions, capabilities, and limitations is essential for motivating the AI-driven resource and service management approaches developed in this thesis.

2.4.1 Containers as the ultimate deployment unit

Container technology has emerged as the dominant packaging and deployment unit for edge–cloud workloads, displacing virtual machines in most application scenarios due to its sub-second startup times, minimal memory and storage footprint, and architecture-agnostic image distribution [Pahl and Lee, 2015]. Docker popularised user-space containerisation and subsequently extracted its core runtime into *containerd*, a Cloud Native Computing Foundation (CNCF)-graduated project that, together with the low-level runtime *runc*, now underpins essentially every Kubernetes distribution through the Container Runtime Interface (CRI). The Open Container Initiative (OCI), founded in 2015, governs the three normative specifications — the runtime-spec, image-spec and distribution-spec — that allow images built once to be pulled, verified by content-addressable digest, and executed across heterogeneous edge and cloud nodes [Open Container Initiative, 2024]. The layered, immutable, copy-on-write structure of OCI images is particularly consequential at the edge: only delta layers need to traverse bandwidth-constrained links, and multi-architecture image indices reconcile the hardware heterogeneity — ARM, x86, RISC-V — typical of edge gateways and IoT-class devices.

2.4.2 Kubernetes and its edge-oriented derivatives

Kubernetes, the direct descendant of Google’s internal Borg and Omega cluster management systems [Verma et al., 2015, Burns et al., 2016], is the de facto orchestrator for containerised workloads. Its control plane comprises an API server, the Raft-consistent *etcd* key–value store, a scheduler, and a controller manager responsible for continuously reconciling the declared desired state with the observed system state. Worker nodes each run a *kubelet*, *kube-proxy*, and a CRI-compliant runtime. The Kubernetes Scheduler Framework, generally available since version 1.19, exposes filter, score, and bind extension points that make Kubernetes a preferred substrate for custom, AI-driven scheduling policies [Carrión, 2022]. However, vanilla Kubernetes embeds architectural assumptions that become problematic at the edge: the *etcd* quorum mechanism is intolerant of Wide Area Network (WAN) partitions, the controller manager evicts pods after sustained heartbeat failures, and the memory footprint of the control plane exceeds the resource budget of most far-edge devices [Kayal, 2020, Goethals et al., 2020].

Several CNCF-hosted distributions address these constraints with different design philosophies. *K3s* collapses the entire control plane into a single lightweight binary of approximately 70 MB, replacing *etcd* with an embedded SQLite store by default and removing legacy in-tree volume and cloud provider drivers. *KubeEdge* [Xiong et al., 2018] introduces a two-tier CloudCore/EdgeCore architecture with MQTT southbound connectivity, a device-twin abstraction for IoT device state management, and offline autonomy through a local metadata cache, enabling edge nodes to continue scheduling workloads during cloud disconnection. *OpenYurt* preserves upstream Kubernetes Application Programming Interfaces (APIs) entirely unchanged and layers edge autonomy through the YurtHub on-disk cache, a Node-Pool topology abstraction, and a Raven cloud-edge tunnel for secure cross-network communication. *SuperEdge* further adds distributed cross-node health checks that prevent false partition detection, and a ServiceGroup primitive that confines

microservice traffic within designated edge node units, avoiding unnecessary backhaul to the cloud.

2.4.3 Service meshes for microservice management

For applications decomposed into microservices, service meshes provide a dedicated infrastructure layer that decouples resilience, identity, and observability concerns from application code. *Istio* interposes Envoy sidecar proxies alongside each microservice container, with the consolidated Istiod control plane compiling routing rules, traffic policies, and mutual TLS certificates into xDS configuration consumed by these proxies [Calcote and Butcher, 2019]. This architecture enables weighted traffic splitting, circuit breaking, retry policies, and end-to-end encryption without any modification to application code. *Linkerd* pursues the same goals through a purpose-built Rust micro-proxy that achieves comparable functionality with substantially lower CPU and memory overhead — a decisive advantage in resource-constrained edge environments. Both meshes propagate W3C Trace-Context headers and export Prometheus and OpenTelemetry telemetry, providing the per-hop latency and error-rate signals that AI-driven placement and traffic management controllers consume. Limitations of both systems in continuum deployments include sidecar resource overhead, control-plane reachability over intermittent WAN links, and added L7 latency, which motivated Istio’s introduction of an ambient mode that substitutes a per-node tunnel process for per-pod sidecar injection.

2.4.4 ETSI MEC orchestration framework

At the standardisation level, the European Telecommunications Standards Institute (ETSI) defines the Multi-access Edge Computing (MEC) framework, specified in *ETSI GS MEC 003* [ETSI, 2024], which establishes a reference architecture in

which MEC hosts run applications and a MEC platform, governed by a MEC Orchestrator responsible for application onboarding, instantiation, relocation across hosts, and termination. A MEC Platform Manager handles the operational interface with the underlying virtualisation infrastructure, while a set of standardised reference points — Mp1 through Mp3 and Mm1 through Mm9 — define the interactions between MEC system components. Federation across operator domains is addressed through the MEC Federator and the complementary *ETSI GS MEC 040* specification. The MEC framework is particularly relevant to the edge–cloud continuum as it provides a vendor-neutral interface for application lifecycle management at the network edge, enabling AI-driven orchestration systems to issue placement, migration, and termination decisions through standardised APIs regardless of the underlying infrastructure vendor [Taleb et al., 2017].

2.4.5 Infrastructure monitoring and observability

Effective AI-driven orchestration depends critically on continuous, high-fidelity visibility into the state of the underlying infrastructure. The canonical observability stack rests on three complementary pillars: metrics, logs, and distributed traces [Sridharan, 2018]. *Prometheus* [Volz and Rabenstein, 2015], a CNCF-graduated time-series monitoring system, periodically scrapes metrics from instrumented targets over HTTP, stores them in a local time-series database, and exposes PromQL for derived signal computation and alerting. *Node Exporter* surfaces host-level hardware and operating system metrics, while *cAdvisor*, embedded within the Kubernetes kubelet continuously collects per-container CPU, memory, network, and filesystem utilisation data. *Netdata* [Netdata Inc., 2024] complements these tools by providing per-second, real-time metric collection with zero configuration overhead, making it particularly well-suited to resource-constrained edge nodes where lightweight, low-latency observability is essential and the operational cost of deploying and maintaining a full Prometheus scrape pipeline may be prohibitive.

Grafana provides visualisation and dashboarding over these time-series sources.

Distributed tracing, conceptually grounded in Google’s Dapper system [Sigelman et al., 2010], is operationalised in open-source deployments by *Jaeger* [CNCF, 2024] and *Zipkin* [OpenZipkin, 2024], with *OpenTelemetry* [OpenTelemetry Authors, 2024], the 2019 merger of the OpenTracing and OpenCensus projects, now serving as the converging vendor-neutral standard for instrumentation, collection, and export of metrics, logs, and traces. Together these tools yield the utilization, latency-percentile, throughput, and error-rate features that learning-based workload predictors, reinforcement-learning schedulers, and anomaly detectors consume as input. In edge deployments, however, raw metric shipping to a centralized collector is often infeasible due to bandwidth constraints and link intermittency. Edge-local pre-aggregation through Prometheus Agent mode, OpenTelemetry Collector, or lightweight log forwarders such as Fluent Bit [Fluent Bit Authors, 2024] is therefore necessary, introducing observability gaps that AI-driven controllers must be explicitly designed to tolerate.

2.4.6 The continuum gap

Each of the foregoing frameworks was engineered for a specific segment of the deployment spectrum. Kubernetes and Istio assume data-center Local Area Network (LAN) connectivity. KubeEdge, OpenYurt, K3s, and SuperEdge extend scheduling to intermittent edges but operate as separate management planes. ETSI MEC governs telco-grade infrastructure but interfaces awkwardly with cloud-native application orchestration. No mainstream system natively spans the IoT–edge–fog–cloud spectrum under a single declarative control plane. The conceptual aspiration, articulated variously as the *computing continuum* [Beckman et al., 2020, Dustdar et al., 2023], *osmotic computing* [Villari et al., 2016], and *Sky Computing* [Stoica and Shenker, 2021] — remains an open research agenda, with unresolved challenges persisting around hardware heterogeneity, intermittent connec-

tivity, latency- and energy-aware placement, zero-trust identity propagation, and data sovereignty constraints [Bittencourt et al., 2018, Rosendo et al., 2022, Moreschini et al., 2022].

Therefore, these technologies form the indispensable operational substrate for AI-driven service placement: containerization standardizes the deployment unit. Kubernetes and its edge-oriented derivatives provide the scheduling and lifecycle management primitives, service meshes enforce resilience and observability at the microservice level, ETSI MEC defines the telco-grade orchestration interfaces, and Prometheus, OpenTelemetry, and distributed tracing tools supply the continuous infrastructure visibility that any intelligent placement controller requires to make informed, real-time decisions. Without this stack in place, the runtime state information, actuation primitives, and feedback loops that AI-driven orchestration depends upon would simply not exist. These technologies therefore do not merely support the service placement problem, they define the environment within which it must be solved.

2.5 Optimization in decentralized systems

The service placement problem, as established in Section 2.3, is NP-hard and inherently multi-objective, making exact optimization infeasible at realistic scales. The decentralized, heterogeneous, and non-stationary nature of edge-cloud infrastructures further amplifies this complexity: node capabilities vary widely, workloads shift continuously, and the data required for globally informed decisions is distributed across geographically dispersed devices whose raw telemetry cannot always be aggregated centrally due to bandwidth, latency, and privacy constraints. This section surveys the principal families of optimization methods applicable to decentralized edge systems, progressing from classical approaches to the AI-driven mechanisms that form the methodological core of this thesis.

2.5.1 Classical optimization methods and their limitations

The earliest approaches to resource management in distributed systems were grounded in mathematical programming. ILP formulations model placement decisions as binary assignment variables subject to linear capacity, latency, and bandwidth constraints, and can produce provably optimal solutions [Taleb et al., 2025]. However, ILP is computationally intractable for NP-hard problems at the scale of realistic edge deployments: the number of binary variables grows combinatorially with the number of services and nodes, and the time required to solve even moderately sized instances renders ILP impractical for real-time or near-real-time decision-making [Bahrami et al., 2023]. Relaxed formulations, such as Linear Programming (LP) relaxations of ILP models, can be solved in polynomial time but sacrifice the integrality guarantees essential to valid placement configurations.

Convex optimization methods, including interior-point and gradient projection algorithms, are applicable where the objective function and feasible region satisfy convexity assumptions. While efficient solvers exist for convex programs, the combinatorial structure of placement decisions and the non-convex nature of end-to-end latency and QoE objectives preclude their direct application in most edge orchestration scenarios [Keshavarz Haddadha et al., 2024]. Game-theoretic formulations have also been proposed, modelling competing edge nodes or tenants as rational agents whose joint equilibrium strategies determine resource allocation [Ning et al., 2020]. Although these approaches yield elegant analytical results under idealized assumptions, they typically require complete information about all agents' utility functions and strategies — a condition that is rarely satisfied in practice under dynamic, partial-observability conditions.

The fundamental limitation shared by all classical methods is their assumption of a static or slowly changing environment. Because they solve a snapshot optimization problem, they are inherently reactive: a new solution must be computed from scratch whenever system state changes significantly, incurring com-

putational overhead that is incompatible with the millisecond-scale dynamics of real edge workloads. Moreover, they provide no mechanism for learning from historical execution data, meaning that recurrent patterns in workload behavior and resource contention cannot be exploited to improve future decisions.

2.5.2 Machine learning for workload prediction

A critical prerequisite for proactive placement decisions is the ability to anticipate near-future resource states before constraint violations occur. ML-based workload predictors address this requirement by learning statistical models of resource utilization dynamics from historical telemetry streams. Supervised learning approaches, including LSTM networks [Lindemann et al., 2021], Gated Recurrent Units (GRUs), and Temporal Convolutional Networks (TCNs), have demonstrated strong performance in modeling the temporal autocorrelations and periodicity patterns present in CPU, memory, and network utilization time series [Siarni-Namini et al., 2019a]. These models are trained on historical telemetry collected by the observability stack described in Section 2.4.5 and produce multi-step ahead forecasts that downstream placement controllers consume to anticipate overload conditions and trigger preemptive service migration before QoS targets are breached.

A critical challenge in deploying such predictors at the edge is *concept drift*: the statistical properties of workload time series are non-stationary and shift over time as usage patterns evolve, new services are deployed, or network topology changes [Lu et al., 2018]. Batch-trained models degrade silently under concept drift unless retrained periodically, but periodic retraining is costly and introduces service disruption. Incremental learning methods, which update model parameters continuously on each new observation without storing the full historical dataset, offer a more suitable alternative for resource-constrained edge nodes [Gepperth and Hammer, 2016]. Techniques such as Elastic Weight Consolidation (EWC) [Liu et al., 2020a] enable online adaptation while mitigating the *catastrophic forgetting phe-*

nomenon [Van de Ven et al., 2024], preserving previously acquired knowledge as the model adapts to new distributional regimes. The second proposed approach in this thesis employs in the initial stage, incremental learning at the node level to maintain up-to-date predictions of local resource availability under continuously shifting workload conditions.

2.5.3 Federated learning for privacy-preserving collaborative optimization

The bandwidth and privacy constraints of edge deployments preclude the centralized aggregation of raw telemetry data from distributed edge nodes. Federated Learning (FL), introduced by McMahan et al. [McMahan et al., 2017], addresses this constraint by training a shared global model through the exchange of model parameter updates rather than raw data. In the canonical synchronous FL protocol (FedAvg), a central server distributes a global model to participating clients, each of which performs local stochastic gradient descent on its private dataset and returns only the resulting parameter gradients or updated weights. The server aggregates these updates, most commonly through weighted averaging proportional to each client’s dataset size, and distributes the refined global model for the next round.

In the context of edge–cloud orchestration, FL enables geographically dispersed edge nodes to collaboratively refine a shared resource prediction model without exposing the raw telemetry streams that may contain sensitive user or application data [Zhang et al., 2021]. This is particularly relevant when edge nodes are operated by different stakeholders or span multiple administrative domains with distinct data governance policies. However, standard synchronous FL protocols introduce coordination overhead that can be prohibitive in large-scale edge deployments characterized by heterogeneous node capabilities and intermittent connectivity. The concept of *asynchronous* FL [Xu et al., 2023], in which nodes push up-

dates to the server independently as their local training completes, eliminates the synchronization barrier and is better suited to the variable-latency, partial participation conditions typical of edge environments. Building on this, the second tier of the resource-centric approach introduces an event-triggered asynchronous FL mechanism that activates selectively upon detected performance degradation, thereby minimizing unnecessary communication overhead while ensuring that the global model remains aligned with current system dynamics.

2.5.4 Reinforcement learning for sequential placement decisions

Unlike supervised learning, which requires labeled training data mapping system states to optimal placement decisions, data that is generally unavailable in practice, RL agents learn effective placement policies through direct interaction with the environment [Li, 2017]. The placement problem can be naturally formalized as a MDP, in which the state space encodes the current resource utilization of all edge nodes and the spatial distribution of active service requests, the action space comprises possible placement, migration, and scaling decisions, and the reward signal assesses the quality of each decision in terms of resource efficiency and QoE satisfaction [Naeem et al., 2020]. Deep Q-Network (DQN) [Mnih et al., 2013] extends tabular Q-learning to continuous and high-dimensional state spaces by approximating the Q-value function with a deep neural network, stabilizing training through experience replay and a periodically updated target network. While DQN has been successfully applied to service placement [Jie et al., 2020], [Luo et al., 2022], its performance degrades in high-dimensional action spaces because the operation over all possible actions becomes computationally prohibitive. PPO [Schulman et al., 2017a], a policy gradient method that directly parameterizes the placement policy as a neural network, sidesteps this limitation by optimizing the policy through a clipped surrogate objective that constrains the magnitude of each update, ensuring stable convergence without the fragility of earlier trust-region

methods. Empirical evaluations have demonstrated that PPO-based placement agents outperform both DQN and meta-heuristic baselines in terms of resource utilization balance and placement decision latency [Ajax et al., 2025], though they require careful reward shaping to simultaneously optimize multiple placement objectives. At the final tier, a centralized PPO agent serves as the placement decision engine, consuming forecasted resource states from the lower tiers to make coordinated, mobility-aware placement decisions across the full edge–cloud continuum. This design demonstrates that RL constitutes a suitable approach for addressing the dynamic, NP-hard placement problem with near-optimal solution quality.

2.5.5 Serverless computing and FaaS as a placement abstraction

An orthogonal perspective on the placement problem is offered by the serverless computing model, realized through FaaS platforms. Rather than managing the deployment and scaling of long-running containerized services, which introduces idle resource consumption particularly costly in resource-constrained edge devices, FaaS decomposes applications into fine-grained, stateless function units that are instantiated on-demand upon invocation and terminated immediately upon completion [Jonas et al., 2019]. This invocation-based resource model eliminates idle resource consumption and naturally accommodates the bursty, unpredictable workload patterns endemic to edge environments, making it particularly well-suited to resource-constrained edge nodes.

From an optimization standpoint, the serverless paradigm shifts the placement problem from the question of *where to deploy a service* to the question of *how to partition a service’s computational pipeline* into function units whose composition minimizes end-to-end response latency while remaining within the resource envelope of available edge infrastructure. This function partitioning problem has received limited attention in the literature: existing FaaS orchestration frameworks largely treat individual functions as isolated execution units and do not systemati-

cally optimize their inter-function composition for latency or QoE objectives [Eismann et al., 2020]. This gap is directly addressed by the service-centric approach of this thesis, which introduces an AI-driven Function Regulator that automatically determines the optimal number and grouping of serverless function blocks composing an ML/DL pipeline, leveraging runtime telemetry to identify the composition that minimizes end-to-end response time and maximizes user QoE.

2.5.6 Shifting towards closed-loop AI-driven orchestration

The methods surveyed in this section, reveal a clear progression in the design of optimization mechanisms for decentralized edge systems. Classical mathematical programming methods offer optimality guarantees but are computationally intractable and inherently static by their nature. Meta-heuristic algorithms improve scalability but restart the search from scratch for each new placement request and do not accumulate knowledge across decisions. Supervised ML predictors enable proactive placement by anticipating resource dynamics, but require labeled data and degrade under concept drift without continuous adaptation. FL extends these predictors to privacy-constrained multi-node settings without raw data aggregation. RL agents learn adaptive placement policies directly from environmental feedback, eliminating the need for labeled training data and enabling real-time inference without re-running the optimization. The serverless FaaS paradigm further refines the problem by abstracting the placement decision to the function composition level, enabling resource-efficient QoE-aware orchestration at fine granularity.

No single method dominates across all these dimensions simultaneously. The central contribution of this thesis is the design of two complementary AI-driven approaches to address the dynamic service placement problem from both the service composition and resource-centric perspectives. The precise problem formulation and the specific gaps in existing solutions that motivate these approaches

are further described in Chapters 3 and 4 respectively.

Chapter 3

Research Problem and Challenges

Chapter Structure

This Chapter is constructed as follows:

- **Section 3.1 – Problem definition**, formally introduces the dynamic service placement problem addressed in this thesis, characterises its infrastructure and application models, defines the constraint taxonomy and placement strategy taxonomy, and establishes the general optimization formulation within which the proposed approaches operate.
- **Section 3.2 – General problem formulation**, presents the general mathematical formulation of the Service Placement Problem (SPP) as it appears in the literature, independently of any specific deployment scenario, defining the placement function, the four constraint families, the mono-objective and multi-objective objective forms, and the complexity class of the problem.
- **Section 3.3 – Resource heterogeneity and scalability**, analyzes the challenge of deploying placement mechanisms across heterogeneous edge infrastructures with varying resource capacities and scaling requirements.
- **Section 3.4 – Latency and QoS constraints**, discusses the latency and QoS requirements imposed by latency-sensitive applications in edge–cloud envi-

ronments and how they constrain placement decisions.

- **Section 3.5 – Dynamic and uncertain environments**, examines the non-stationary nature of edge deployments, including workload variability, user mobility, and unpredictable resource availability, and their implications for adaptive placement mechanisms.

This Chapter formally grounds the research contributions of this thesis within a well-defined problem space. Building upon the architectural and conceptual foundations established in Chapter 2, the problem of dynamic service placement is here precisely formulated as a multi-objective optimization problem, and the key technical challenges that render classical solutions insufficient are systematically analyzed. Together, these challenges motivate the AI-driven approach developed in the thesis methodology, and establish the evaluation criteria against which the proposed approaches are assessed.

3.1 Problem definition

The service placement problem (SPP) constitutes one of the most extensively studied open challenges in distributed computing research. Its formal statement encompasses three interdependent components: the infrastructure model that describes the physical resources available for deployment, the application model that characterizes the computational and communication requirements of the services to be placed, and the deployment pattern that defines the mapping between applications and infrastructure nodes subject to a set of constraints and optimization objectives [Sfafi et al., 2024].

As established in Chapter 2, the SPP is NP-hard in its general form: even the feasibility problem of determining whether a valid placement exists satisfying all constraints is NP-complete, and the optimization problem of minimizing a composite placement objective subject to these constraints is at least as hard [He et al.,

2018], [Bahrami et al., 2023]. Classical approaches based on ILP produce provably optimal solutions under fixed, static assumptions but become computationally intractable as the number of services and nodes grows, and are fundamentally unsuited to the time-varying conditions of real edge deployments [Taleb et al., 2025]. Meta-heuristic methods improve scalability but restart the search from scratch for each new placement event, accumulate no knowledge across decisions, and are incapable of proactive adaptation to anticipated workload changes. These limitations collectively motivate the need for an adaptive, data-driven approach capable of making high-quality placement decisions in near-real time under dynamic and uncertain conditions.

3.1.1 Infrastructure model

The physical edge–cloud infrastructure consists of a heterogeneous collection of computational entities organized in a hierarchical topology. At the lowest tier, sensor and actuator devices possess no intrinsic computational capability and generate the data streams that higher tiers must process. At intermediate tiers, edge and fog nodes — including routers, gateways, base stations, cloudlets, and small-scale servers — possess varying degrees of computational, storage, and networking capacity. At the highest tier, cloud data centers provide effectively unbounded resources accessible over wide-area network links [Sfafi et al., 2024].

The infrastructure network is abstracted as a connected graph $G_{infra} = (V_{infra}, E_{infra})$, where vertices V_{infra} represent the set of IoT devices, edge nodes, and cloud servers, and edges E_{infra} represent the communication links between them. Each node $n_i \in V_{infra}$ is characterized by a multidimensional resource vector encompassing computational resources (CPU cores, clock frequency, Random Access Memory (RAM)), network resources (link type, bandwidth, latency, error rate), storage resources (available disk capacity), and hardware accelerators (Graphics Processing Unit (GPU), Field-Programmable Gate Array (FPGA)) where applicable. Due to

the physical structure of edge deployments, edge nodes are inherently resource-constrained and heterogeneous with respect to one another [Sfafi et al., 2024]. This heterogeneity prevents the application of uniform resource models and requires placement mechanisms that reason across multiple resource dimensions simultaneously.

3.1.2 Application model

Applications deployed across edge–cloud infrastructure can be characterized at three levels of structural complexity [Sfafi et al., 2024]. **Monolithic services** represent the simplest abstraction, treating the entire application as a single computational unit that must be placed on one physical node. **Sets of inter-dependent components** partition the application into a collection of functional modules without explicitly modeling the dependencies between them. **Connected graphs**, and specifically Directed Acyclic Graphs (Directed Acyclic Graph (DAG)s), constitute the most expressive and widely adopted model: vertices represent computational components and directed edges encode data-flow dependencies and communication demand between components.

The DAG topology is particularly well-suited to ML/DL analytics workloads whose sequential processing stages — data collection, cleaning, preprocessing, feature engineering, and model inference — map naturally onto a directed acyclic dependency graph. This is the application model adopted for the serverless function chain approach presented in Chapter 5, where the DAG vertices correspond to individual serverless function blocks and the edges encode the data transfer between pipeline stages.

Application resource requirements are specified across four dimensions. **Computational requirements** encompass the required CPU cores and RAM per component. **Network-oriented requirements** define the required bandwidth, maximum

tolerable latency, error rate, and jitter per inter-component link. **Task-oriented requirements** specify deadline constraints on the maximum permissible end-to-end processing time. **Location-oriented requirements** restrict certain components to specific geographical zones, fog node subsets, or co-location requirements with other components due to privacy, compliance, or hardware dependency constraints.

3.1.3 Deployment pattern and constraint taxonomy

The deployment pattern defines a mapping from application components to infrastructure nodes. A valid placement must satisfy a set of constraints that collectively define the feasible deployment space [Sfafi et al., 2024]:

Resource constraints (C_R): for each node n_i , the aggregate resource demand of all co-located application components must not exceed the node's available capacity:

$$\sum_{j: \varphi(a_j)=n_i} cpu_j^t \leq CPU_i^t, \quad \sum_{j: \varphi(a_j)=n_i} mem_j^t \leq Mem_i^t, \quad \forall n_i \in \mathcal{N}, t \in \mathcal{T}, \quad (3.1)$$

where $\varphi(a_j) = n_i$ denotes the placement of application a_j on node n_i .

Network constraints (C_N): inter-component communication links impose latency and bandwidth requirements on the physical network edges connecting nodes hosting dependent components.

Locality constraints (C_L): certain components are restricted to specific geographical zones, regulatory domains, or hardware platforms due to privacy, compliance, or dependency requirements.

Delay sensitivity constraints (C_D): latency-sensitive applications specify a deadline threshold D_j that the end-to-end processing time must not exceed, regardless of the placement configuration selected.

3.1.4 Placement Strategy Taxonomy

Existing placement strategies can be classified along four orthogonal dimensions [Sfafi et al., 2024].

Centralized vs. decentralized control. Centralized placement algorithms operate with global knowledge of application demands (global observability for deciding the placement) and infrastructure resources, potentially producing globally optimal solutions at the cost of scalability and single-point-of-failure vulnerability. Decentralized placement algorithms divide authority across multiple orchestrator nodes that make decisions based on local information, improving scalability and locality awareness but sacrificing global optimality guarantees.

Offline vs. online placement. Offline placement computes a deployment configuration at design time, assuming complete knowledge of system state. Online placement makes deployment decisions at runtime based on current system observations, accommodating dynamic workload changes but unable to guarantee optimal resource utilization.

Static vs. dynamic placement. Static strategies compute a fixed deployment configuration and do not revise it during system operation. Dynamic strategies continuously re-evaluate and update service locations in response to changes in infrastructure state, application requirements, or user distribution.

Mobility support. Placement strategies that explicitly model user and device mobility can proactively migrate services to follow the user population, while mobility-agnostic strategies may experience latency degradation as users move away from the nodes hosting their applications.

The approaches proposed in this thesis operate as **online, dynamic, centralized** placement mechanisms with explicit **mobility support**: placement decisions are made continuously at runtime based on current and predicted system state, the service configuration is updated in response to detected or anticipated changes

in resource availability and user proximity, and a centralized control plane with global observability produces coordinated placement decisions across the full edge–cloud infrastructure.

3.1.5 The two dimensions of the placement problem

The placement problem addressed in this thesis operates across two complementary dimensions, each corresponding to a distinct and consequential level of the service deployment architecture.

- ***Service composition dimension:*** concerns the internal structure of the service itself. When a service is realized as a pipeline of computational units — as is the case for ML/DL analytics workflows decomposed into a chain of serverless functions — the manner in which those units are grouped into discrete execution blocks directly governs end-to-end response latency through the data transfer overhead introduced at each inter-function boundary. This dimension is orthogonal to, and independent of, the infrastructure-level resource allocation, and has received limited attention in the existing literature. Chapter 5 addresses this dimension through the AI-based Function Regulator.
- ***Resource allocation dimension:*** concerns the assignment of services to physical edge nodes. Given a heterogeneous set of edge nodes with time-varying resource capacities and a set of services with dynamic resource demands, the objective is to identify, at each decision point, an assignment that jointly satisfies resource feasibility constraints, minimizes user-perceived latency, and maintains balanced resource utilization across the infrastructure. Chapter 6 addresses this dimension through a closed-loop, three-tier AI-driven orchestration approach.

3.2 General formulation of the service placement problem

The SPP is formally defined as the problem of finding an optimal or near-optimal mapping of application components onto a set of infrastructure nodes, such that the resource requirements of all components are satisfied, the constraints imposed by the infrastructure and the application are respected, and a specified objective function is optimized. This section presents a general formulation of the SPP that is independent of any specific deployment scenario or framework, and serves as the theoretical foundation upon which the system model and optimization problem of this thesis are grounded.

3.2.1 Notation and definitions

Let $\mathcal{G}_{infra} = (\mathcal{V}, \mathcal{E})$ denote the infrastructure graph, where $\mathcal{V}$ is the set of computational nodes, encompassing IoT devices, edge nodes, fog nodes, and cloud servers, and $\mathcal{E}$ is the set of directed communication links between them. Each node $v_i \in \mathcal{V}$ is characterized by a resource capacity vector:

$$\mathbf{c}_i = [c_i^{CPU}, c_i^{Mem}, c_i^{Bw}, c_i^{Store}], \quad (3.2)$$

where c_i^{CPU} , c_i^{Mem} , c_i^{Bw} , and c_i^{Store} denote the available CPU, memory, bandwidth, and storage capacity of node v_i respectively.

Let $\mathcal{G}_{app} = (\mathcal{S}, \mathcal{D})$ denote the application graph, where $\mathcal{S} = \{s_1, s_2, \dots, s_K\}$ is the set of application components (services) and $\mathcal{D} \subseteq \mathcal{S} \times \mathcal{S}$ is the set of directed data-flow dependencies between components. Each component $s_k \in \mathcal{S}$ has a resource demand vector:

$$\mathbf{d}_k = [d_k^{CPU}, d_k^{Mem}, d_k^{Bw}, d_k^{Store}], \quad (3.3)$$

and each dependency $(s_k, s_l) \in \mathcal{D}$ carries a communication demand b_{kl} in terms of

required bandwidth and a latency bound ϵ_{kl} on the inter-component data transfer.

3.2.2 Placement function

A placement φ is a function that maps each application component to a hosting infrastructure node:

$$\varphi : \mathcal{S} \rightarrow \mathcal{V}. \quad (3.4)$$

A placement is said to be *feasible* if and only if it satisfies all constraints imposed by the infrastructure, the application, and any applicable regulatory requirements. The set of all feasible placements is denoted Φ .

3.2.3 Constraint formulation

Four families of constraints define the feasible placement space Φ [Sfafi et al., 2024]:

Resource constraints (C_R). For each node $v_i \in \mathcal{V}$, the aggregate resource demand of all components co-located on that node must not exceed its available capacity across all resource dimensions:

$$\sum_{k: \varphi(s_k)=v_i} d_k^r \leq c_i^r, \quad \forall r \in \{CPU, Mem, Bw, Store\}, \quad \forall v_i \in \mathcal{V}. \quad (C_R)$$

Network constraints (C_N). For each dependency $(s_k, s_l) \in \mathcal{D}$, the physical path between $\varphi(s_k)$ and $\varphi(s_l)$ in $\mathcal{G}_{infra}$ must provide sufficient bandwidth and satisfy the inter-component latency bound:

$$\epsilon(\varphi(s_k), \varphi(s_l)) \leq \epsilon_{kl}, \quad \forall (s_k, s_l) \in \mathcal{D}. \quad (C_N)$$

Locality constraints (C_L). Certain components are restricted to a designated sub-

set of admissible nodes due to geographical, hardware, privacy, or regulatory requirements:

$$\varphi(s_k) \in \mathcal{V}_k^{loc}, \quad \forall s_k \in \mathcal{S}_{loc} \subseteq \mathcal{S}, \quad (C_L)$$

where $\mathcal{V}_k^{loc} \subseteq \mathcal{V}$ is the set of nodes to which component s_k may legally be assigned.

Delay sensitivity constraints (C_D). Applications with hard real-time requirements (critical delay-sensitive applications) impose an upper bound D on the end-to-end processing latency:

$$\sum_{(s_k, s_l) \in \mathcal{P}} \ell(\varphi(s_k), \varphi(s_l)) + \sum_{k \in \mathcal{P}} p_k \leq D, \quad (C_D)$$

where $\mathcal{P}$ denotes the critical execution path through the application DAG, p_k is the processing time of component s_k on its assigned node, and D is the application deadline.

3.2.4 Objective function

The SPP optimization objective is in general a function of the placement φ that quantifies the quality of the deployment configuration with respect to one or more performance criteria. The literature identifies four principal optimization metrics [Sfafi et al., 2024]: service latency, resource utilization, cost, and energy consumption. These metrics may be optimized individually in mono-objective formulations or can be jointly optimized in multi-objective formulations [Liu and Zhang, 2024] [Shi et al., 2025] .

In its general **mono-objective** form, the SPP is stated as:

$$\varphi^* = \arg \min_{\varphi \in \Phi} f(\varphi), \quad (3.5)$$

where $f : \Phi \rightarrow \mathbb{R}$ is a scalar objective function. Common instantiations of f include end-to-end service latency, total resource cost, and energy consumption.

In its general **multi-objective** form, the SPP is stated as:

$$\varphi^* = \arg \min_{\varphi \in \Phi} \mathbf{f}(\varphi) = [f_1(\varphi), f_2(\varphi), \dots, f_M(\varphi)], \quad (3.6)$$

where $\mathbf{f} : \Phi \rightarrow \mathbb{R}^M$ is a vector of M objective functions. Because multiple objectives are typically conflicting, reducing latency by concentrating services on proximity-optimal nodes tends to increase resource utilization at those nodes, while distributing load evenly tends to increase latency for some users, there is in general no single placement that simultaneously minimizes all objectives. The set of Pareto-optimal solutions $\Phi^* \subseteq \Phi$ consists of all placements for which no objective can be improved without degrading at least one other [Petrucchi et al., 2025]. A widely adopted practical approach is the **weighted sum scalarization**, which aggregates multiple objective functions into a single scalar through a convex combination:

$$\varphi^* = \arg \min_{\varphi \in \Phi} \sum_{m=1}^M w_m \cdot f_m(\varphi), \quad \text{s.t.} \quad \sum_{m=1}^M w_m = 1, \quad w_m \geq 0. \quad (3.7)$$

The weights w_m encode the relative importance of each objective and allow the operator to express a preference between competing performance criteria. For the two-objective case of latency minimization and resource utilization balance, the weighted sum reduces to:

$$\min_{\varphi \in \Phi} (a \cdot f_{\text{latency}}(\varphi) + \beta \cdot f_{\text{resource}}(\varphi)), \quad a + \beta = 1, \quad a, \beta \geq 0, \quad (3.8)$$

which is the form adopted and instantiated in the system model of this thesis, presented in Chapter 6, Section 6.2.

3.2.5 Complexity and resolution strategies

The SPP in its general form is NP-hard [He et al., 2018] [Bahrami et al., 2023]. The NP-hardness is established by reduction from the Bin Packing problem: the problem of assigning items of varying sizes to bins of fixed capacity such that no bin exceeds its capacity is NP-hard [Coffman et al., 1999], and the SPP generalizes this problem to multiple resource dimensions, inter-component dependencies, and network topology constraints. The hardness is further compounded in the dynamic formulation, where φ must be re-evaluated at each time step as node capacities and application demands evolve continuously.

Four principal resolution strategies have been proposed in the literature:

Exact methods — including ILP solvers and exhaustive enumeration — produce provably optimal solutions but are computationally intractable for problem instances with more than a small number of components and nodes, making them unsuitable for real-time placement in large-scale edge deployments [Al-Zobaie et al., 2025].

Approximation algorithms provide solutions with provable performance guarantees relative to the optimum, computed in polynomial time. They are theoretically well-grounded but difficult to design for multi-objective, multi-constraint formulations [Asgarian et al., 2024].

Heuristic and meta-heuristic methods, including GA [Kumar et al., 2010], ACO [Zarei et al., 2025], PSO [Rodríguez et al., 2021], and Tabu Search [Chen et al., 2021b], produce acceptable solutions within reasonable time but provide no performance guarantees and restart the optimization from scratch for each new placement event without accumulating knowledge across decisions.

AI-driven methods, including Deep Reinforcement Learning (DRL), Transfer Learning, Continual Learning, and FL, learn placement policies from experience and execute them in constant time at inference, enabling near-real-time adaptive place-

ment decisions that generalize across previously unseen system states [Keshavarz Hadadha et al., 2024]. These methods constitute the methodological core of the two approaches proposed in this thesis, and their detailed survey is presented in Chapter 4.

3.3 Resource heterogeneity and scalability

One of the most consequential structural properties of real edge–cloud deployments is the profound heterogeneity of the underlying infrastructure. Edge nodes are not uniform computational units but diverse devices spanning a wide spectrum of hardware capabilities: from resource-constrained single-board computers and base station processors at the far edge, through more capable gateway devices at the near edge, to the effectively unbounded resources of cloud data centers at the core [Shi et al., 2016]. This heterogeneity manifests across multiple resource dimensions simultaneously. Nodes differ in CPU architecture — ARM, x86, RISC-V — and in the number of available cores, clock frequency, and instruction set capabilities. Memory capacities range from hundreds of megabytes on far-edge devices to tens of gigabytes on near-edge servers. Storage type and capacity, network interface bandwidth, and operating system platform introduce further variation.

This multi-dimensional heterogeneity has direct and non-trivial implications for placement mechanisms. A placement model that assumes uniform node capabilities will systematically misallocate services: assigning a computationally intensive ML/DL inference service to a far-edge node that lacks the processing capacity to execute it within the application’s latency deadline is not merely suboptimal but results in a constraint violation that the simplified model cannot detect [Kazmi et al., 2025]. The resource state vector, which captures CPU, memory, and network metrics as independent dimensions of the node’s operational state, explicitly models this multi-dimensional nature. However, the feasibility constraint guards against static capacity violations in isolation, the dynamic interaction between co-located

services competing for shared resources introduces interference effects that per-resource capacity bounds alone cannot fully capture [Hong and Varghese, 2019]. A placement that satisfies the feasibility constraint for each service individually may still result in resource contention under high co-location density, making accurate performance prediction substantially harder.

The scalability challenge is equally consequential. As the number of edge nodes N and the number of applications A grow, the placement search space expands combinatorially: under the binary placement model where each application is assigned to exactly one node, the number of feasible configurations is N^A , which grows exponentially in both dimensions simultaneously. For realistic deployments with tens of edge nodes and hundreds of services, exact enumeration is computationally intractable [Bahrami et al., 2023]. This exponential growth directly motivates the adoption of approximate, learning-based placement mechanisms that trade global optimality guarantees for polynomial-time inference and continuous adaptation to changing conditions. The RL-based placement tier presented in Chapter 6 addresses this requirement by learning a policy that maps high-dimensional system state vectors to placement actions in constant time at inference, regardless of deployment scale.

The heterogeneity challenge also affects the normalized resource utilization model, which expresses utilization as a dimensionless fraction of each node’s available capacity. Because different nodes have different absolute resource capacities, the normalization by CPU_i^t , Mem_i^t , and $NetCap_i$ ensures that the utilization metric is expressed as a dimensionless fraction of each node’s available capacity rather than as an absolute value. This normalization is essential for making utilization values comparable across nodes of different capability levels, enabling the placement policy to reason about relative load rather than absolute consumption — a property that becomes particularly important when the edge infrastructure spans multiple orders of magnitude in raw resource capacity. Furthermore, heterogeneity

in CPU architecture introduces binary compatibility constraints: a container image compiled for x86 cannot execute on an ARM edge node without recompilation or emulation, adding a dimension to the feasibility constraint that is not expressible purely in terms of resource quantities [Carrión, 2022]. Practical placement mechanisms must therefore maintain architecture-compatibility metadata alongside resource availability information when evaluating candidate placement configurations.

3.4 Latency and QoS constraints

Latency constitutes one of the primary QoS dimensions for the majority of edge applications. The end-to-end latency experienced by a user requesting a service is a composite quantity encompassing three physically distinct components [Taleb et al., 2025].

- **Propagation delay:** determined by the physical distance between the user and the hosting node and the speed of signal propagation through the network medium. It is therefore a direct function of the user-to-node proximity.
- **Transmission delay:** depends on the size of the data payload and the available network bandwidth between the user and the node. It is directly influenced by the current network utilization level of the hosting node.
- **Processing delay:** determined by the computational load at the hosting node and the resource requirements of the application. It is a function of the node's current CPU and memory utilization.

Latency-sensitive applications impose strict deadline constraints on this composite quantity. Augmented and virtual reality applications (e.g., holographic displays, immersive gaming, and remote surgical assistance) require end-to-end response times below 20 milliseconds to prevent perceptible visual artifacts [Shi et al., 2016]. Autonomous vehicle control systems require sub-10 millisecond responses for safety-

critical maneuver decisions [KUMAR et al., 2025]. Cloud gaming demands consistent sub-50 millisecond frame delivery to maintain playable frame rates. Industrial Industrial Internet of Things (IIoT) control loops require deterministic sub-millisecond responses for closed-loop feedback control in manufacturing environments [Sari et al., 2020]. Violating these deadlines does not merely degrade QoE but can result in application failure or, in safety-critical contexts, physical harm.

The fundamental tension in the placement problem arises from the conflict between latency minimization and resource utilization balance. Placing all services on the nearest available edge node minimizes propagation delay for each individual user but concentrates workload at proximity-optimal nodes, driving them toward resource saturation and increasing processing delay as queuing effects accumulate. Distributing services evenly across all available nodes minimizes resource stress and processing delay but may assign some services to nodes that are geographically distant from the users requesting them, thereby increasing propagation delay. This three-way tension between propagation delay, processing delay, and resource saturation is the defining structural challenge of the latency-QoS-constrained placement problem and cannot be resolved by optimizing any single dimension in isolation [Maia et al., 2020].

Navigating this tension requires a placement objective that jointly accounts for both latency and resource utilization rather than optimizing either dimension in isolation. In practice, the relative importance assigned to each criterion should reflect the operational context of the deployment: latency-critical applications demand aggressive proximity-aware placement, while resource-constrained infrastructures require stronger emphasis on load balance to prevent saturation. A balanced treatment of both criteria reflects the operational reality of edge deployments where QoE and infrastructure stability must be maintained simultaneously, which constitutes the design principle adopted in the evaluation presented in Chapter 6.

A further dimension of the QoS challenge concerns the distinction between average-case and worst-case latency. Many placement objectives are formulated in terms of mean latency across all applications and time steps, but many latency-sensitive applications impose constraints on tail latency — the 90th or 99th percentile of the response time distribution — rather than the mean [Menon et al., 2024]. A placement policy that minimizes average latency may still produce occasional outliers that violate deadline constraints for individual requests. The evaluation of Chapter 6 accounts for this by reporting 90th percentile response times, providing a more conservative and practically relevant assessment of placement quality than mean-based metrics alone.

3.5 Dynamic and uncertain environments

The operational conditions under which placement decisions are made in real edge deployments are never stationary. This non-stationarity manifests across three orthogonal dimensions, workload variability, user mobility, and unpredictable resource availability, each of which independently renders static placement configurations suboptimal over time and collectively necessitates continuous, adaptive re-evaluation of placement decisions.

- **Workload variability** refers to the temporal fluctuations in the resource demands of deployed applications. Edge workloads are characteristically bursty: request arrival rates follow non-Poisson distributions with heavy-tailed inter-arrival times, and demand intensities exhibit strong diurnal, weekly, and event-driven periodicities [Hong and Varghese, 2019]. A service provisioned to handle peak demand will under-utilize its allocated resources during off-peak periods, wasting capacity on resource-constrained edge nodes, a service provisioned for average demand will become overloaded during peaks, violating QoS constraints precisely when user demand is highest. Sudden, unpredictable load spikes — triggered by news events, emergency broadcasts, or

viral content propagation — can saturate edge nodes within seconds, leaving no time for reactive placement responses that must recompute and deploy a new configuration from scratch. These dynamics motivate the forecasting-first design of the three-tier approach in Chapter 6, where IL-based resource predictors generate ahead-of-time warnings of impending saturation, enabling the RL placement agent to act proactively before constraints are violated rather than reactively after they have been breached.

- **User mobility** introduces a spatial dimension to the non-stationarity of the placement problem. As users move through the network, the distance between each user and the node hosting their requested application changes continuously, directly affecting propagation latency. A placement configuration that was proximity-optimal at time t may become latency-suboptimal at time $t + 1$ if a significant fraction of the user population has moved closer to a different edge node [Maia et al., 2020]. Proactively tracking and anticipating these mobility patterns requires the placement controller to incorporate user location dynamics into its decision process — a requirement reflected in the state space of the RL agent in Chapter 6, which includes both the current hosting node and the closest node to the current user population as explicit state dimensions. User coordinates are non-stationary across consecutive time steps, and any placement policy that ignores this temporal variation will incur systematic latency degradation as users drift away from the nodes to which their services are statically assigned.
- **Unpredictable resource availability** arises from node failures, hardware degradation, software crashes, and unexpected resource consumption by co-located background processes. Edge nodes are typically deployed in environments that are less controlled than cloud data centers — on street furniture, in industrial facilities, or on mobile platforms — and are therefore subject to a higher rate of partial or total failure events [Shi et al., 2016]. When a node's

effective capacity drops unexpectedly, any service currently hosted on that node may experience degraded performance or become unreachable, requiring emergency migration to an alternative node whose resources are sufficient to accommodate the transferred workload. This fault-induced migration must be completed before the application's QoS deadline is violated, imposing strict time constraints on the placement controller's response latency. The incremental learning tier in Chapter 6 addresses this challenge by continuously updating its resource predictions on streaming telemetry rather than relying on batch-trained models whose accuracy degrades silently following unexpected capacity changes.

The combination of these three sources of non-stationarity establishes a fundamental inadequacy of any placement strategy that does not continuously re-evaluate and adapt its decisions. A static placement computed at deployment time and never revised will accumulate increasing latency degradation as workloads shift, users move, and nodes fail or degrade. A reactive placement that responds only after QoS violations have occurred cannot prevent the violation itself — only remediate it after the fact — which is insufficient for applications with hard deadline constraints. Only a proactive, adaptive placement mechanism that anticipates future system states and acts before constraints are breached can maintain QoS guarantees under the continuously evolving conditions of real edge deployments. This requirement constitutes the central design principle underlying both approaches proposed in this thesis. The AI-driven approaches that realize it are presented in Chapters 5 and 6 respectively.

Chapter 4

Related Work

Chapter structure

This chapter surveys the state of the art across the two research dimensions that underpin the contributions of this thesis: the intersection of serverless computing and ML/DL workload deployment, and the application placement problem in edge–cloud environments. For each dimension, the principal works are reviewed, their strengths and limitations are assessed, and the specific gaps that motivate the contributions of Chapters 5 and 6 are identified.

- **Section 4.1 – Serverless computing and ML/DL workload deployment**, covers the intersection of serverless computing and ML/DL workloads, encompassing FaaS-based deployment frameworks, data analytics pipelines, and the challenges of adapting serverless architectures to support computationally intensive AI applications.
- **Section 4.2 – Application placement in edge–cloud environments**, covers the application placement problem in edge–cloud environments, encompassing orchestration mechanisms, scheduling strategies, and AI-driven approaches to dynamic resource allocation.
- **Section 4.3 – Summary and research gaps**, synthesizes the identified gaps

across both dimensions and positions the contributions of Chapters 5 and 6 within the existing landscape.

4.1 Serverless computing and ML/DL workload deployment

The convergence of serverless computing and ML/DL has attracted substantial research attention, driven by the operational advantages of the FaaS model — zero-idle resource consumption, invocation-based billing, and infrastructure abstraction — and the growing demand for agile, cost-effective deployment of data-intensive AI workloads. The serverless paradigm is particularly well-suited to edge application deployment for three interconnected reasons. First, edge nodes operate under stringent resource constraints and cannot afford the idle capacity overhead introduced by traditional reservation-based deployment models, the invocation-based resource model of FaaS platforms ensures that computational capacity is consumed only when user requests actively trigger service execution, making it inherently compatible with the resource budget of edge infrastructure [Baldini et al., 2017]. Second, edge workloads are characteristically bursty and unpredictable, with demand patterns that vary substantially across time and space, the elastic, on-demand instantiation of serverless functions naturally accommodates these fluctuations without requiring pre-provisioned capacity or manual scaling intervention [Shafiei et al., 2022]. Third, the abstraction of infrastructure management from application logic is particularly valuable at the edge, where developers and data scientists deploying ML/DL services should not be required to manage the operational complexity of distributed, heterogeneous hardware [Jonas et al., 2019]. The literature in this area spans three interconnected threads: the provision of ML capabilities as serverless services, the execution of computationally intensive ML/DL training and inference workflows through FaaS platforms, and the deploy-

ment of AI workloads at the network edge using serverless architectures.

4.1.1 Machine learning as a serverless service

Early efforts in this area explored the viability of exposing ML functionality as a serverless service, offloading the infrastructure management burden from data scientists and developers. Representative work exposed ML capabilities for user behaviour analysis across varied demographic contexts in a serverless manner, leveraging the Watson ML API on IBM Cloud [Baldini et al., 2017]. This approach exemplifies the core benefit of the serverless model for ML practitioners: the developer is concerned exclusively with data analysis, processing, and evaluation, while resource allocation, scaling, and runtime management are handled transparently by the platform.

The suitability of the serverless model for data-intensive applications was further demonstrated through its application to MapReduce workloads. Research exploiting serverless architecture for executing MapReduce jobs using Amazon Web Services (AWS) Lambda and Amazon S3 as the storage backend established that FaaS platforms provide a viable computing substrate for high-throughput computing jobs [Baldini et al., 2017]. However, the same work identified a critical limitation: inhomogeneous performance behavior under tightly coupled computing jobs, a finding that underscores the need for careful pipeline composition when deploying ML workflows in serverless environments.

4.1.2 Serverless frameworks for ML/DL pipelines

The research community has developed several dedicated frameworks to address the challenges of executing ML/DL pipelines through serverless architectures. PyWren [Sampé et al., 2018] constitutes a map-reduce framework specialized for serverless architectures that serializes Python source code and dispatches it to

AWS Lambda functions [Amazon Web Services, 2024b] for massively parallel execution, using Amazon S3 [Amazon Web Services, 2024a] as an intermediate storage layer. PyWren is particularly relevant to the training phase of ML pipelines, where it overcomes the stateless nature of serverless functions by using external storage for intermediate computation results, a design pattern that informs the two-phase offline/online architecture adopted in Chapter 5.

A related approach applies serverless technology to distributed ML training, avoiding operational costs by using concrete functions as distributed workers and a coordinating function as the FL server, again relying on external storage for result retention and aggregation [Jiang et al., 2021]. This work demonstrates that data parallelism aligns naturally with serverless architectures: multiple stateless functions can simultaneously process different data batches without requiring server management, substantially reducing system configuration overhead.

The SIREN framework [Wang et al., 2019] constitutes an asynchronous distributed ML framework based on serverless architecture that leverages a collection of stateless AWS Lambda functions, each operating on a distinct data batch. SIREN achieves higher parallelism and elasticity while significantly reducing system configuration expenses, and its experimental results demonstrate a reduction in ML model training time of up to 44% compared to traditional ML benchmarks at equivalent cost. This result establishes a strong empirical case for serverless-based ML pipeline execution and directly motivates the online inference pipeline design presented in Chapter 5.

Resource management for DL services under dynamic workloads has been addressed through a serverless-based framework that hosts containerized, pre-trained DL models for predictive analytics in cloud environments [Bhattacharjee et al., 2019b]. The framework functions as a data-driven resource allocator that estimates resource requirements ahead of time by exploiting incoming request patterns and a forecast-aware scheduling mechanism, improving resource utilization

while preventing physical resource exhaustion. This work is directly relevant to the IL-based forecasting approach adopted in the first tier of the framework presented in Chapter 6. A further contribution introduced a serverless framework specialized in the agile deployment and scheduling of data analytics processes, live-streaming applications, and ML-as-a-service inference tasks [Bhattacharjee et al., 2019a]. The proposed platform provides a native domain-specific modeling language (Domain-Specific Modelling Language (DSML)) and a user interface with higher-level abstractions, enabling ML developers to create and evaluate models using standard libraries without managing the underlying infrastructure. This work consolidates the emerging consensus that serverless architecture is a promising and practical substrate for ML/DL workloads across cloud–edge environments.

4.1.3 Edge AI and serverless architectures

While the frameworks surveyed in the preceding subsection demonstrate the viability of serverless ML/DL pipeline execution in cloud environments, their deployment assumptions — reliable connectivity, abundant storage, and low-latency access to managed services — are not satisfied in edge settings. The integration of AI approaches into the serverless model for edge deployment has therefore emerged as a distinct and active research direction, driven by the exponential growth of data generated at the network periphery and the need to execute AI inference closer to end users. This integration has taken several forms: the embedding of AI lifecycle management concepts into serverless platforms to simplify edge AI workflow development, the adaptation of serverless execution models to support distributed ML workloads across heterogeneous edge–cloud nodes, and the application of optimization techniques to overcome the resource and runtime constraints that FaaS platforms impose on computationally intensive AI workloads.

Research has proposed serverless platforms specifically designed for building and

operating edge AI applications, analyzing edge AI use cases to illustrate the challenges of building and operating AI applications in edge–cloud scenarios [Rausch et al., 2019]. By integrating concepts from AI lifecycle management into the serverless model, this work demonstrates that the development of edge AI workflow functions can be substantially simplified, reducing the barrier between AI model development and edge deployment.

The STOIC framework constitutes a system for executing distributed ML applications in IoT edge and cloud settings based on serverless architecture, directly addressing the deployment of computationally intensive ML workloads across heterogeneous edge–cloud nodes [Zhang et al., 2020]. STOIC reflects the growing convergence of AI approaches with the serverless model in edge environments and provides a reference architecture for the type of pipeline-based deployment explored in Chapter 5.

The integration of AI into serverless architectures has also exposed a critical practical barrier: constraints on code size, memory, and execution duration have traditionally limited the viability of deploying large DL models within FaaS platforms [Wen et al., 2021]. Leveraging vendor-specific services, researchers demonstrated the deployment of large AI model-based applications in a serverless setting and conducted performance comparisons against on-premises deployment, establishing that these constraints are a recognized and actively addressed challenge in the serverless AI deployment literature — and directly motivating the container-extended runtime mechanism presented in Section 5.2.3.

Techniques for optimizing serverless AI workloads have been developed to address the challenges of deploying production AI workloads within the constraints of FaaS platforms. These techniques encompass library slimming and selective packaging, dynamic loading of pre-trained models from cloud storage, and optimizations for large-volume data lookup within function lifetime limits — an optimization direction that is directly complementary to the approach adopted in

Section 5.2.3.

4.2 Application placement in edge–cloud environments

The application placement problem in edge–cloud environments has been extensively studied from multiple perspectives, encompassing infrastructure-level orchestration mechanisms, optimization-based formulations, and increasingly, AI-driven adaptive approaches. The literature reviewed in this section is organized around two complementary dimensions: orchestration mechanisms for managing edge application lifecycles, and placement strategies for optimally distributing application components across heterogeneous infrastructure.

4.2.1 Orchestration mechanisms

Cloud–edge orchestration solutions must achieve two principal objectives: efficient workload distribution under dynamic conditions, and optimal resource allocation across the cloud–edge continuum while satisfying multi-objective requirements including energy efficiency, resilience, balanced utilization, and user satisfaction.

Kubernetes has emerged as the state-of-the-art container orchestration platform, offering high availability, scalability, and fault tolerance for managing cloud–edge applications [Kubernetes,]. However, extending Kubernetes capabilities to heterogeneous edge environments remains an active research challenge. The FLEDGE framework proposes a Kubernetes-compatible approach enabling secure orchestration across heterogeneous edge devices through virtual kubelets and VPN-based communication, achieving low resource overhead [Goethals et al., 2022]. While effective in addressing the security and compatibility dimensions of edge orchestration, this work does not incorporate AI-based scheduling or predictive mechanisms, which limits its adaptability to non-stationary workload conditions.

A learning-based scheduling framework combining Multi-Agent Actor-Critic (Multi-Agent Actor-Critic (MAAC)) [Konda and Tsitsiklis, 1999] and Graph Neural Networks (GNNs) has been proposed for improving edge–cloud scheduling, demonstrating throughput gains on real workloads [Han et al., 2021]. This work advances the state of the art by incorporating decentralized ML-based scheduling, yet it does not address workload forecasting or collaborative learning across nodes, leaving the proactive adaptation dimension unresolved.

Context-aware scheduling for Industrial IoT environments has been addressed through a cognition-based scheduler that integrates physical, operational, and network parameters [Chima Ogbuachi et al., 2020]. While the runtime adaptability of this approach is a notable strength, the absence of learning-based prediction or global optimization limits its effectiveness in highly dynamic and unpredictable edge deployments.

Federated multi-cloud Kubernetes orchestration with predictive analytics for proactive scaling and fault tolerance has been proposed to improve scalability and resilience across multi-cloud deployments [Kumar, 2025]. However, this work is primarily cloud-focused rather than edge-oriented, and does not address the resource-constrained, geographically distributed conditions characteristic of edge infrastructure. A holistic cognitive orchestration framework for sustainable and resilient workload distribution across the edge–cloud continuum using AI-driven orchestration and heuristic scheduling has been presented in [Sofia et al., 2024], though this framework does not incorporate a closed-loop IL/FL learning pipeline or RL-based placement decisions.

Energy-efficient task scheduling in cloud environments using DRL has been extensively surveyed [Hou et al., 2024], establishing that DRL-based approaches consistently demonstrate superior performance in reducing energy consumption compared to traditional scheduling heuristics. This body of work provides important context for the RL-based placement tier presented in Chapter 6, confirming that

policy-gradient methods are particularly well-suited to sequential scheduling decision problems in cloud-edge environments.

4.2.2 Application placement strategies

The optimal placement of application components across edge resources constitutes a non-convex optimization problem due to multiple constraints, nonlinear system correlations, and the combinatorial complexity of the placement action space [Chen et al., 2020]. Existing approaches address this problem under various objectives, considering execution environment characteristics, optimization targets, resource constraints, and placement strategy type — whether static, dynamic, heuristic, or learning-based.

Classical optimization approaches have been applied to the placement problem with strong theoretical guarantees. Application-aware offloading and resource allocation using branch-and-bound optimization achieves near-optimal performance in static environments [Lin et al., 2023], while a two-stage offloading strategy for minimizing processing delay through joint workload distribution and execution ordering has been proposed for MEC scenarios [Gao et al., 2020]. Both approaches produce high-quality solutions but are fundamentally static optimizers that cannot adapt to the non-stationary, dynamically evolving conditions characteristic of real-world edge deployments.

Classical heuristic placement strategies including First-Fit and Best-Fit Bin Packing have been widely applied to the VM and container placement problem [Li et al., 2014]. While computationally efficient, these methods rely exclusively on single-snapshot resource availability, do not account for user proximity or network utilization, and cannot adapt to the multi-resource constraints present in dynamic edge deployments — limitations that are empirically quantified in the comparative evaluation of Chapter 6.

The application component placement problem in cloud environments has been formulated as a Markov Decision Process and addressed using RL-based approaches in [Paraskevoulakou et al., 2024], which combines DRL, graph-based combinatorial optimization, and metaheuristic techniques to identify optimal component-to-VM groupings. This work established that PPO substantially outperforms DQN as application complexity increases and that optimal placement can be achieved in fewer than ten episode steps on average. It constitutes a direct predecessor of the RL-based placement tier presented in Chapter 6, extending its scope from the cloud-only VM context to the more demanding edge–cloud continuum setting.

AI-driven approaches to edge service placement have been further developed through DRL-based service placement frameworks for cyber-physical systems, demonstrating adaptive placement capabilities under dynamic infrastructure conditions [Hao et al., 2020]. However, this work employs a single-layer learning architecture without forecasting or federated refinement mechanisms, limiting its robustness under non-stationary workload patterns. Graph-based DRL task scheduling using GNNs has been proposed for efficient distributed scheduling in edge networks [Xiao et al., 2025], enabling proactive orchestration under dynamic workloads, though this approach relies on centralized training without FL or IL mechanisms.

The need for adaptive scheduling in heterogeneous cloud–IoT–edge systems has been highlighted in [Menon et al., 2024], which emphasises that modern schedulers must handle workload variability, heterogeneous resources, and strict latency constraints. This work motivates the intelligent and adaptive placement mechanisms developed in Chapter 6, while its limited QoS scope and assumption of independent tasks leave the multi-resource, user-proximity-aware placement problem unaddressed.

Service migration policy methods based on DQN for MEC environments have also been developed, demonstrating superior performance compared to traditional methods and improved resilience to migration faults [Rui et al., 2021]. DRL-based

VM placement in cloud data-centers addressing the trade-off between datacenter performance, security requirements, and power costs has further been demonstrated to be effective under workloads with major fluctuations [Caviglione et al., 2021].

The network-aware placement of application components in cloud environments has been addressed through an algorithm based on Markov Decision Process theory and linear programming, identifying solutions that minimize inter-component communication costs while maximizing available resources [Barshan and Sharafat, 2017]. Application component placement in NFV-based hybrid cloud/fog systems has been formulated as an Integer Linear Programming problem and solved using a Tabu-based algorithm, demonstrating near-optimal performance at small scales and improved makespan at larger scales [Mouradian et al., 2019]. Data-flow-aware placement of cloud application components, positioning processing elements as close as possible to the data sets they consume, has also been proposed to improve distributed application performance [Zimmermann and Koziol, 2020].

4.2.3 Approaches to service placement by using the serverless paradigm

A specific and relatively underexplored dimension of the placement problem concerns the deployment of services as serverless function chains, where the internal composition of the service — the grouping of computational tasks into function blocks — constitutes an independent placement decision at the service composition level. This perspective has been addressed in preliminary form in the cloud-based VM context by [Paraskevoulakou et al., 2024], which demonstrated that the grouping of application components directly governs application performance independently of the hosting infrastructure. The serverless extension of this insight — that the partitioning of an ML/DL pipeline into chained serverless functions governs end-to-end response latency independently of which edge node hosts the

pipeline — constitutes the central contribution of Chapter 5 and has not been addressed in the prior literature reviewed above.

The broader question of optimal function composition in serverless architectures has been recognized as a research challenge in the serverless computing literature. The Serverless Trilemma [Baldini et al., 2017] identifies function composition as a fundamental tension in serverless system design, establishing that simultaneously achieving function independence, arbitrary composition, and observable intermediate state is impossible under the stateless execution model. This theoretical constraint motivates the empirical investigation of Chapter 5, which demonstrates that the practical consequence of this tension, variable response time as a function of pipeline composition, can be systematically addressed through an AI-driven optimization mechanism.

Automated configuration of serverless function parameters to minimize cost while satisfying user-specified performance criteria has been explored in [Akhtar et al., 2020], and memory size optimization for serverless functions has been addressed through resource sizing techniques [Eismann et al., 2021]. However, neither work addresses the question of how many function blocks should compose a pipeline, nor how those blocks should be grouped to minimize end-to-end response time — the specific gap that the AI-based Function Regulator of Chapter 5 is designed to address.

4.3 Summary and research gaps

Table 4.1 presents a consolidated comparison of the most relevant works reviewed in this chapter, summarising their focus, strengths, and limitations relative to the contributions of this thesis.

The survey presented in this chapter reveals two distinct and complementary research gaps that the contributions of this thesis are designed to address.

The **first gap** concerns the service composition dimension of the placement problem in serverless ML/DL deployments. The existing literature on serverless ML/DL frameworks addresses the execution of analytical pipelines through FaaS platforms but does not investigate the relationship between function pipeline composition — the number and grouping of function blocks — and end-to-end response latency. Works such as PyWren [Sampé et al., 2018] and SIREN [Wang et al., 2019] treat the pipeline as a fixed structure, while approaches addressing serverless function configuration [Akhtar et al., 2020] and memory sizing [Eismann et al., 2021] optimize individual function parameters without considering the pipeline partitioning problem. The AI-based Function Regulator presented in Chapter 5 addresses this gap by establishing, for the first time, that the optimal number and grouping of serverless functions in an ML/DL pipeline is a non-trivial empirical property that requires systematic AI-driven optimization, and that this optimization constitutes a distinct and consequential dimension of the service placement problem.

The **second gap** concerns the absence of closed-loop, proactive, and privacy-preserving orchestration frameworks for dynamic application placement in edge–cloud environments. None of the reviewed works integrates local incremental forecasting to capture concept drift, asynchronous federated refinement to improve prediction robustness without raw data exchange, and centralized RL-based placement to exploit global observability within a unified, closed-loop pipeline. The three-tier orchestration approach presented in Chapter 6 addresses this gap by providing the first integrated design that combines all three mechanisms, IL, FL, and RL, into a cohesive, event-driven orchestration architecture for proactive and resource-aware application placement across the edge–cloud continuum.

Table 4.1: Consolidated comparison of related work across the serverless ML/DL deployment and edge-cloud application placement dimensions. Limitations are assessed relative to the research gaps addressed by the contributions of Chapters 5 and 6.

Work	Focus	Strengths	Limitations
PyWren [Sampé et al., 2018]	Serverless MapReduce for ML	High parallelism, reduced training time	No function partitioning optimization, external storage dependency
SIREN [Wang et al., 2019]	Async distributed ML on FaaS	44% training time reduction, elastic scaling	No edge deployment, no pipeline composition optimization
STOIC [Zhang et al., 2020]	Serverless ML for edge/cloud IoT	Edge-cloud ML distribution	No function partitioning, no AI-based composition optimization
Akhtar et al. [Akhtar et al., 2020]	Serverless function configuration	Cost-performance optimization	No pipeline cardinality or grouping optimization
Eismann et al. [Eismann et al., 2021]	Serverless memory sizing	Automated resource sizing	No function composition or placement optimization
Baldini et al. [Baldini et al., 2017]	Serverless function composition	Theoretical composition framework	No empirical optimization, no ML/DL context
Goethals et al. [Goethals et al., 2022]	Edge container orchestration	Low footprint, secure, k8s compatible	No AI scheduling or prediction
Han et al. [Han et al., 2021]	Edge-cloud workload scheduling	Decentralized, higher throughput	No workload forecasting or FL
Ogbuachi et al. [Chima Ogbuachi et al., 2020]	Context-aware edge scheduling	Runtime adaptive decisions	No learning, forecasting, or global optimization
Kumar et al. [Kumar, 2025]	Multi-cloud orchestration	Scalable, portable, multi-cloud ready	Not edge-focused, no learning prediction
Sofia et al. [Sofia et al., 2024]	Cognitive decentralized orchestration	Holistic design, sustainability focus	No IL/FL closed-loop or RL placement
Lin et al. [Lin et al., 2023]	Cloud resource allocation	High packing efficiency	Static optimization, no adaptive learning
Gao et al. [Gao et al., 2020]	MEC delay optimization	Low delay, efficient scheduling	No prediction or distributed learning
Li et al. [Li et al., 2014]	Bin packing placement	Computationally efficient	Static, ignores proximity and network
Paraskevoulakou et al. [Paraskevoulakou et al., 2024]	Cloud-edge VM placement	Near-optimal RL-based placement	Cloud-only, no IL, FL, or edge context
Hao et al. [Hao et al., 2020]	Edge service placement	Adaptive placement in Cyber-Physical System (CPS)	Single-layer learning, no forecasting or FL
Xiao et al. [Xiao et al., 2025]	GNN edge task scheduling	Efficient distributed graph scheduling	Centralized training, no FL or IL
Menon et al. [Menon et al., 2024]	Task scheduling in cloud/IoT/edge	Better load balance, faster scheduling	Limited QoS scope, assumes independent tasks

Chapter 5

An AI-driven service-centric approach to dynamic service placement via the serverless paradigm

The research presented in this chapter has been published in preliminary form in: E. Paraskevoulakou and D. Kyriazis, “Leveraging the Serverless Paradigm for Realizing Machine Learning Pipelines across the Edge–Cloud Continuum,” 24th Conference on Innovation in Clouds, Internet and Networks (ICIN 2021), pp. 110–117, March 2021 [Paraskevoulakou and Kyriazis, 2021], and in its full extended form in: E. Paraskevoulakou and D. Kyriazis, “ML-FaaS: Toward Exploiting the Serverless Paradigm to Facilitate Machine Learning Functions as a Service,” IEEE Transactions on Network and Service Management, vol. 20, no. 3, pp. 2110–2123, September 2023 [Paraskevoulakou and Kyriazis, 2023].

Chapter structure

This Chapter is constructed as follows:

- **Section 5.1 – Motivation and service-centric perspective**, establishes the ra-

tionale for approaching the dynamic service placement problem from the service composition level, and identifies the limitations of exclusively infrastructure-centric views that motivate the serverless FaaS paradigm as a complementary placement abstraction.

- **Section 5.2 – Proposed approach: ML-FaaS**, introduces the two-phase ML-FaaS approach as a proof-of-concept realization of the service-centric placement approach, demonstrating how a complete ML/DL inference workflow can be decomposed into a chain of serverless functions deployed across edge-cloud infrastructure. The offline data analytics and model training phase and the online serverless function pipeline are described in detail, together with the container-extended runtime mechanism that overcomes the code-size constraints inherent to FaaS platforms.
- **Section 5.3 – AI-based function regulator**, presents the central contribution of this chapter: the AI-driven approach that automatically determines the optimal number and grouping of serverless function blocks composing an ML/DL pipeline, with the objective of minimizing end-to-end response time and maximizing user QoE, taking the ML-FaaS proof-of-concept introduced in Section 5.2 as the evaluation case study.
- **Section 5.4 – Evaluation**, describes the experimental setup and presents a comparative performance assessment across multiple pipeline configurations and candidate DL architectures.
- **Section 5.5 – Summary**, synthesizes the contributions of this chapter and establishes the transition to the resource-centric approach developed in Chapter 6.

5.1 Motivation and service-centric perspective

The dynamic service placement problem, as established in Chapter 3, is conventionally approached from the perspective of the underlying infrastructure: given a heterogeneous set of edge nodes with known resource capacities and a set of services with known computational demands, the objective is to identify an assignment of services to nodes that jointly satisfies resource constraints and QoS requirements. This infrastructure-centric formulation is well-grounded and captures the combinatorial complexity of the placement problem with precision. However, it implicitly treats the service itself as a fixed, monolithic entity whose internal structure lies outside the scope of the placement optimizer. This assumption becomes increasingly untenable in modern edge–cloud deployments, where ML and DL services are inherently multi-stage in nature, encompassing a sequence of interdependent operations including data collection, cleaning, preprocessing, and feature preparation, all of which must be executed in order before the model can be invoked for inference. These operations are naturally realized as sequential computational pipelines whose internal composition directly and non-trivially governs end-to-end response latency, which is the primary determinant of user QoE.

The relationship between service structure and placement outcome is not just incidental. As data traverses a pipeline of interconnected functions, each inter-function boundary introduces a data transfer overhead that accumulates across the entire execution chain. The manner in which a service’s computational steps are grouped into discrete execution units therefore has a direct and measurable effect on the latency experienced by end users, independently of which specific edge node hosts the pipeline. This observation motivates a complementary perspective on the placement problem: rather than asking exclusively *where* to deploy a service, it is equally necessary to ask *how* the service’s internal pipeline should be partitioned into function units whose composition minimizes end-to-end latency

while remaining within the resource constraints of the available edge infrastructure.

Modern edge applications are subject to a dual imperative: they must be placed on nodes whose available computational resources are sufficient to sustain their execution, while simultaneously fulfilling user demand in terms of response latency and service continuity. The breadth of application domains that depend on this balance is considerable. In the autonomous vehicles domain, onboard systems integrate data streams from cameras, radar, lidar, and additional sensors [Zhao and Tang, 2020], generating between five and twenty terabytes per day, and must perform safety-critical inference tasks including object detection [Arnold et al., 2019], lane keeping [Chen and Huang, 2017], and collision avoidance [Sathiyamurthy and Naidu, 2025] instantaneously and independently of cloud connectivity. Any latency introduced by offloading these computations to a remote server would directly compromise vehicle safety, making edge-local inference not merely preferable but operationally necessary [Chen et al., 2021a]. In industrial and manufacturing environments, dense networks of IIoT sensors continuously monitor machinery, production lines, and remote infrastructure such as offshore platforms, where real-time fault detection, predictive maintenance, and anomaly identification must operate without dependence on backhaul connectivity [Sari et al., 2020]. The ability to localize inference in these settings also satisfies data residency requirements and protects proprietary operational data from external exposure [Hou et al., 2021]. In smart healthcare, AI-driven monitoring systems embedded in wearable sensors, portable diagnostic devices, and smart medical equipment analyse patient vitals and detect clinical irregularities at the point of care, enabling faster time-to-intervention while ensuring that sensitive health data is processed locally in compliance with regulatory constraints [Balakrishnan et al., 2025]. This is particularly consequential in distributed or low-connectivity environments such as rural clinics and emergency response scenarios, where dependence on cloud processing would introduce unacceptable delays. In smart city

deployments, AI-driven systems process data from IoT sensors, traffic cameras, and public infrastructure in real time to dynamically manage traffic flow, monitor environmental conditions, and respond to safety incidents, with local inference ensuring immediate reaction even under intermittent cloud connectivity [Krislata et al., 2020]. Across all these domains, the common operational constraint is identical: the service must be placed on an edge node with sufficient computational capacity while concurrently delivering responses within the latency bounds that the application's QoE requirements demand.

This joint requirement, resource feasibility on the node side and QoE satisfaction on the user side, defines the fundamental tension that any effective placement strategy must resolve. A critical dimension of this tension, however, arises not only from the allocation of services to nodes but from the manner in which the services themselves are designed and deployed. Traditional deployment models, whether based on virtual machines or long-running containerized instances, allocate computational resources statically and maintain them in a reserved state regardless of whether user requests are actively being processed [Jonas et al., 2019]. In resource-constrained edge environments, this idle resource consumption is particularly costly, as it occupies capacity that could otherwise be used to host additional services or accommodate fluctuating demand [Shi et al., 2016].

The serverless computing model directly addresses this inefficiency: rather than reserving capacity in advance, resources are allocated exclusively upon invocation and released immediately upon completion, ensuring that computational capacity is consumed only when user requests actively trigger service execution [Baldini et al., 2017]. Beyond eliminating idle resource consumption, this execution model introduces a more granular and operationally transparent billing policy, in which costs are calculated at runtime in units of milliseconds rather than billed against statically reserved capacity [Li et al., 2022]. From the infrastructure provider's perspective, the invocation-based model reduces idle resource pools to near zero, en-

abling a fixed hardware budget to serve a substantially larger number of concurrent users compared to traditional reservation-based deployment models [Shafiei et al., 2022]. This makes the serverless paradigm inherently well-suited to the dynamic, bursty workload patterns characteristic of edge deployments. The serverless model is applicable across the full spectrum of application complexity: simple, stateless edge applications can be realized as individual functions, while advanced applications incorporating data analytics and ML/DL models can be realized as composite services, that is, chains of interconnected functions in which each stage of the analytical pipeline corresponds to a dedicated, independently invocable function unit [Baldini et al., 2017].

The exponential growth in data generation at the network periphery has intensified the need for intelligent service deployment models that abstract the complexity of infrastructure management from application developers and data scientists [Christidis et al., 2019]. In practice, the management of infrastructure resources for the execution of ML and AI algorithms, encompassing the specification of memory allocations, CPU configurations, physical topology, and runtime adaptation, demands full-stack development and deployment expertise that diverts the analyst's attention away from the core business logic and algorithmic concerns [Baldini et al., 2017]. The evolution of cloud computing toward progressively higher levels of abstraction, from virtual machines through microservices to the serverless model, reflects a continuous drive to eliminate this operational burden and enable developers to concentrate exclusively on application logic [Sewak and Singh, 2018].

The serverless computing model, realized through the FaaS paradigm, represents the current frontier of this abstraction trajectory and provides a natural architectural basis for the service-centric view of placement. Rather than provisioning long-running, pre-allocated service instances on designated nodes, FaaS platforms decompose application logic into stateless, fine-grained function units, referred to

in the literature as *nanoservices*, a term that denotes the finest achievable granularity of application decomposition, whereby each unit encapsulates a single, well-defined computational task constituting a part of a broader microservice [Decker et al., 2022]. These units are instantiated on demand upon invocation and terminated immediately upon completion [Kumari et al., 2026]. From the user and developer perspective, this model abstracts away infrastructure management entirely: resources are allocated by the platform as invocation events arrive, eliminating the need for costly pre-provisioned capacity and freeing the application developer to focus on the ML/DL algorithms and business logic [Baldini et al., 2017].

The FaaS model supports three distinct deployment granularities: the deployment of a service as a single function, as a set of independently deployed functions sharing a common application context, and as a sequential chain of functions in which the output of each stage serves as the input to the next [Ozdemir, 2024]. Among these, the sequential chain of functions emerges as the most appropriate model for realizing intelligent edge applications, as it directly reflects the inherent structure of ML/DL workflows whose constituent stages form a natural sequence of dependent computational tasks. It simultaneously satisfies the key requirements of modern edge intelligence: modular composition, independent invocability of each processing stage, stateless execution that scales without coordination overhead, and the ability to distribute pipeline stages across heterogeneous edge–cloud nodes in accordance with their respective resource profiles. It is this deployment model that elevates the serverless paradigm from a simple infrastructure abstraction to a principled architectural foundation for the next generation of AI-driven edge services.

The significance of service composition as an independent dimension of the placement problem extends beyond the serverless paradigm. In the cloud-based virtual machine context, the grouping of application components into VM clusters has been shown to directly govern application performance and resource provision-

ing cost, independently of which physical nodes host those VMs [Paraskevoulakou et al., 2024], a finding that mirrors and reinforces the function partitioning argument developed in this chapter for the serverless setting. Even when ML/DL services are realized as serverless function chains and the platform handles resource provisioning transparently, the question of how to optimally partition the service’s computational pipeline into function blocks remains open and consequential. Different groupings of the same pipeline tasks produce different inter-function data transfer volumes and therefore different end-to-end latency profiles, meaning that the placement of a serverless ML/DL service is not fully determined until both the hosting infrastructure and the internal function composition have been jointly optimized.

The apparent deployment simplicity afforded by the serverless model does not eliminate the service placement problem - it re-frames it. The optimal grouping of functions within a serverless pipeline therefore constitutes a specific and previously under-addressed aspect of the service placement problem, one approached from the service composition level rather than the infrastructure level, and it is precisely this aspect that the AI-driven approach presented in this chapter addresses. This chapter presents both contributions as a unified research trajectory, constituting a service-centric approach to dynamic placement that is principled, empirically validated, and deliberately scoped to the service composition level, demonstrating that the optimal deployment of an ML/DL service at the edge cannot be reduced to infrastructure allocation alone but requires explicit reasoning over the internal structure of the service itself.

The proof-of-concept presented in Section 5.2 therefore serves a dual purpose. It validates the technical feasibility of deploying complex ML/DL pipelines as serverless function chains on edge-compatible infrastructure, and simultaneously establishes the operational context that exposes a distinct and previously unaddressed aspect of the service placement problem: once a service is realized as a chain of

serverless functions, the question of how to optimally partition that chain into function blocks becomes a placement decision in its own right, one that operates at the service composition level rather than the infrastructure level, and whose resolution directly determines the end-to-end latency and QoE delivered to the end user. It is this aspect of the placement problem that the AI-based Function Regulator, presented in Section 5.3, is designed to address. The complementary perspective, approaching the same placement problem from the infrastructure resource level by collecting, modeling, and acting upon runtime telemetry from heterogeneous edge nodes, is developed in Chapter 6, where a closed-loop, three-tier AI-driven orchestration approach is presented as the resource-centric counterpart to the service-centric approach established here.

5.2 Proposed approach: ML-FaaS

The ML-FaaS approach addresses the challenge of deploying data-intensive ML/DL workloads as end-to-end serverless pipelines across edge–cloud environments, with particular emphasis on the demands introduced by heavy computational tasks spanning the full data processing and model inference lifecycle. To manage this complexity in a structured and operationally tractable manner, the approach partitions the ML/DL lifecycle across two distinct phases: an *offline phase*, responsible for all data preparation and model training activities, and an *online phase*, responsible for delivering inference results against new, unseen data supplied by the user through the serverless pipeline. This bisection is not arbitrary, it reflects the fundamental distinction between the computationally intensive, data-heavy procedures that can be performed in a controlled environment before deployment, and the latency-sensitive inference path that must operate under the performance constraints of the serverless platform at runtime.

The designed workflow follows the established ML/DL lifecycle, progressing through the following sequential steps: (i) data cleaning, (ii) feature transformation and

dimensionality reduction, (iii) label encoding for the categorical features, (iv) normalization and standardization (merely for the DL models), and (v) ML/DL model training and inference. The outcomes of each step are encapsulated within the corresponding serverless function, enabling them to be applied systematically to new unseen data during the online phase. Importantly, this workflow composition is intentionally flexible and extensible because the pipeline can be updated or augmented to accommodate the requirements of different ML/DL workflows, for instance, by introducing a dedicated data filtering function as an additional stage — without restructuring the overall architecture. The core components associated with feature elimination, normalization, and model training are realized as part of the offline phase, while ML/DL inference is exclusively realized as part of the online phase, ensuring a clean separation between preparation and execution responsibilities.

The proposed approach is realized on the Apache OpenWhisk event-driven serverless platform. This platform has been chosen for its multi-tenancy, which aligns with the core scope of the serverless paradigm, and for its comprehensive native support for event-driven, scalable function execution, its flexibility in terms of agile deployment and configuration, and critically, its ability to establish function chains without requiring additional cloud-provider-specific products such as external API gateways, managed container registries, or topology-dependent workflow configuration files and additional load balancer services in case of scalability raise. This distinction is consequential: commercial serverless solutions such as AWS Step Functions [Amazon Web Services, 2026] impose configuration overhead, including the setup of Amazon State Machines, API Gateway endpoints, and Elastic Container Registry resources, that extends well beyond the application development boundary and undermines the core abstraction principle of the serverless model. Apache OpenWhisk, by contrast, provides all necessary components for scalable, event-driven function execution within a single, self-contained platform, enabling the proposed approach to remain focused on the application's

business logic rather than on infrastructure configuration. It is also worth reporting that while cross-provider function workflows are technically feasible, composing pipelines across multiple cloud service providers introduces disproportionate costs, data transfer overhead, and performance variability that renders such configurations impractical for latency-sensitive ML/DL deployments. The proposed approach therefore, adopts a single-platform deployment model, realizing a generalized, custom-made chain of functions in which each stage encapsulates a specific analytical task aligned with the developer's business logic. Figure 5.1 depicts the realization of the ML/DL pipeline as a serverless function chain within the OpenWhisk ecosystem.

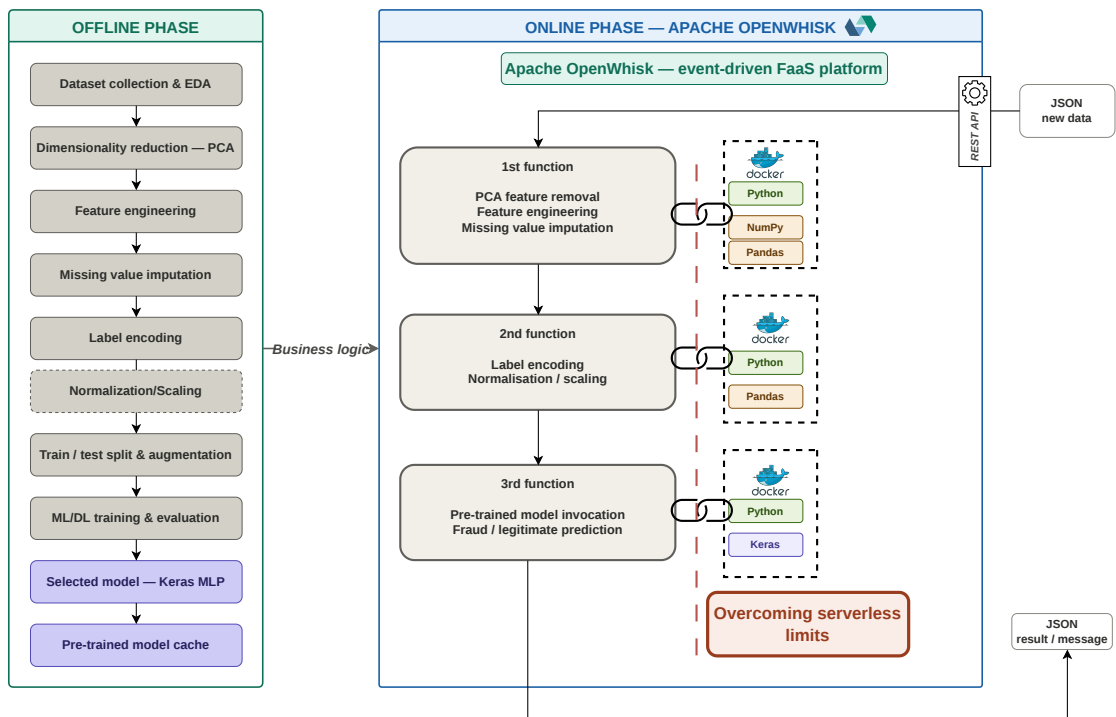


Figure 5.1: Overview of the proposed two-phase ML-FaaS architecture, illustrating the offline data analytics and model training phase, and the online serverless function pipeline realized on Apache OpenWhisk, with container-extended runtimes overcoming platform code-size constraints.

5.2.1 Offline phase

The offline phase centers on the data analytics and ML/DL model preparation activities that constitute the prerequisites for effective online inference. It is structured around the established ML/DL lifecycle, progressing through the following key stages: dataset collection and EDA, missing value imputation [Emmanuel et al., 2021], dimensionality reduction, feature engineering [Zheng and Casari, 2018], label encoding [Kosaraju et al., 2023], data separation and enrichment, model training, cross-validation, and performance evaluation and model selection. Figure 5.2 illustrates the complete data analytics exploitation path followed during the offline phase.

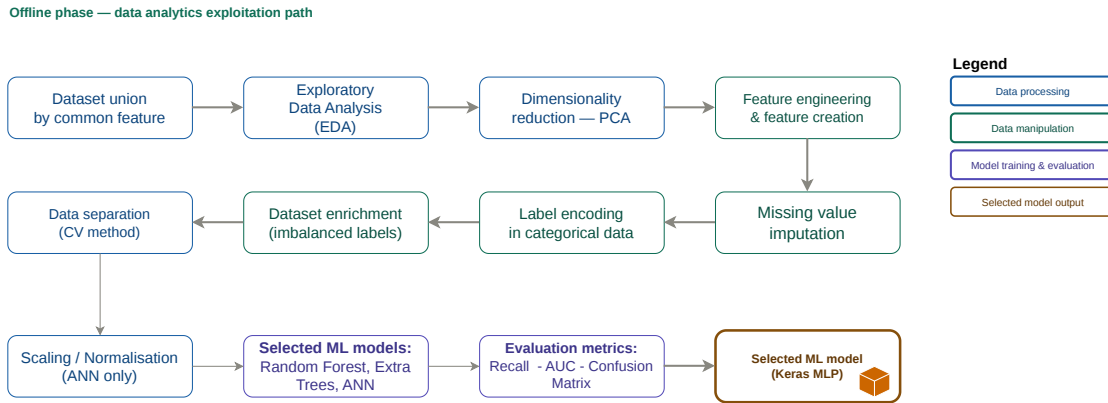


Figure 5.2: Proposed data analytics exploitation path for the offline phase of the ML-FaaS approach, depicting the sequential progression from dataset collection and EDA through dimensionality reduction, feature engineering, data enrichment, model training, and evaluation, culminating in the selection of the Keras MLP as the production inference model.

The offline phase is deliberately scoped to encompass all computationally intensive and data-heavy procedures that can be executed in a controlled environment prior to deployment, so that the online serverless pipeline is relieved of any training burden and can focus exclusively on low-latency inference against new, unseen data.

5.2.1.1 Dataset characterization, exploratory analysis, and preprocessing

The evaluation scenario employs a real-world fraud detection use case grounded in financial transaction data, a domain of direct operational relevance given the increasing prevalence of digital payment fraud across financial and e-commerce sectors. The dataset, sourced from a public benchmark provided by VESTA Corporation and the IEEE Computational Intelligence Society [Kaggle, 2019] comprises approximately 600,000 labeled records structured in comma-separated values format, encompassing 435 anonymized features of both numerical and categorical types representing legitimate and fraudulent e-commerce transactions. The selection of this dataset is motivated by two considerations: its real-world provenance, which ensures that the evaluation reflects genuine data distributions and quality challenges, and its direct relevance to the financial fraud detection domain, which constitutes one of the most consequential application areas of ML/DL classification in both the financial sector and law enforcement authorities.

A fundamental structural challenge of this dataset is severe class imbalance: only 3.5% of observations correspond to fraudulent transactions, as it is depicted in Figure 5.3, with the overwhelming majority representing legitimate activity. This distribution reflects the natural rarity of fraud events in real transactional data, but poses a significant risk to classifier development: standard training procedures would cause models to overfit the majority class, developing a systematic bias toward predicting legitimate transactions and consequently failing to detect the rare but operationally consequential fraud cases.

EDA is applied as the first analytical step, providing a structured overview of the dataset's distributional properties, revealing hidden patterns, and identifying the extent of missing values across feature columns. The missing-value imputation strategy is selected based on each feature's characteristics and its contribution to the overall data pattern, with the chosen strategy subsequently stored and applied consistently in the online phase to ensure alignment between training and infer-

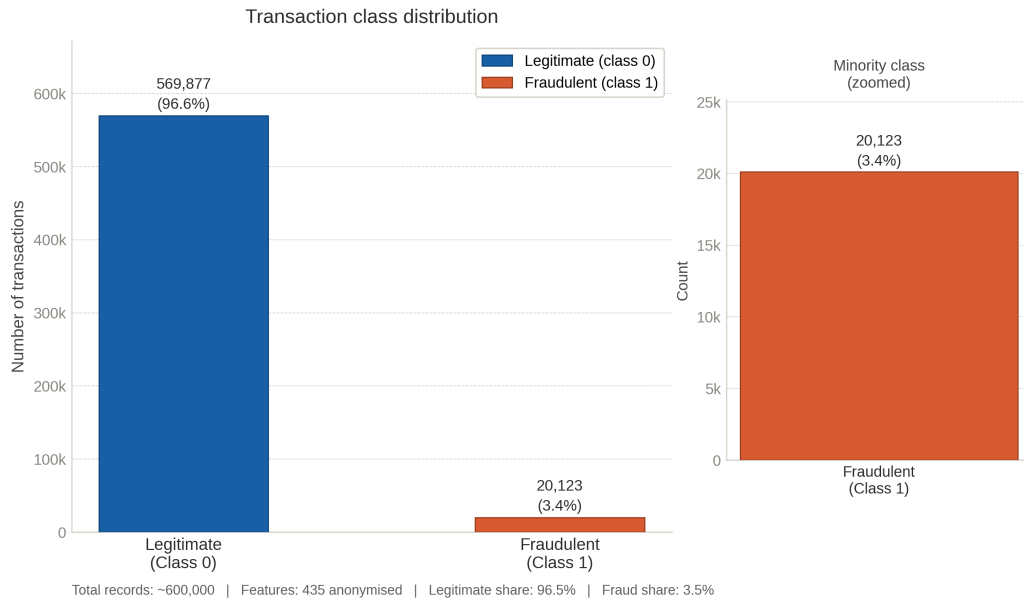


Figure 5.3: Class distribution of the fraud detection dataset, illustrating the severe imbalance between legitimate and fraudulent e-commerce transactions, with fraudulent observations comprising only 3.5% of the total dataset.

ence data transformations.

Figure 5.4 depicts the nullity matrix of the dataset, where grey shading indicates data presence and white indicates absence, revealing the extent and distributional pattern of missing values across a representative sample of features.

Feature dimensionality is subsequently reduced via Principal Component Analysis (PCA) [Abdi and Williams, 2010], which identifies 53 principal components retaining the most significant proportion of dataset variance. This dimensionality reduction step is essential for two reasons: the original 435-feature space contains correlated and redundant attributes that increase model complexity without contributing predictive value, and the reduced representation substantially decreases the computational burden of the training procedure.

Feature engineering further enriches the dataset by constructing meaningful derived attributes from the anonymized raw features. A particularly informative in-

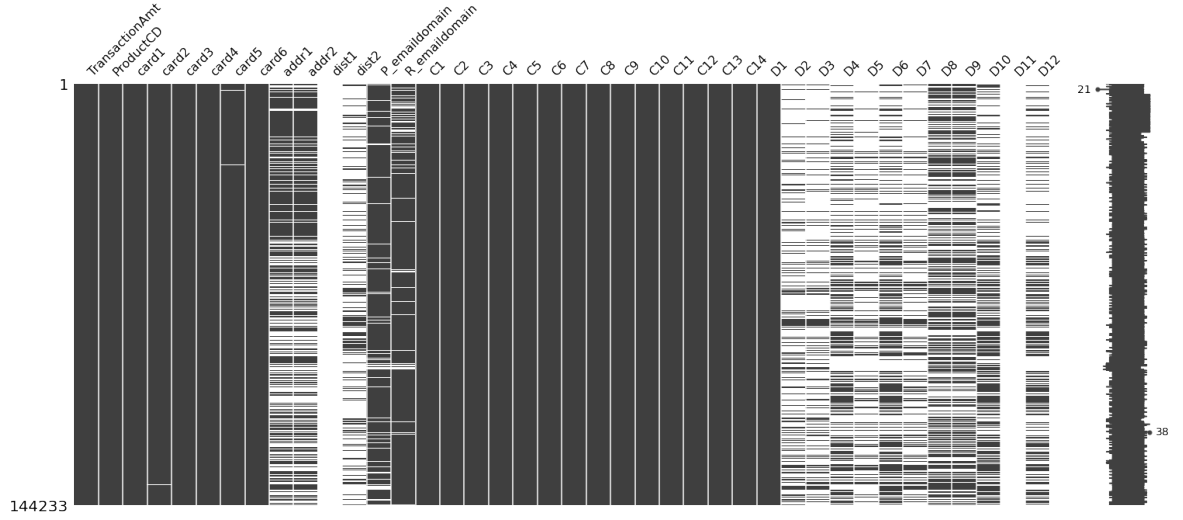


Figure 5.4: Nullity matrix visualizing the patterns of missing values across a representative sample of the dataset’s features. Grey shading indicates data presence, and white indicates absence. The high proportion of missing values across multiple feature columns necessitates systematic imputation before model training.

sight derived from this analysis concerns the temporal distribution of fraudulent transactions across weekdays: as illustrated in Figure 5.5, fraudulent activity is disproportionately concentrated on Saturdays, Sundays, and Fridays. This pattern is consistent with the reduced operational responsiveness of banking institutions and law enforcement agencies during weekend periods, which creates a temporal window that perpetrators systematically exploit. This finding directly motivates the inclusion of weekday-derived features in the feature engineering step, as the temporal dimension carries a meaningful discriminative signal for the fraud detection classifier.

As the next step in the data analysis pipeline, the One-Hot Encoding technique [Rodríguez et al., 2018] is applied to the dataset’s categorical variables to ensure numerical compatibility with the selected model architectures [Hancock and Khoshgoftaar, 2020].

A critical challenge inherent to real-world datasets of this nature is the class im-

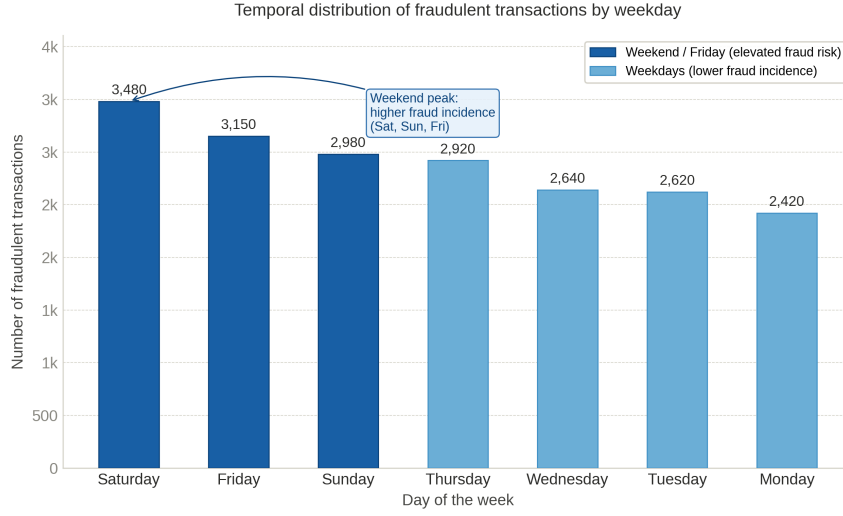


Figure 5.5: Distribution of fraudulent transactions across weekdays, revealing a statistically elevated concentration of fraud on Saturday, Sunday, and Friday. This temporal pattern is attributable to the reduced responsiveness of financial institutions and law enforcement during weekend periods, and motivates the extraction of weekday-derived features during the feature engineering step.

balance problem [Altalhan et al., 2025]. Anomalous events (such as fraudulent transactions) are naturally rare in practice, and the resulting class distribution is heavily skewed toward the majority class. This imbalance introduces a systematic bias into standard training procedures: classifiers trained on such data tend to exploit the distributional asymmetry by predominantly predicting the majority class, which yields deceptively high overall accuracy scores while fundamentally failing to detect the minority class instances that are precisely the ones of greatest operational interest. In the fraud detection context, this failure mode is particularly consequential: a model that correctly classifies 96.5% of transactions as legitimate while missing the majority of fraudulent ones provides no practical value, and may instil a false sense of security in operational deployments where the cost of a missed fraud event substantially exceeds the cost of a false alarm. Addressing class imbalance is therefore not merely a modelling consideration but a prerequisite for producing a classifier that is genuinely useful in practice. To this end,

three distinct data enrichment strategies are applied and comparatively evaluated across all model families:

- **Random oversampling** of minority class instances, which replicates existing fraud observations to increase their representation in the training set.
- **Weight assignation to the minority class**, which penalizes misclassification of fraud instances more heavily during training without modifying the dataset distribution.
- **SMOTE combined with Tomek Link [Chawla et al., 2002] undersampling**, which synthetically generates new minority class instances using the Synthetic Minority Oversampling Technique while simultaneously removing borderline majority class instances through Tomek Link cleaning, producing a more balanced and clearly separated class boundary.

The training set comprises 80% of the available records, with the remaining 20% reserved as a held-out evaluation set. To ensure robust generalization performance and mitigate the risk of overfitting to any particular data split, stratified k -fold cross-validation [Widodo et al., 2022] is applied to the training set, preserving the class distribution across all folds.

5.2.1.2 ML/DL model design and selection

The ML/DL models evaluated span three algorithmic families: Random Forest, Extremely Randomized Trees, and a MLP implemented using the Keras deep learning framework [Keras Contributors, 2019]. Random Forest and Extremely Randomized Trees are ensemble methods that construct multiple decision trees over random subsets of the training data and aggregate their predictions, providing strong baseline performance and robustness to noisy features. The MLP constitutes the primary model of interest, as it is the architecture subsequently deployed in the online serverless pipeline for inference, and its design is oriented toward exploita-

tion through the FaaS paradigm.

The MLP architecture is designed as a feedforward neural network comprising three hidden layers and a single output node that defines the classification result. The Rectified Linear Unit (ReLU) activation function is applied across all three hidden layers, enabling the model to capture the non-linear relationships present in the feature space without suffering from the vanishing gradient problem that affects earlier activation functions [Agarap, 2018]. A dropout regularization layer is incorporated into the architecture to prevent overfitting, randomly deactivating a proportion of neurons during each training iteration so that the model cannot become excessively dependent on any particular subset of features.

The output layer employs the Sigmoid activation function, which maps the network's output to a probability value in the range $[0, 1]$, making it directly appropriate for the binary classification formulation of the fraud detection problem [Karlik and Olgac, 2011]. The model is optimized using the Adam algorithm, which adaptively adjusts the learning rate throughout training and has demonstrated reliable convergence performance across a wide range of deep learning architectures. The binary cross-entropy function is selected as the training loss objective, consistent with the binary nature of the classification labels.

Prior to training, a data normalization procedure is applied to all input features. This step is essential for MLP-based models: without normalization, the gradient updates during backpropagation become disproportionately influenced by features with larger numerical ranges, rendering the training process slow, unstable, and unreliable [Karlik and Olgac, 2011]. Normalization ensures that all features contribute on an equal numerical footing, enabling the optimizer to converge efficiently. Regarding the training procedure, stratified k -fold cross-validation is applied to the training set [Raschka, 2018, scikit-learn Developers, 2019], preserving the class distribution across all folds and ensuring that the model's generalization performance is assessed robustly across multiple data partitions rather than on a

single held-out split.

Model selection is based on recall and AUC evaluation metrics, rather than raw classification accuracy, reflecting the asymmetric cost structure of the fraud detection problem: a false negative, classifying a fraudulent transaction as legitimate, carries a substantially higher operational cost than a false positive, making recall the primary performance criterion. Among all evaluated models and augmentation combinations, the MLP trained with minority class weight assignment achieves the highest classification performance, attaining a recall of 84% and an AUC of 85% on the held-out evaluation set. This configuration is selected as the production model, and its trained weights, preprocessing transformations, and normalization parameters are persisted and subsequently loaded within the third serverless function of the online pipeline for inference against new, unseen data.

5.2.2 Online phase

The central architectural challenge of realizing effective service provisioning through the serverless model lies in the decomposition and integration of the application's business logic into fine-grained, independently invocable function units known as *nanoservices*. Nanoservices constitute an architectural pattern characterized by properties of isolation and technological independence [Wolff, 2016], conceived primarily to address the operational complexities associated with microservice-based architectures. They represent an evolutionary refinement in application decomposition: by implementing even finer-grained functional components, maintenance efficiency is improved, and the boundary between serverless functions and nanoservices effectively converges, as both adopt the same fundamental properties of statelessness, isolation, and independent scalability [Morenets and Shabin-skiy, 2020, Ibanez et al., 2019, Kritikos and Skrzypek, 2018]. The serverless computing model operationalizes this principle by enabling developers to encapsulate discrete processes within individual functions, entirely relieving them of the re-

sponsibility for resource allocation, dynamic workload management, and infrastructure administration.

In the proposed approach, a pipeline of three interconnected functions is deployed on Apache OpenWhisk to execute the data analytics and ML/DL inference tasks defined during the offline phase. Each function is implemented in Python and executed within a compatible Python runtime environment. Data is exchanged between functions as JSON-structured parameter dictionaries, with each function's output serving as the input to the subsequent stage. The responsibilities of each function are as follows.

- **First function:** receives the input parameters in JSON format and restructures them to conform to the data format established during the offline phase. It subsequently traces and imputes any missing values according to the imputation strategy determined in the offline phase, applies the feature engineering transformations to construct the derived attributes required by the model, and encodes categorical variables to maintain a consistent data representation throughout the pipeline.
- **Second function:** receives the transformed output of the first function as its input. Its primary responsibility is to apply label encoding to categorical variables and to perform normalization according to the standardization procedure fitted during the offline phase, ensuring full alignment between the preprocessing applied during training and that applied at inference time. The normalized parameters are then forwarded as input to the third and final function.
- **Third function:** loads the pre-trained MLP model produced during the offline phase and integrates it as an inference engine. Upon execution, the model processes the normalized input parameters and produces a binary classification output, returning a structured JSON response that indicates whether the input transaction is classified as fraudulent or legitimate.

The end-to-end pipeline is activated through a REST API endpoint exposed within the OpenWhisk environment. Upon receiving a new data request, an HTTP invocation is directed through the platform's core components — the NGINX server [NGINX, Inc., 2024], controller, CouchDB [Apache Software Foundation, 2024a] state store, load balancer, and function container — which collectively manage request routing, authorization, and action scheduling before propagating execution through the function chain.

5.2.3 Overcoming serverless constraints via container-extended runtimes

A fundamental operational constraint of FaaS platforms is the strict limitation imposed on the size of a function's source code package. In the case of Apache OpenWhisk, this limit is set at 48 MB [IBM Cloud Documentation, 2022], a threshold that is routinely exceeded by ML/DL workloads due to their dependence on analytical libraries — including Pandas, NumPy, scikit-learn, Keras, and SciPy — whose compiled native dependencies collectively produce dependency trees amounting to several hundred megabytes.

To address this constraint, the approach extends the OpenWhisk runtime through custom Docker container mechanisms. Dedicated container images are constructed in which the required libraries are pre-installed during the build process and exposed as part of the runtime environment at invocation time. These custom runtime containers are injected into the OpenWhisk environment at function creation time using the `--docker` flag, and are linked to the constructed function without violating the platform's code-size restriction [Apache Software Foundation, 2021]. The custom runtime containers must expose an HTTP server on port 8080 with `/init` and `/run` endpoints through which the platform passes invocation requests. This mechanism preserves the stateless nature of the serverless execution model, as the container functionality is pre-installed at function creation time rather than

at invocation time.

Beyond addressing the code-size constraint, this hybrid deployment model increases the modularity of the overall pipeline, making each stage easier to develop, test, and update independently. Apache OpenWhisk is preferred over alternatives such as OpenFaaS [OpenFaaS Ltd., 2024] because it supports multi-tenancy, enabling it to serve multiple concurrent users within a shared edge infrastructure — a critical requirement for edge deployment scenarios where a single node must host multiple independent services simultaneously.

5.3 AI-based function regulator for approaching the service placement problem

5.3.1 Problem formulation

Although cloud providers offer template engines through their service marketplaces to assist developers in constructing serverless functions for specific programming languages, no established mechanism exists for determining the optimal number of functions required to compose a given service workflow. Prior research has explored adjacent dimensions of this problem: efforts have been directed toward learning the relationships between cost and runtime across different serverless function configurations to identify the configuration that minimizes cost while satisfying user-defined performance criteria, such as response time [Akhtar et al., 2020], and toward automating the memory size optimization of individual serverless functions through resource sizing techniques [Eismann et al., 2021]. However, neither line of work addresses the question of how many function blocks should compose a pipeline, nor how those functions should be grouped to minimize end-to-end response time.

The AI-based Function Regulator proposed in this chapter moves beyond stan-

standard resource adaptation mechanisms by directly addressing this open problem. The central hypothesis examined is whether a smaller number of composed pipeline functions necessarily produces a lower end-to-end response time — an assumption that, if valid, would simplify the composition problem to a trivial minimization. To evaluate this hypothesis, the ML-FaaS proof-of-concept introduced in Section 5.2 is instantiated as four distinct pipeline configurations comprising 2, 3, 4, and 5 chained functions, respectively. The experimentation covers four composition scenarios structured around the five established data preprocessing lifecycle steps — cleaning, imputation, transformation, label encoding, and normalization-standardization — with 1,000 successive invocation requests triggered against each configuration to collect comparative response time distributions.

Figure 5.6 presents the response time measurements derived (from the experiment) for each of the constructed pipelines.

The results lead to the rejection of the initial hypothesis: a smaller number of pipeline functions does not reliably produce a lower response time. In particular, the 3-function pipeline yields a shorter end-to-end response time than the 2-function pipeline, demonstrating that the relationship between pipeline cardinality and response time is non-monotonic. The 5-function pipeline is excluded from further consideration due to extreme divergence in response times. Additionally, an analysis of CPU utilization during function execution reveals a homogeneous usage pattern across all configurations, confirming that CPU consumption is not a discriminating attribute and should not be incorporated into the composition optimization framework.

5.3.2 Proposed architecture

Having established that pipeline cardinality does not determine response time, the AI-based Function Regulator is proposed as a holistic, data-driven approach



Figure 5.6: Response time distributions across the four candidate pipeline configurations (2, 3, 4, and 5 functions), illustrating the non-monotonic relationship between pipeline cardinality and end-to-end response time. The 5-function pipeline is excluded from further analysis due to response time divergence.

that automatically identifies the optimal number of serverless functions for a given ML/DL pipeline by predicting the function composition’s runtime execution behavior. The developer provides the implemented source code as input and receives as output a concrete recommendation for the number of functions that will deliver the minimum end-to-end response time, thereby ensuring the highest achievable QoE for the end user. Figure 5.7 depicts the conceptual architecture of the AI-based Function Regulator and Algorithm 1 presents its complete operational workflow. The internal components are described in detail below.

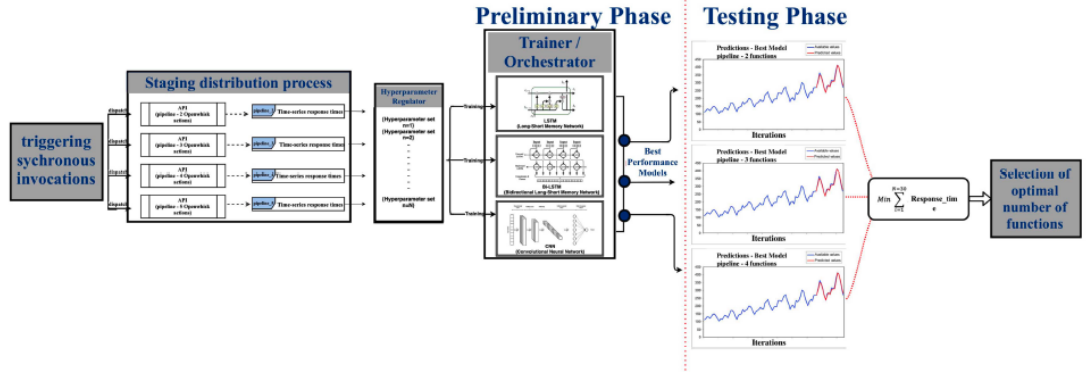


Figure 5.7: Conceptual architecture of the AI-based Function Regulator, comprising the Staging Distribution Process, the Hyperparameter Regulator, and the Trainer/Orchestrator components.

5.3.2.1 Staging distribution process

The Staging Distribution Process constitutes the initial component of the regulator. It accepts the pipeline source code as input and generates the set of candidate pipeline configurations by applying distinct function partitioning strategies to the available computational tasks. Based on the empirical findings of the hypothesis evaluation, three configurations are retained for the optimization process, pipelines of 2, 3, and 4 functions, with the 5-function configuration excluded due to its demonstrated response time instability.

For each candidate configuration, the component triggers 1,000 successive invocation requests and collects the resulting response times as time-series data. Since FaaS platforms are subject to the *cold-start phenomenon* [Vahidinia et al., 2020], where the first invocation following a period of container inactivity incurs a disproportionately elevated response time due to the initialization of runtime containers, the collected time-series contain spike points that do not reflect the steady-state performance of the pipeline. To address this, the Hampel filter is applied to each series to identify and replace outlier values through a sliding window mechanism operating over the sequential observations [Pearson, 2002]. This pre-processing

Algorithm 1 AI-based Function Regulator for optimal serverless pipeline composition

Require: Source code of ML/DL pipeline, candidate pipeline cardinalities $K \in \{2, 3, 4\}$, number of invocation requests $N = 1000$, Hampel filter window size w , DL architectures $\mathcal{M} = \{\text{LSTM, BLSTM, CNN}\}$

Ensure: Optimal number of functions k^* minimizing end-to-end response time

```

1: // === Stage 1: Staging Distribution Process ===
2: for each cardinality  $k \in K$  do
3:   Partition pipeline source code into  $k$  function blocks
4:   Deploy  $k$ -function pipeline on the serverless platform
5:   Trigger  $N$  successive invocation requests
6:   Collect response times as time-series  $T_k$ 
7:   Apply Hampel filter to  $T_k$ : remove cold-start spike points           ✦Outlier detection
8:   Store pre-processed series  $\tilde{T}_k$ 
9: end for

10: // === Stage 2: Hyperparameter Regulator ===
11: for each architecture  $m \in \mathcal{M}$  do
12:   for each cardinality  $k \in K$  do
13:     Generate all valid hyperparameter combinations for  $m$ 
14:     Train and evaluate each combination on  $\tilde{T}_k$ 
15:     Select best configuration  $\lambda_{m,k}^* = \arg \min \text{RMSE}(m, \tilde{T}_k)$ 
16:   end for
17: end for

18: // === Stage 3: Trainer/Orchestrator ===
19: for each architecture  $m \in \mathcal{M}$  do
20:   for each cardinality  $k \in K$  do
21:     Train model  $m$  on  $\tilde{T}_k$  using  $\lambda_{m,k}^*$ 
22:     Generate forecasted response time series  $\hat{T}_{m,k}$  over testing horizon
23:     Compute cumulative forecasted response time:  $S_{m,k} = \sum \hat{T}_{m,k}$ 
24:   end for
25: end for

26: // === Selection ===
27: for each cardinality  $k \in K$  do
28:    $S_k^* = \min_{m \in \mathcal{M}} S_{m,k}$                                            ✦Best model per cardinality
29: end for
30:  $k^* = \arg \min_{k \in K} S_k^*$                                            ✦Optimal pipeline cardinality
31: return  $k^*$ 

```

step ensures that the downstream forecasting models are trained on time-series that accurately reflect warm execution behavior, undistorted by initialization artifacts.

5.3.2.2 Hyperparameter regulator

The Hyperparameter Regulator is an automated optimization component designed to eliminate the manual trial-and-error process associated with neural network hyperparameter configuration. Rather than requiring the practitioner to manually explore the parameter space, the component systematically generates a comprehensive landscape of all valid combinations of neural network parameters and evaluates the effectiveness and robustness of each configuration across the candidate architectures. Performance is quantified in terms of RMSE on the pre-processed response time series, providing a standardized basis for comparing the predictive accuracy of different algorithmic approaches and parameter combinations. This automated process ensures that all subsequent forecasting models operate at their best achievable configuration, enabling a fair and rigorous comparative evaluation.

5.3.2.3 Trainer/Orchestrator

The Trainer/Orchestrator is the principal component responsible for the training of the time-series forecasting models and the derivation of the final pipeline composition recommendation. The pre-processed response time series for each candidate pipeline configuration are supplied as inputs to three DL architectures, evaluated comparatively:

- **LSTM Neural Network** [Lindemann et al., 2021]: a recurrent neural network architecture specifically designed to model temporal sequences, capable of capturing long-range dependencies in time-series data through its gated memory cell mechanism, and widely recognized for its effectiveness in time-series forecasting tasks.
- **Bi-directional LSTM** [Schuster and Paliwal, 1997]: an extension of the standard LSTM architecture that processes the input sequence simultaneously in both forward and backward temporal directions, enabling the model to

exploit both past and future context when forming predictions and thereby improving forecasting accuracy for sequences with bidirectional temporal dependencies.

- **Convolutional Neural Network (CNN)** [Sadouk, 2019]: originally developed for image recognition and computer vision tasks, CNN architectures have demonstrated strong performance in sequence modeling problems through their ability to extract local temporal patterns via convolutional filters, making them a competitive alternative to recurrent architectures for time-series forecasting.

For each architecture and pipeline configuration, the model with the lowest RMSE (as identified by the Hyperparameter Regulator) is selected and used to generate a fixed number of forecasted response-time values during the testing phase. Upon completion of the forecasting phase, the Trainer/Orchestrator aggregates the forecasted values for each pipeline configuration by computing their cumulative sum, which represents the expected total response time over the testing horizon. The pipeline configuration that produces the minimum cumulative forecasted response time is selected as the optimal composition template for the application's realization.

5.4 Evaluation

5.4.1 Experimental setup

The evaluation is structured across three complementary dimensions: (a) the quality of the ML/DL classification model produced during the offline phase regarding the serverless pipeline proof-of-concept, (b) the performance of the online serverless pipeline in terms of per-action and end-to-end response time, and (c) the effectiveness of the AI-based Function Regulator in identifying the optimal

pipeline composition. In the offline phase of the constructed data analysis serverless pipeline, the evaluation includes the testing of three ML/DL model families trained under four distinct data manipulation strategies. The online phase evaluation measures per-action and end-to-end response times across 10 re-execution iterations, and as it concerns the AI-based Function Regulator results, tests two hardware configurations: a 4-core and an 8-core Ubuntu 16.04 host at low and high memory allocation levels.

5.4.2 Offline phase evaluation results

Table 5.1 presents the evaluation results of the offline phase across all model families and data augmentation strategies (Weight to the minority class samples, over-sampling to the minority class samples, SMOTE and Tomek Link technique). Three ML/DL model families are evaluated: **Random Forest**, an ensemble method that constructs multiple decision trees over random feature subsets and aggregates their predictions through majority voting, offering strong robustness to noisy and high-dimensional features [Breiman, 2001], **Extremely Randomized Trees** (Extra Trees), a variant of the Random Forest that introduces additional randomness in the split selection process, reducing variance at the cost of a slight increase in bias and generally achieving faster training convergence [Geurts et al., 2006], and a **Multilayer Perceptron (MLP)** [Popescu et al., 2009] implemented using the Keras framework [Keras Contributors, 2019], a feedforward deep neural network architecture selected as the primary model of interest given its capacity to learn complex non-linear decision boundaries and its suitability for deployment as a serverless inference function within the online phase of the ML-FaaS pipeline.

The results confirm that the choice of augmentation strategy has a substantial impact on model performance, particularly for the Artificial Neural Network (ANN)-based architecture. Among all evaluated configurations, the MLP trained with minority class weight assignation achieves the highest classification performance,

with a recall of 84% and an AUC of 85%. This configuration is selected as the production model for deployment in the online serverless pipeline. The ensemble methods, while competitive in terms of AUC, with Extremely Randomized Trees achieving 84.18% AUC under the Synthetic Minority Oversampling Technique (SMOTE) and Tomek Link strategy, consistently underperform the MLP in terms of recall, which constitutes the primary evaluation criterion given the asymmetric cost of false negatives in the fraud detection context. Several configurations across both ensemble families exhibit overfitting when trained with oversampling techniques, as evidenced by elevated training-phase performance that fails to generalize to the held-out evaluation set, confirming that the weight assignment approach offers a more stable generalization profile for this dataset and problem formulation. Figure 5.8 provides a comprehensive visual overview of the classification performance across all model families and augmentation strategies. Sub-figures 5.8(a) and 5.8(b) present the grouped bar charts of recall and AUC respectively across all three model families — Random Forest, Extremely Randomized Trees, and Keras MLP — under each of the four augmentation strategies, enabling a direct cross-family and cross-strategy comparison. Sub-figures 5.8(c) and 5.8(d) isolate the Keras MLP results, providing a detailed view of the recall and AUC performance per augmentation strategy and confirming the superiority of the weight assignment approach over the remaining three configurations. The best-performing configuration Keras MLP with minority class weight assignment, achieving 84% recall and 85% AUC, is highlighted across both detail panels. Sub-figures 5.8(e) and 5.8(f) present factor plots that trace the trajectory of each augmentation strategy across model families, revealing that the weight assignment strategy produces the most pronounced performance improvement specifically for the ANN-based architecture, while remaining comparatively stable across the ensemble methods, a behavioral pattern that directly motivates its selection as the production augmentation strategy for the deployed inference model.

Table 5.1: Evaluation results of the offline phase across ML/DL model families and data augmentation strategies. The best-performing configuration is highlighted in bold.

Model	Recall	AUC
Random Forest (raw data)	56.00%	78.00%
Random Forest — applying weight to minority class samples	55.26%	77.39%
Random Forest — with oversampling 40% minority class	63.46%	81.37%
Random Forest — applying SMOTE & Tomek Link technique	57.32%	78.56%
Extra Trees Classifier (raw data)	61.07%	80.42%
Extra Trees Classifier — applying weight to minority class samples	61.97%	80.23%
Extra Trees Classifier — with oversampling 40% minority class	62.47%	80.89%
Extra Trees Classifier — applying SMOTE & Tomek Link technique	69.32%	84.18%
Keras ANN (raw data)	54.25%	76.9%
Keras ANN — applying weight to minority class samples	84%	85%
Keras ANN — with oversampling 40% minority class	72.1%	84.84%
Keras ANN — applying SMOTE & Tomek Link technique	67.99%	82.97%

5.4.3 Online phase and pipeline performance

Figure 5.9 presents the per-action and end-to-end response time measurements collected across 10 re-executions of the 3-function pipeline under diverse execution conditions (both cold and warm action invocations). Sub-figure (a) depicts the execution time of the first function, responsible for data transformation and imputation, whose warm execution times range from 53 milliseconds to approximately 350 milliseconds, with a cold-start peak of 1.40 seconds on the first invocation. Sub-figure (b) shows the second function, responsible for normalization, which achieves the lowest warm execution times of the three stages, ranging from 20 to 180 milliseconds, with a cold-start of 3.60 seconds. Sub-figure (c) presents the third function, which invokes the pre-trained MLP for inference, exhibiting warm execution times between 180 and 600 milliseconds and a cold-start peak of 4.30 seconds. Sub-figure (d) shows the end-to-end pipeline response time, which under warm conditions stabilises within an acceptable latency range of a few hun-

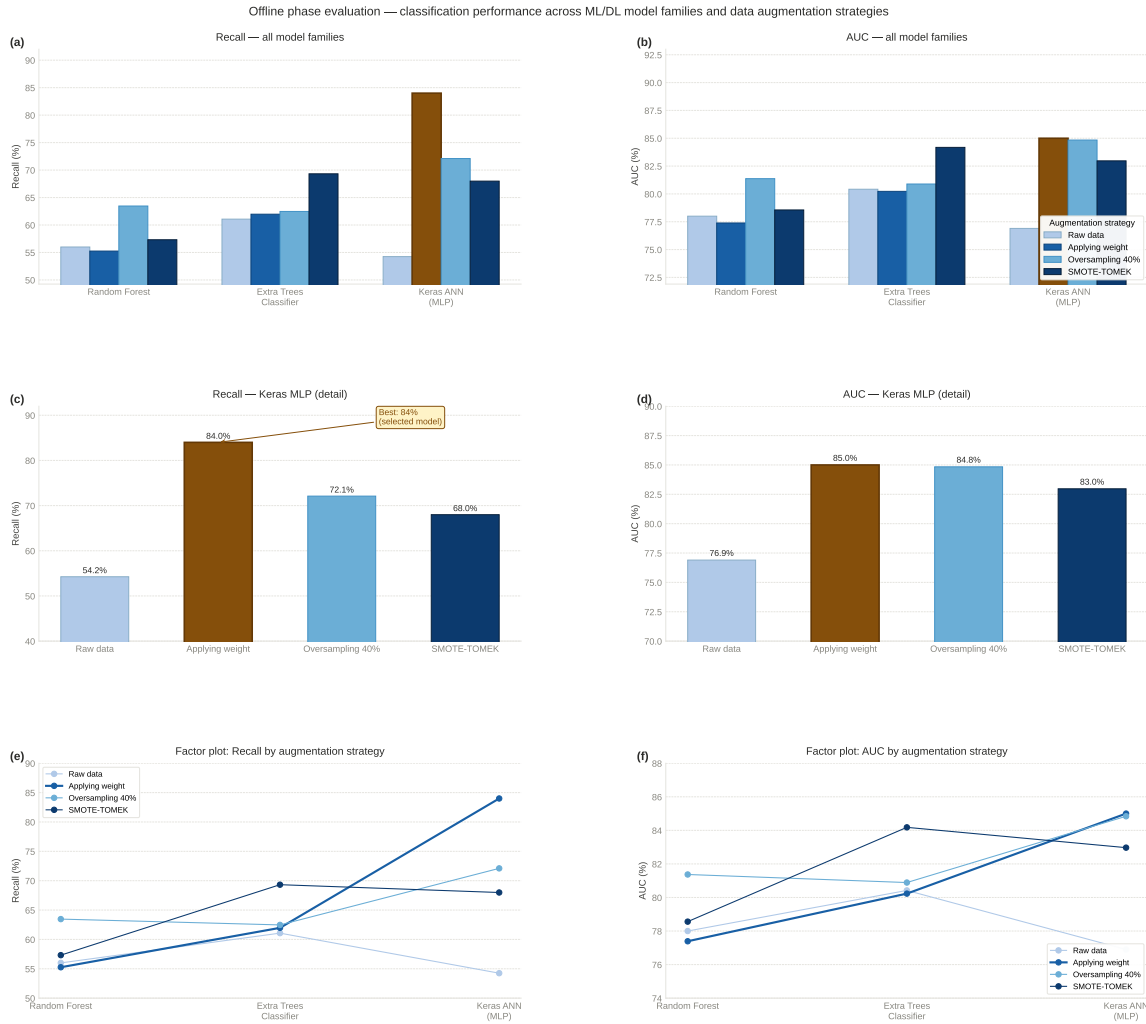


Figure 5.8: Classification performance of the evaluated ML/DL models across data augmentation strategies, measured in terms of recall (sensitivity) and AUC.

dred milliseconds to approximately 3 seconds, while the first invocation incurs a cold-start penalty of approximately 25 seconds due to the sequential initialization of all three runtime containers. The dashed horizontal line in each panel indicates the average warm execution time across iterations 2–10, confirming rapid convergence to stable steady-state performance following the cold-start phase.

A particularly significant finding emerging from the resource allocation experiment is that response time is effectively independent of the allocated computa-

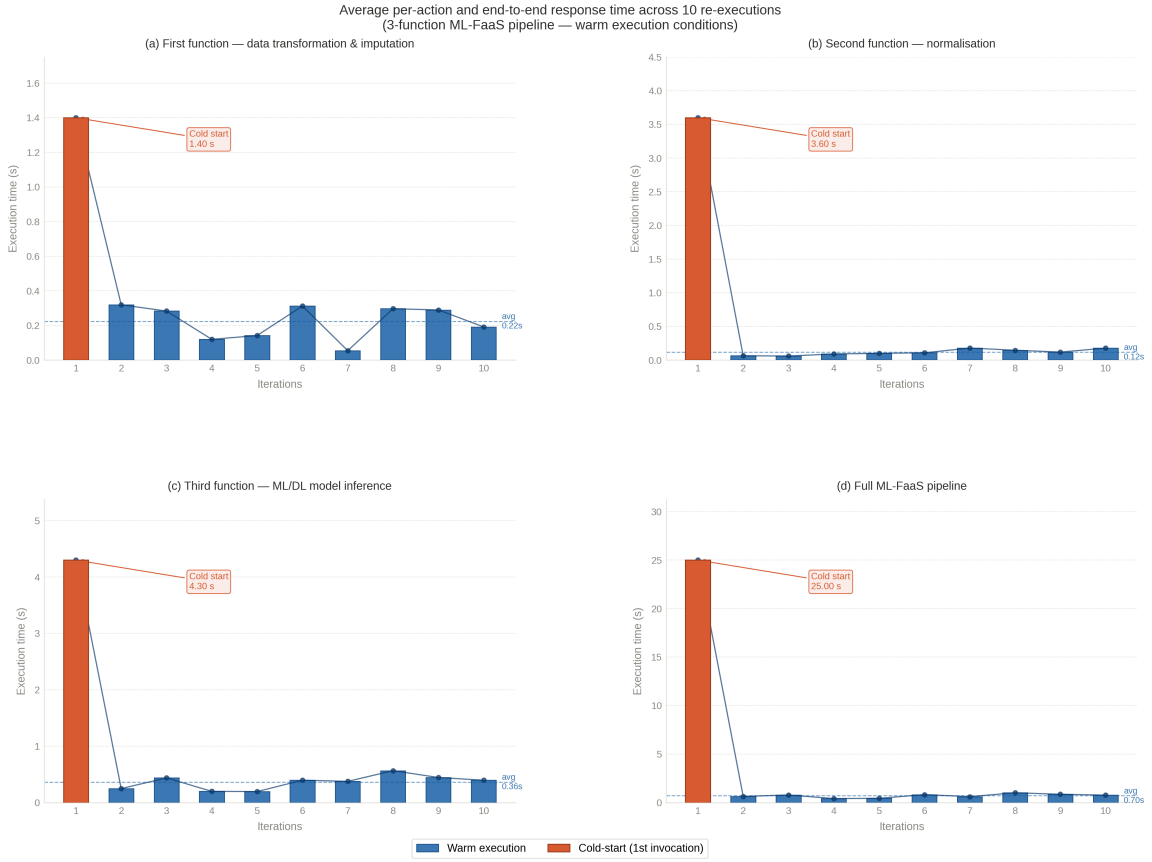


Figure 5.9: Average per-action and end-to-end pipeline response time across 10 re-executions of the 3-function ML-FaaS pipeline under warm execution conditions. Sub-figures show: (a) first function — data transformation and imputation, (b) second function — normalisation, (c) third function — ML/DL model inference, (d) full end-to-end pipeline. The cold-start invocation (iteration 1, orange bar) is annotated separately in each panel. The dashed line indicates the average warm execution time across iterations 2–10.

tional resources.

Table 5.2 demonstrates this for two hardware configurations, confirming that the dominant determinant of response time is the structural composition of the pipeline rather than the computational capacity of the hosting node. This result empirically validates the central claim of this chapter that service-level composition decisions are a necessary and independent dimension of the placement problem,

not reducible to infrastructure-level resource allocation.

Table 5.2: Average response time of the ML-FaaS pipeline tested under two hardware configurations, demonstrating the independence of response time from allocated computational resources.

Performance results	On average response time			
	Ubuntu 16.04 — 4 processor cores — 4 RAM		Ubuntu 16.04 — 8 processor cores — 8 RAM	
	Low-level	High-level	Low-level	High-level
1st $f(x)$	53 ms	1.40 sec	51 ms	1.41 sec
2nd $f(x)$	20 ms	3.60 sec	22 ms	3.51 sec
3rd $f(x)$	180 ms	4.30 sec	180 ms	4.11 sec
$f(x)$ s pipeline	3.8 sec	25.4 sec	3.4 sec	25.6 sec

5.4.4 Function partitioning and regulator results

This subsection presents the evaluation of the AI-based Function Regulator from two distinct perspectives: the optimal function chain grouping results and the comparative performance of the candidate DL models employed within the Trainer-/Orchestrator component.

5.4.4.1 Function partitioning results and regulator grouping recommendations

Table 5.3 presents the benchmarking results across the different task groupings for each pipeline cardinality. As the results demonstrate, different groupings of the same computational tasks into function blocks yield substantially different end-to-end response times, owing to the inter-function data transfer overhead introduced at each function boundary. The optimized grouping, identified automatically by the AI-based Function Regulator and highlighted in the table, minimizes this data transfer latency and thereby achieves the most efficient utilization of the function chain for the given ML/DL workflow. It is important to emphasize that the identification of this optimal partitioning is performed in a fully automated manner, requiring no manual experimentation from the developer.

Table 5.3: Benchmarking of different task groupings during the function partitioning procedure. Average response time is reported over 30 executions per configuration.

Combinations	Task [1] Cleaning imputation	Task [2] Data transform.	Task [3] Label encoding	Task [4] Data normalisation	Task [5] ML/DL prediction	Avg. time (30 exec.)
2 f.: [[1,2,3,4],[5]]	1st f(x)	1st f(x)	1st f(x)	1st f(x)	2nd f(x)	1.42 sec
2 f.: [[1,2,3],[4,5]]	1st f(x)	1st f(x)	1st f(x)	2nd f(x)	2nd f(x)	1.44 sec
2 f.: [[1],[2,3,4,5]]	1st f(x)	2nd f(x)	2nd f(x)	2nd f(x)	2nd f(x)	1.70 sec
3 f.: [[1,2,3],[4],[5]]	1st f(x)	1st f(x)	1st f(x)	2nd f(x)	3rd f(x)	0.87 sec
3 f.: [[1],[2,3,4],[5]]	1st f(x)	2nd f(x)	2nd f(x)	2nd f(x)	3rd f(x)	0.89 sec
3 f.: [[1],[2],[3,4,5]]	1st f(x)	2nd f(x)	3rd f(x)	3rd f(x)	3rd f(x)	0.93 sec
3 f.: [[1,2],[3,4],[5]]	1st f(x)	2nd f(x)	3rd f(x)	3rd f(x)	3rd f(x)	0.92 sec
>4 f.: [[1,2],[3],[4],[5]]	1st f(x)	1st f(x)	2nd f(x)	3rd f(x)	4th f(x)	0.96 sec
4 f.: [[1],[2,3],[4,5]]	1st f(x)	2nd f(x)	3rd f(x)	4th f(x)	4th f(x)	1.1 sec
4 f.: [[1],[2,3],[4],[5]]	1st f(x)	2nd f(x)	2nd f(x)	3rd f(x)	4th f(x)	0.98 sec
5 f.: [[1],[2],[3],[4],[5]]	1st f(x)	2nd f(x)	3rd f(x)	4th f(x)	5th f(x)	18.45 sec

5.4.4.2 Comparative performance of candidate DL models

Table 5.4 presents the forecasting performance of the three evaluated DL architectures in terms of RMSE across all pipeline configurations. The LSTM achieves the lowest RMSE values across the majority of configurations, establishing it as the most effective architecture for forecasting the response time series produced by this workload. The CNN and Bi-directional LSTM deliver competitive performance, with their optimal hyperparameter combinations identified automatically by the Hyperparameter Regulator for each configuration.

5.4.5 Development effort estimation via the constructive cost model

This subsection evaluates the business value of the proposed ML-FaaS pipeline from a software engineering perspective, by applying two well-established models for estimating development effort in computer programming: Constructive Cost Model (COCOMO) [Boehm, 1984] and its optimized model COCOMO 2.0 [Boehm

Table 5.4: Forecasting performance of the evaluated DL architectures (LSTM, Bi-directional LSTM, CNN) in terms of RMSE across the three candidate pipeline configurations, together with the optimal hyperparameter combinations identified by the Hyperparameter Regulator.

Model	RMSE train/test	Pipeline-2	RMSE train/test	Pipeline-3	RMSE train/test	Pipeline-4
		Hyperparameter		Hyperparameter		Hyperparameter
LSTM	180.71 183.19	units=8, kernel_size=2, epochs=200, lag=8	79.01 80.42	units=6, kernel_size=2, epochs=90, lag=7	60.93 75.98	filters=6, kernel_size=3, epochs=200, lag=8
Bi-LSTM	192.94 199.17	units=7, kernel_size=2, epochs=90, lag=8	78.55 87.15	units=7, kernel_size=3, epochs=90, lag=7	70.85 77.44	units=4, kernel_size=2, epochs=200, lag=8
CNN	452.24 502.33	filters=64, kernel_size=2, dense=50, pool=2, lag=7	290.11 330.72	filters=64, kernel_size=2, epochs=200, dense=50, pool=2, lag=8	320.56 410.88	filters=64, kernel_size=2, epochs=90, dense=50, pool=2, lag=6

et al., 1995].

COCOMO is a regression-based model that estimates the effort and schedule of a software product, expressed in person-months (PM), using the source lines of code (SLOC) of the project. Beyond code size, the model incorporates a set of adjustment factors known as *cost drivers*, which account for aspects such as developer capability and experience, as well as product complexity. The core formula is defined as:

$$E = a \times (KLOC)^b \quad (5.1)$$

where a denotes the project development mode constant, $KLOC$ is the code size measured in thousands of SLOC, and b is a scaling factor capturing economies of scale. Projects are further categorised into three types: (i) Organic, (ii) Semi-detached, and (iii) Embedded. Given the scope of our implementation, the *Organic* development mode is selected, appropriate for small-scale projects, yielding the constants $a = 3.2$ and $b = 1.05$.

The implemented ML-FaaS application comprises 352 SLOC, covering the full pipeline: data acquisition, preprocessing, EDA, training across multiple ML/DL algorithms,

hyperparameter tuning, and function-level inference logic. Applying Equation 5.1, the nominal effort is computed as:

$$E_{nominal} = 3.2 \times (0.352)^{1.05} \approx 1.02 \text{ PM} \quad (5.2)$$

This baseline estimate is then adjusted using effort multipliers corresponding to the cost drivers defined in the COCOMO specification. COCOMO defines 15 such drivers, while COCOMO 2.0 extends this to 17, improving estimation accuracy. In this evaluation, nominal multiplier values (1.0) are applied to all cost drivers, with the exception of Product Complexity (CPLX), which reflects the operational burden associated with serving the final application.

For the ML-FaaS deployment, the serverless paradigm abstracts infrastructure concerns such as scaling, monitoring, and maintenance from the developer. The primary operational task reduces to the installation of Apache OpenWhisk and the deployment of the corresponding functions — a well-documented and straightforward procedure [Apache Software Foundation, 2024b]. Accordingly, a CPLX multiplier of 0.70 (very low complexity) is assigned, yielding a corrected effort estimate of **0.71 PM** for building, deploying, and serving the serverless ML application.

By contrast, a conventional cloud infrastructure deployment introduces substantially greater operational complexity. This encompasses the provisioning of virtual instances, configuration of load balancers for horizontal scaling, deployment of API gateways for service exposure, and setup of a message queuing system for handling concurrent inference requests. These requirements impose a significant burden on the developer, warranting a CPLX multiplier of 1.30 (very high complexity), which results in an estimated effort of **1.33 PM** — approximately double that of the serverless approach.

These findings quantitatively substantiate the practical advantage of the ML-FaaS paradigm not only in terms of technical performance, but also with respect to the

development and operational effort required to bring an ML/DL application to production.

5.5 Summary

This chapter presented the first of two complementary AI-driven approaches proposed in this thesis for addressing the dynamic service placement problem in heterogeneous edge–cloud environments. The research trajectory spans two publications: the preliminary work (*Leveraging the serverless paradigm for realizing machine learning pipelines across the edge-cloud continuum*), established the feasibility of realizing complete ML/DL analytics workflows as serverless function chains across the edge–cloud continuum, characterized the performance properties and operational challenges of the 3-function pipeline configuration, and identified the cold-start phenomenon and code-size constraints as the principal barriers to practical deployment. The extended work (*ML-FaaS: Toward Exploiting the Serverless Paradigm to Facilitate Machine Learning Functions as a Service*), introduced the AI-based Function Regulator, which addressed the previously open problem of optimal pipeline partitioning by demonstrating that response time is a non-monotonic function of pipeline cardinality and composition, and by providing a fully automated AI-driven mechanism for identifying the optimal configuration.

The evaluation results collectively established three findings of broader significance for the service placement problem. First, the non-monotonic relationship between pipeline cardinality and response time confirms that optimal composition is a non-trivial empirical property accessible only through systematic data-driven analysis. Second, the independence of response time from computational resource allocation, demonstrated empirically across two hardware configurations, provides empirical grounding for the service-centric perspective advanced throughout this chapter: infrastructure-level placement decisions alone are insufficient to guarantee QoE for serverless ML/DL services. Third, the effectiveness of the

AI-based Function Regulator in identifying the optimal composition in a fully automated manner validates the practical feasibility of this approach for real-world deployment scenarios.

These findings directly address **RQ1** and **RQ2** as defined in Chapter 1, establishing that AI-driven orchestration mechanisms can effectively address the dynamic service placement problem, and that a service-centric approach exploiting the serverless FaaS paradigm constitutes a resource-efficient and QoE-aware solution for dynamic service placement at the edge. The resource-centric perspective, which approaches the same placement problem from the infrastructure level through a three-tier AI-driven orchestration architecture, is developed in Chapter 6.

Chapter 6

A Closed-loop, three-tier AI-driven orchestration approach for dynamic resource-centric service placement

The research presented in this chapter has been published in: E. Paraskevoulakou, C. Symvoulidis, P. Karamolegkos, G. Kousiouris, and D. Kyriazis, “A Three-tier AI-based approach for dynamic application placement in cloud-edge environments,” Future Generation Computer Systems, 2026 [Paraskevoulakou et al., 2026].

Chapter structure

This Chapter is constructed as follows:

- **Section 6.1 – Motivation and resource-centric perspective**, establishes the rationale for approaching the dynamic application placement problem from the infrastructure resource level, positioning the proposed framework as the complementary resource-centric counterpart to the service-centric approach presented in Chapter 5.

- **Section 6.2 – System model and problem formulation**, defines the formal system model, including the edge node and application resource representations, the constraint formulation, and the multi-objective optimization problem governing placement decisions.
- **Section 6.3 – Core functionality of the three-layer AI-based approach**, presents the conceptual architecture of the proposed orchestrator and describes the data collection pipeline, the training phase, and the three AI layers: Incremental Learning (IL), Federated Learning (FL), and Reinforcement Learning (RL).
- **Section 6.4 – Evaluation**, describes the experimental setup, the deployment environment, and the evaluation results across all three layers, including comparative performance benchmarking against supervised ML baselines and classical heuristic placement strategies.
- **Section 6.5 – Summary**, synthesizes the contributions of this chapter and establishes the transition to the concluding discussion in Chapter 7.

6.1 Motivation and resource-centric perspective

Chapter 5 presented that the dynamic service placement problem possesses a service composition dimension that infrastructure-level reasoning alone cannot resolve: the internal grouping of a service’s computational tasks into serverless function blocks directly governs end-to-end latency, independently of which physical node hosts the service. The approach described in that chapter addressed this service-centric dimension by providing an AI-driven mechanism for identifying the optimal serverless function partitioning. The present chapter approaches the same placement problem from the complementary perspective: given that services have been appropriately composed, the question of *where* to deploy them across a heterogeneous edge–cloud infrastructure becomes the primary concern,

and it is this question that the resource-centric approach developed here is designed to answer.

The architectural and operational context within which this approach operates, encompassing the edge–cloud continuum, the challenges of dynamic application placement, and the limitations of classical and single-stage AI-driven approaches, has been established in Chapters 2, 3, and 4 respectively. The present chapter builds directly on that foundation to address the resource-centric dimension of the placement problem through a closed-loop, three-layer AI-driven orchestration architecture. In practice, edge environments exhibit dynamically changing workload patterns, system heterogeneity in terms of node capabilities, and privacy and bandwidth constraints that limit centralized training on raw telemetry data. These conditions create a gap in closed-loop orchestration designs that can simultaneously forecast near-future resource dynamics locally at each edge node, improve prediction robustness collaboratively without raw-data exchange, and exploit global observability to make coordinated placement decisions under user mobility.

A preliminary exploration of this resource-centric placement perspective was conducted in [Paraskevoulakou et al., 2024], which formulated the cloud-based application component placement problem as a Markov Decision Process and demonstrated that RL-based approaches, specifically DQN and PPO, can identify optimal component-to-VM groupings that balance application performance against resource provisioning cost under dynamic workload conditions. The results established that policy-gradient methods such as PPO substantially outperform value-based methods such as DQN as application complexity grows, and that the optimal placement can be identified in fewer than ten episode steps on average, a finding that directly informed the design of the three-layer RL-based orchestrator presented in this chapter.

To address this gap, this chapter presents a novel closed-loop, three-layer AI-based

orchestration approach for dynamic application placement in edge–cloud environments. The design of this approach is guided by three driving questions:

- Whether lightweight IL forecasting models deployed at each edge node can capture non-stationary resource dynamics sufficiently well to enable proactive resource re-adaptation.
- Whether FL can improve prediction stability and accuracy without exchanging raw telemetry data when local forecasting performance degrades due to concept drift [Lu et al., 2019].
- Whether a centralized RL model with global observability can generate accurate placement decisions that better balance user proximity and resource availability compared to heuristic and supervised baselines.

The proposed approach to these three questions constitutes its principal contribution: a unified, end-to-end orchestration design that integrates forecasting, privacy-preserving collaborative refinement, and sequential decision-making for dynamic placement in heterogeneous edge–cloud environments.

6.2 System model and problem formulation

In this chapter, the dynamic service placement is approached from the infrastructure perspective, considering the time-varying system conditions, within a cloud–edge architecture following the Master–Worker paradigm [Bouziane et al., 2010]. A large number of geographically distributed edge nodes are responsible for hosting and executing applications in proximity to users with diverse and evolving demands.

Mathematically formulated:

Let $\mathcal{N} = \{n_1, n_2, \dots, n_N\}$ denote the set of edge nodes, each associated with static geographical coordinates $O(x_n, y_n)$, and let $\mathcal{A} = \{a_1, a_2, \dots, a_A\}$ denote the set of ap-

plications to be placed. Each edge node $n_i \in \mathcal{N}$ is characterized by time-dependent resource capacities: $CPU_i^t \in \mathbb{R}^+$ representing available CPU at time t , and $Mem_i^t \in \mathbb{R}^+$ representing available memory at time t . Each application $a_j \in \mathcal{A}$ has corresponding time-varying resource requirements: cpu_j^t and mem_j^t denoting the CPU and memory demanded at time t respectively. Applications are assumed to be general-purpose, encompassing CPU-intensive, memory-intensive, and network-intensive workloads.

The feasibility constraint for deploying application a_j on node n_i at time t is formulated as:

$$cpu_j^t \leq CPU_i^t, \quad mem_j^t \leq Mem_i^t, \quad \forall a_j \in \mathcal{A}, n_i \in \mathcal{N}, t \in \mathcal{T}. \quad (6.1)$$

Each edge node n_i is additionally characterized by a time-dependent resource state vector capturing its current operational status:

$$r_i^t = [CPU_i^t, Mem_i^t, NetRx_i^t, NetTx_i^t], \quad (6.2)$$

where $NetRx_i^t$ and $NetTx_i^t$ denote the incoming and outgoing network traffic rates respectively. The user population consists of mobile entities with time-varying geographical coordinates: $u_i^t = (x_i^t, y_i^t)$, whose inter-timestep displacement is:

$$d = \sqrt{(x_i^{t+1} - x_i^t)^2 + (y_i^{t+1} - y_i^t)^2} \neq 0. \quad (6.3)$$

The latency experienced by a user is a function of the physical proximity between the user's current geolocation and the node hosting the requested application: when the spatial distance between the user and the hosting node remains within a predefined bound, the experienced network latency is low, thereby improving the user's perceived QoE.

6.2.1 Optimization problem

The application placement problem is formulated as a multi-objective optimization problem with two principal goals: (a) minimizing end-user latency by placing applications as close as possible to requesting users, and (b) maintaining balanced resource utilization across edge nodes by selecting nodes that can safely accommodate application workloads without approaching saturation. The resource utilization objective considers CPU, memory, and network efficiency, promoting balanced and efficient use of system resources rather than minimizing utilization alone.

Let $\mathcal{T} = \{1, 2, \dots, T\}$ denote the set of discrete time steps. The normalized resource utilization of node n_i at time t is defined as:

$$\text{ResUtil}_i^t = \left[\frac{\sum_j \text{cpu}_j^t}{\text{CPU}_i^t}, \frac{\sum_j \text{mem}_j^t}{\text{Mem}_i^t}, 1 - \frac{\text{NetRx}_i^t + \text{NetTx}_i^t}{\text{NetCap}_i} \right], \quad (6.4)$$

where a higher network value indicates more efficient (less congested) usage. The aggregate normalized resource utilization across all nodes and time steps is:

$$\text{ResourceUsage} = \frac{1}{|\mathcal{N}| \cdot |\mathcal{T}|} \sum_{t \in \mathcal{T}} \sum_{n_i \in \mathcal{N}} \left(\text{CPU}_{\text{norm},i}^t + \text{Mem}_{\text{norm},i}^t + \text{Net}_{\text{norm},i}^t \right), \quad (6.5)$$

and the average user-perceived latency is modelled as:

$$\text{AvgLatency} = \frac{1}{|\mathcal{A}| \cdot |\mathcal{T}|} \sum_{t \in \mathcal{T}} \sum_{a_j \in \mathcal{A}} \text{Latency}_{a_j}^t. \quad (6.6)$$

The final multi-objective optimization problem is:

$$\min (a \cdot \text{AvgLatency} + \beta \cdot \text{ResourceUsage}), \quad (6.7)$$

where α and β are scalar weights defining the relative importance of latency minimization versus resource efficiency. This formulation enables adaptive, resource-aware application placement under dynamic conditions, jointly balancing end-user QoE and efficient utilization of limited edge node resources.

6.3 Core functionality of the three-layer AI-based approach

6.3.1 Conceptual Architecture

The proposed approach is organized into three logical layers: the *Users Layer* at the bottom, the *Infrastructure Layer* in the middle, and the *Control Plane* at the top, extending into the Cloud Tier. Figure 6.1 depicts the conceptual architecture and execution workflow of the proposed three-layer AI-driven orchestrator across the edge–cloud continuum.

The *Users Layer* represents mobile users interacting with applications deployed on geographically distributed edge nodes. The physical proximity between users and hosting nodes directly affects latency and QoE, and as user mobility and request patterns evolve, the orchestrator dynamically adapts application placement to maintain acceptable response times.

The *Infrastructure Layer* consists of multiple resource-constrained edge nodes, each hosting an application together with a lightweight local intelligent agent responsible for workload data collection and preprocessing, IL, FL participation, and resource utilization forecasting. Workload telemetry data is continuously collected locally and transformed into time-series inputs for IL models, enabling adaptation to evolving system behavior while preserving previously learned knowledge. When local model performance degrades, the FL mechanism is triggered, transmitting only model parameters to the FL aggregator at the master node while raw

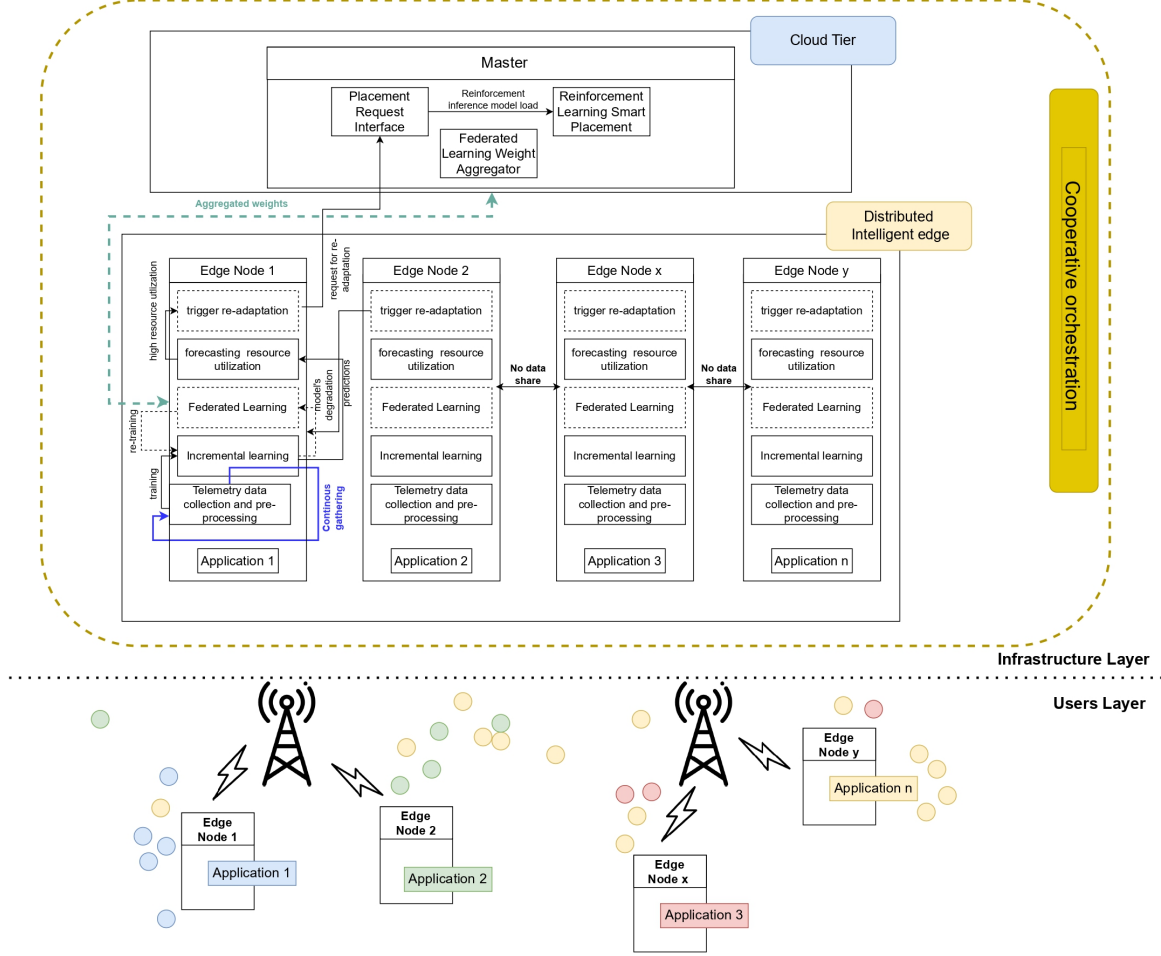


Figure 6.1: Conceptual architecture of the proposed three-layer AI-driven orchestration mechanism across the edge-cloud continuum, illustrating the Users Layer, Infrastructure Layer, and Control Plane with the IL, FL, and RL components.

workload data remain local, preserving privacy and enabling collaborative learning across heterogeneous devices.

The *Control Plane*, implemented at the master node and optionally supported by the Cloud Tier, hosts both the FL aggregator and the centralized RL inference module. When an edge node predicts resource saturation, it sends a re-adaptation request through the placement interface. The RL module evaluates the global cluster state, incorporating resource availability, predicted load, and user proximity, and produces optimized placement actions: scaling on the current node or mi-

grating the application to an alternative node. The three-layer design of the proposed architecture reflects a deliberate separation of concerns across the orchestration stack. The first layer handles local, node-level intelligence through continuous IL-based forecasting, ensuring that each edge node maintains an up-to-date model of its own resource dynamics without depending on external coordination. The second layer provides collaborative refinement through event-driven FL aggregation, activating only when local model performance degrades and exchanging only model parameters rather than raw telemetry, thereby preserving data privacy across the distributed infrastructure. The third layer delivers globally informed, centralized decision-making through the RL module, which exploits full cluster observability to produce placement actions that jointly optimize user proximity and resource utilization across the entire edge–cloud deployment. Figure 6.2 depicts the complete pipeline flow, including the threshold-based escalation logic across the IL, FL, and RL layers.

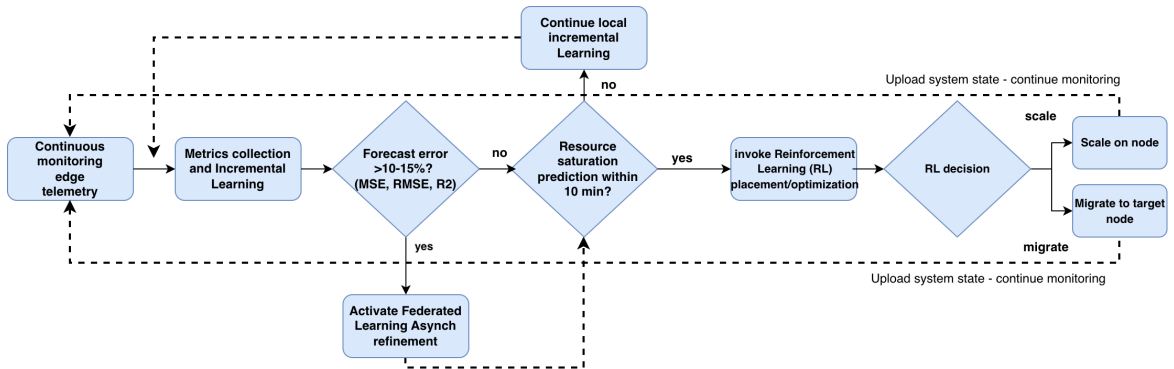


Figure 6.2: Complete pipeline flow with threshold-based escalation across the IL, FL, and RL layers, illustrating the event-driven activation conditions governing layer transitions.

Algorithm 2 describes the complete end-to-end workflow of the proposed orchestration approach, detailing the escalation conditions that govern layer transitions and the interactions between all three layers across the full orchestration lifecycle.

Algorithm 2 Three-layer AI-driven orchestration pipeline for addressing the service placement problem

Require: $\mathcal{N}, \mathcal{A}, B, \mathcal{F}_\theta, \mathcal{G}, \pi_\phi, \delta, \tau$
Ensure: $a_t \in \{\text{scale}, \text{migrate_node}_k\}$

```

1: while Orchestrator is active do
2:   for each  $n_i \in \mathcal{N}$  in parallel do
3:     Collect telemetry batch  $\mathbf{r}_i^{(t)}$  over  $B$  observations
4:     Preprocess: impute missing values, apply min-max normalisation
5:     Update IL model using Elastic Weight Consolidation (EWC) loss (Eq. 6.13)
6:     Forecast:  $\hat{r}_i^{(t+1)} = \mathcal{F}_{\theta_i}(\mathbf{r}_i^{(t)})$ 
7:     Compute  $\text{MSE}_i^{(t)}, \text{RMSE}_i^{(t)}, R_i^{2(t)}$  ✎  $R^2$  reported, not used for escalation
8:     if  $\text{RMSE}_i^{(t)} - \text{RMSE}_i^{(t-1)} > \delta$  or  $\text{MSE}_i^{(t)} - \text{MSE}_i^{(t-1)} > \delta$  or patience exceeded then
9:       Trigger FL for  $n_i$  ✎ Escalate to Layer 2
10:    end if
11:    if  $\hat{r}_i^{\text{CPU}} > \tau$  or  $\hat{r}_i^{\text{Mem}} > \tau$  then
12:      Send re-adaptation request to control plane ✎ Escalate to Layer 3
13:    end if
14:  end for
15:  if FL trigger received then
16:    for each triggered  $n_i$  do
17:      Train locally, transmit  $\Delta\theta_i$  to  $\mathcal{G}$  ✎ Raw data remain local
18:    end for
19:     $\theta_{\text{global}} \leftarrow \mathcal{G}(\{\Delta\theta_i\})$ 
20:    Redistribute  $\theta_{\text{global}}$ , resume IL
21:  end if
22:  if re-adaptation request received from  $n_i$  then
23:    Collect global state  $S_t$ , apply action masking
24:     $a_t \sim \pi_\phi(a \mid S_t)$ 
25:    Compute  $R_t = R_{\text{latency}} + R_{\text{overload}} + \text{ADE}$  (Eq. 6.20)
26:    if  $a_t = \text{scale}$  then
27:      Scale on  $n_{\text{curr}}$ 
28:    else
29:      Migrate to  $n_k$ , update  $n_{\text{curr}}$ 
30:    end if
31:    Update  $S_{t+1}$ 
32:  end if
33: end while

```

6.3.2 Data Collection and Analysis

Effective workload management in edge computing requires intelligent systems capable of optimizing latency and resource utilization under continuously changing conditions. Real-time infrastructure monitoring is essential for supplying the data needed to support informed placement and resource allocation decisions

[Menon et al., 2024]. In the proposed approach, system-level metrics are continuously collected during application execution using Prometheus and Node Exporter, deployed as lightweight monitoring agents on every edge node. These tools, as described in Chapter 2, Section 2.4.5, capture a comprehensive set of indicators including CPU activity, memory consumption, disk I/O, network throughput, and file-system status, providing the visibility required to maintain system stability and operational efficiency.

From the collected telemetry, five key metrics are extracted to characterize system behavior: (1) CPU usage, (2) memory usage, (3) network receive rate, (4) network transmit rate, and (5) system load defined as the average number of processes awaiting CPU access. The normalized CPU usage is defined as:

$$CPU_{\text{norm}} = 1 - \frac{CPU_{\text{idle}}}{\sum CPU_{\text{mode}}}, \quad (6.8)$$

the normalized memory usage as:

$$RAM_{\text{norm}} = 1 - \frac{Memory_{\text{available}}}{Memory_{\text{total}}}, \quad (6.9)$$

the normalized network transmit utilization as:

$$Net_TX_{\text{norm}} = \frac{\text{rate}(\text{node_network_transmit_packets_total})}{P_{\text{max}}}, \quad (6.10)$$

the normalized network receive utilization as:

$$Net_RX_{\text{norm}} = \frac{\text{rate}(\text{node_network_receive_bytes_total})}{B_{\text{max}}}, \quad (6.11)$$

and the normalized system load as:

$$Load_{norm} = \frac{node_load1}{Core_{count}}, \quad (6.12)$$

where P_{max} is the maximum expected packet transmission rate, B_{max} is the maximum expected bandwidth, and $Core_{count}$ is the total number of CPU cores available on the node. Table 6.1 presents a representative sample of the telemetry data collected from edge nodes during application runtime.

Table 6.1: Representative sample of telemetry data collected from edge nodes during application runtime, illustrating the five normalized metrics used as inputs to the IL forecasting models

Timestamp	CPU usage	Memory usage	Net RX norm	Net TX norm	Load 1m norm
25-05-15 10:35:40	0.107	0.402	2.68e-6	1.49e-5	0.071
25-05-15 10:36:40	0.111	0.411	6.57e-7	3.30e-6	0.054
25-05-15 10:37:40	0.111	0.411	6.57e-7	3.30e-6	0.054
25-05-15 10:38:40	0.103	0.406	1.29e-6	7.17e-6	0.075
25-05-15 10:39:40	0.103	0.406	1.29e-6	7.17e-6	0.075

Figure 6.3 presents the pairwise scatter plot matrix of the five collected telemetry metrics, providing a visual characterisation of their joint distributional properties. The diagonal panels display the marginal distribution of each metric individually, while the off-diagonal panels show the bivariate scatter between every pair of metrics. Several structural observations are immediately apparent from this analysis. The CPU usage distribution is right-skewed, concentrated in the range 0.08–0.12 with occasional spikes indicative of transient computational bursts. Memory usage exhibits a narrow, concentrated distribution around 0.400–0.408, characteristic of the stable memory footprint of containerized workloads with periodic garbage collection events. The network metrics (Net_RX_norm and Net_TX_norm) are highly concentrated near their respective baselines with sparse high-value outliers, reflecting the bursty and episodic nature of inter-node communication in edge deployments. The system load metric ($Load_1m_norm$) shows a moderate right-skewed distribution in the range 0.05–0.22.

The off-diagonal scatter plots reveal predominantly diffuse, structureless point clouds across all metric pairs, with no visually discernible linear or monotonic relationships. This absence of pairwise correlation confirms that the telemetry metrics exhibit complex non-linear inter-dependencies that cannot be captured by traditional statistical or linear forecasting models — a finding that directly motivates the adoption of neural network-based architectures for resource utilization prediction, as elaborated in Section 6.3.3.

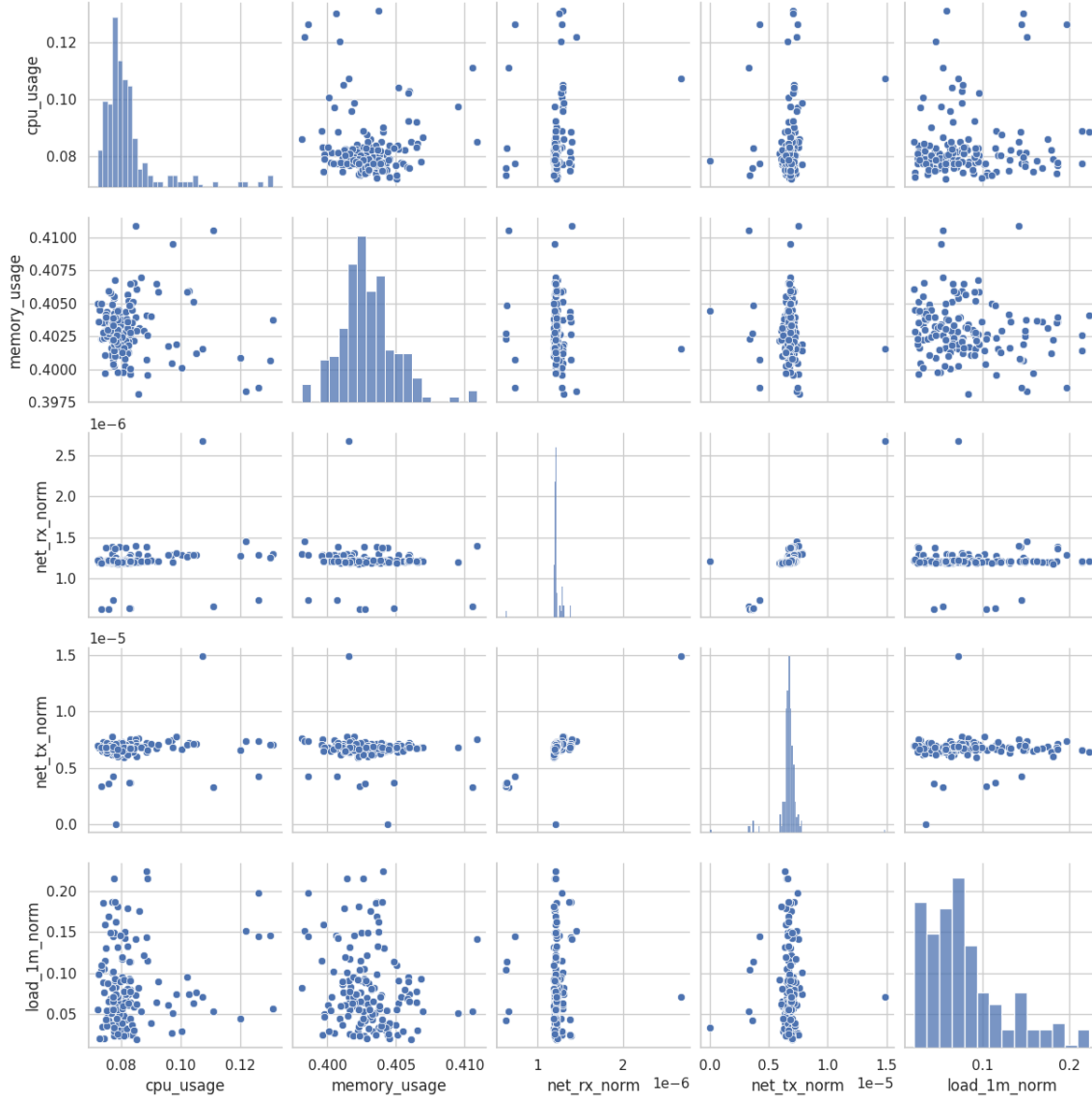


Figure 6.3: Pairwise scatter plot matrix of the five normalized telemetry metrics collected from edge nodes during application runtime: CPU usage, memory usage, normalized network receive rate (Net_RX_norm), normalized network transmit rate (Net_TX_norm), and normalised system load ($Load_1m_norm$). Diagonal panels show the marginal distribution of each metric, off-diagonal panels show bivariate scatter plots between metric pairs. The absence of discernible linear structure across all pairs confirms the presence of complex non-linear temporal dependencies, motivating the adoption of neural network-based forecasting architectures.

Following the accumulation of a sufficient batch of time-series metrics, a data pre-processing pipeline is initiated incorporating missing value imputation and min-max normalization. Missing value imputation addresses absent time-dependent observations caused by data loss, transmission errors, or improper storage. Rolling mean imputation over a 5-observation window is applied, selected as most appropriate given the statistical properties of the dataset over alternative methods such as linear interpolation [Oh et al., 2020], k -nearest neighbour imputation [Murti et al., 2019], and constant value imputation.

6.3.3 Training phase

Following preprocessing, model selection for time-series forecasting is addressed. Traditional statistical and linear models were initially evaluated but found unsuitable, as the correlation analysis of the collected telemetry reveals predominantly low or near-zero correlation across feature pairs, indicating the presence of complex non-linear relationships in the data. Figure 6.4 presents the correlation heatmap of the collected telemetry metrics, confirming this non-linear structure and motivating the adoption of neural network-based architectures.

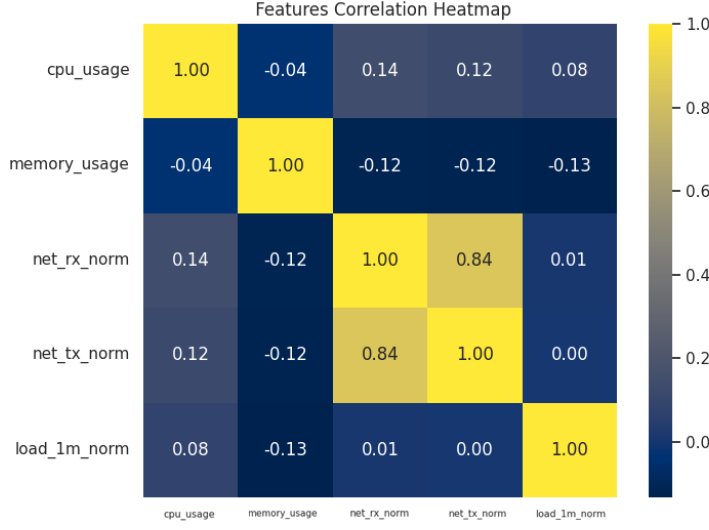


Figure 6.4: Correlation heatmap of the collected telemetry metrics, revealing low or near-zero pairwise correlations indicative of complex non-linear temporal dependencies in edge node resource dynamics.

Two separate forecasting models are developed to predict the future CPU and memory utilization of each edge node independently. The LSTM network architecture [Yu et al., 2019] is selected due to its capacity to capture complex, non-linear temporal patterns and to selectively retain or discard information across time steps through its gated memory cell mechanism. Among all evaluated architectures, the attention-enhanced LSTM variant was identified as the best-performing model, as established empirically in the evaluation presented in Section 6.4. This choice is consistent with prior work demonstrating the effectiveness of contextually-aware ML models for dynamic resource allocation at the edge [Symvoulidis et al., 2024].

6.3.4 Incremental learning layer

In the first layer of the proposed orchestrator, the predictive model follows an incremental learning procedure. Each model, deployed on an individual edge node, is initially trained on a sufficiently large batch of pre-processed time-series data,

after which its state is stored and updated continuously as new observations arrive. This design enables the model to refine previously learned patterns while integrating new information, improving robustness and forecasting accuracy under dynamic and non-stationary conditions.

A well-known challenge in continual learning is catastrophic forgetting, whereby neural networks lose previously acquired knowledge when trained on new data [Liu et al., 2018]. To address this, the orchestrator incorporates Elastic Weight Consolidation (EWC) [Batra and Clark, 2024], which protects important parameters by adding a regularization term to the loss function that penalizes large deviations from values essential to earlier tasks:

$$\mathcal{L}_{EWC} = \mathcal{L}_{task} + \frac{\lambda}{2} \sum_k F_k (\theta_k - \theta_k^*)^2, \quad (6.13)$$

where $\mathcal{L}_{task}$ is the task-specific loss, F_k is the diagonal of the Fisher Information Matrix [Thorne and Vlachos, 2020], θ_k^* are the parameter values learned during previous tasks, and λ controls the strength of the regularization penalty. This formulation ensures that parameters identified as important to earlier tasks are preserved during incremental updates, enabling each node-level model to retain past knowledge while continuously adapting to evolving system behavior.

6.3.5 Federated learning layer

The FL layer functions as an optional reinforcement layer, activated when incremental models fail to converge or exhibit degraded performance — specifically, when MSE or RMSE values exceed predefined thresholds. Since edge nodes operate as heterogeneous, geographically distributed entities, exchanging raw telemetry is both bandwidth-intensive and may expose sensitive user or infrastructure information. FL mitigates these concerns by enabling local training and sharing

only model parameters, supporting collaborative knowledge building while preserving data privacy [Li et al., 2020].

To address the variability inherent in edge environments, an asynchronous FL paradigm (FedAsync) is adopted [Chen et al., 2020]. Federated aggregation is event-driven rather than continuous and is triggered only when local incremental models exhibit convergence degradation. Nodes transmit parameter updates independently upon local training completion, while the central aggregator incrementally refines and redistributes the global model. This design improves scalability, responsiveness to dynamic conditions, and overall learning stability when local incremental training alone proves insufficient. This event-driven federated design is consistent with the broader direction toward scalable, federated container orchestration frameworks for the edge–cloud continuum [Sofia et al., 2026].

6.3.6 Reinforcement learning layer

The RL layer constitutes the final and most consequential decision layer of the proposed orchestrator. It is formally modelled as a Markov Decision Process (MDP) defined by the tuple $\mathcal{M} = (S, A, P, R, \gamma)$, whose components are specified as follows.

The **state space** (S) represents the system state, incorporating node-level resource utilization (CPU, memory, network), the number of active users, the current hosting node identifier, and user-to-node proximity information. The **action space** (A) defines the set of admissible orchestration actions, comprising application scaling on the current node and migration to any alternative edge node, invalid migration actions targeting the currently hosting node are prevented through action masking. The **transition dynamics** (P) describe how the system state evolves following each action, accounting for both the effects of scaling or migration decisions and stochastic workload variations. The **reward function** (R) evaluates orchestration

decision quality by jointly considering service latency, resource utilization, and decision effectiveness. The **discount factor** (γ) controls the trade-off between immediate and long-term performance optimization.

6.3.6.1 State space and action space

The state is represented as an n -dimensional vector combining resource utilization metrics and contextual information. For an environment with $N = 3$ worker nodes, the state vector is:

$$\mathbf{s} = \begin{bmatrix} \text{cpu}_{node_0}, \text{mem}_{node_0}, \text{net}_{node_0} \\ \text{cpu}_{node_1}, \text{mem}_{node_1}, \text{net}_{node_1} \\ \text{cpu}_{node_2}, \text{mem}_{node_2}, \text{net}_{node_2} \\ n_{\text{users}}, \text{node_id}, \text{closest_node} \end{bmatrix}, \quad (6.14)$$

where n_{users} denotes the total number of users requesting the hosted application, node_id identifies the node currently hosting the application, and closest_node denotes the average closest node to the requesting users. The action space is discrete, encompassing application scaling on the current node and migration to any of the available alternative nodes. For $N = 3$ nodes, the action space is:

$$A = \begin{cases} 0 & \text{scale on current node,} \\ 1 & \text{migrate to } node_0, \\ 2 & \text{migrate to } node_1, \\ 3 & \text{migrate to } node_2. \end{cases} \quad (6.15)$$

6.3.6.2 Reward function design

The reward function is designed to evaluate orchestration decisions based jointly on resource availability and service latency, encouraging the agent to deploy ser-

vices on nodes that are both lightly loaded and geographically proximate to users. Three main components are defined.

The **latency penalty** is:

$$R_{\text{latency}} = -L(x), \quad (6.16)$$

where $L(x)$ reflects both node resource saturation and the physical distance between the selected node and the majority of requesting users.

The **overload penalty** is:

$$R_{\text{overload}} = -a \cdot \sum_{r \in \{c, m, n\}} \max(0, r_x - \theta), \quad (6.17)$$

where $a \in (0, 1)$ is a tunable penalty coefficient, $r \in \{c, m, n\}$ corresponds to CPU, memory, and network resources respectively, r_x is the observed resource utilization of the current node, and θ is the acceptable utilization threshold.

The **Agent Decision Effect (ADE)** provides additional feedback tailored to the quality and impact of the selected action. For migration actions, the ADE is:

$$ADE_{\text{migrate}} = \begin{cases} +\beta_{\text{gain}} & \text{if } \Delta r \geq \delta, \\ -\beta_{\text{loss}} & \text{if } \Delta r \leq -\delta, \\ \eta \mathbf{I}[x = x^*] - \lambda d(x, x^*) & \text{otherwise,} \end{cases} \quad (6.18)$$

where $\Delta r = \bar{r}_{\text{prev}} - \bar{r}_x$ denotes the change in average resource load before and after migration, δ is a positive threshold for significant improvement, β_{gain} and β_{loss} are reward and penalty coefficients, $\mathbf{I}[x = x^*]$ is a proximity indicator returning 1 if the application is placed on the geographically closest node, η and λ are proximity reward and penalty coefficients, and $d(x, x^*)$ quantifies the distance between the

selected and the closest node.

For scaling actions, the ADE is:

$$ADE_{\text{scale}} = \begin{cases} +\gamma_{\text{low}} & \text{if } \bar{r}_x < \rho_1 \text{ and } u \leq u_1, \\ +\gamma_{\text{mid}} & \text{if } \bar{r}_x < \rho_2 \text{ and } u \leq u_2, \\ +\gamma_{\text{high}} & \text{if } \bar{r}_x > \rho_3 \text{ and } u \geq u_3, \\ -P_{\text{scale}} & \text{otherwise,} \end{cases} \quad (6.19)$$

where u is the number of active users, $\rho_{1,2,3}$ and $u_{1,2,3}$ are resource utilization and user demand thresholds, γ_{low} , γ_{mid} , γ_{high} are reward coefficients for desirable scaling actions under varying load conditions, and P_{scale} is a penalty for inefficient scaling.

The total reward is the cumulative sum of the three components:

$$R = R_{\text{latency}} + R_{\text{overload}} + ADE. \quad (6.20)$$

The complete operational workflow of the RL-based placement layer, including the event-driven re-adaptation logic and the decision procedure, is summarized in Algorithm 3.

Algorithm 3 Event-driven RL-based edge application placement layer

Require: Trained PPO policy π_θ , current global system state S_t , resource thresholds θ , optimization weights (α, β)
Ensure: Optimal placement action $a_t \in \{\text{scale}, \text{migrate_node}_k\}$

- 1: **while** Orchestrator is active **do**
- 2: Receive re-adaptation trigger from edge node n_i
- 3: Collect: resource utilization (*CPU, Mem, Net*), hosting node n_{curr} , active users u , closest node n_{close}
- 4: Construct state vector: $S_t = f(\text{resource_metrics}, \text{context_metrics})$
- 5: Select action: $a_t \sim \pi_\theta(a|S_t)$
- 6: Compute reward: $R_t = R_{\text{latency}} + R_{\text{overload}} + ADE$
- 7: **if** $a_t = \text{scale}$ **then**
- 8: Apply horizontal scaling on n_{curr}
- 9: **else**
- 10: Migrate application to selected target node n_k
- 11: **end if**
- 12: Update global system state S_{t+1}
- 13: **end while**

6.4 Evaluation

6.4.1 Experimental setup

The proposed orchestration mechanism is implemented within a Kubernetes cluster using Kubernetes-in-Docker (KinD) [Kubernetes SIGs,], deployed on a host system equipped with an AMD CPU (16 cores), 32 GB RAM, and 1 TB SSD storage. The experimental setup consists of one control plane node representing the cloud tier and three worker nodes simulating geo-distributed edge devices running lightweight user-oriented applications. Prometheus and Node Exporter are deployed across all nodes to provide continuous telemetry collection and system monitoring.

Each worker node is provisioned with 3 virtual CPU cores and 6 GB RAM, supporting concurrent execution of lightweight applications, monitoring agents, IL models, and FL client operations for CPU and memory usage prediction. The con-

trol plane node is provisioned with approximately 4 virtual CPU cores and 8 GB RAM to support Kubernetes control-plane services, centralized Prometheus aggregation, asynchronous federated aggregation, and the centralized RL inference module deployed as a Kubernetes pod. Simulated client requests (10–300 per experiment) follow a Gaussian distribution to reflect realistic, fluctuating demand patterns. Agents collect system data in batches of 30 observations, corresponding to approximately one minute of system behavior at the telemetry collection rate of one observation every two seconds.

6.4.2 Training models comparison

The first stage of the evaluation focuses on identifying the optimal neural network architecture for the distributed forecasting agents responsible for CPU and memory prediction. Four DL architectures are comparatively evaluated: Convolutional Neural Networks (CNNs) [Chen et al., 2016], Gated Recurrent Units (GRUs) [Rahman et al., 2019], Bidirectional LSTM (BiLSTM) networks [Siarni-Namini et al., 2019b], and attention-enhanced LSTM networks [Vaswani et al., 2017]. Traditional statistical and linear forecasting methods — including Random Forests [Breiman, 2001], ARIMA [Newbold, 1983], and exponential smoothing [Gardner Jr, 1985] — are excluded from consideration because the extracted features exhibit weak linear correlation, limiting the effectiveness of such approaches in capturing the spatiotemporal dependencies present in the edge telemetry.

All models are evaluated within the four-node KinD cluster integrated into the proposed orchestration approach. To ensure reliability and reproducibility, each experiment is executed three times, and results are aggregated across all worker nodes and runs. Figures 6.5 and 6.6 present the comparative performance of each architecture in terms of MSE, RMSE, and R^2 across the incremental training phase, including the contribution of the FedAsync mechanism activated selectively based on prediction divergence criteria.

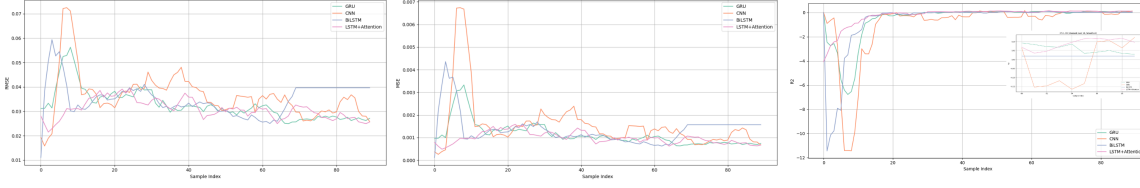


Figure 6.5: Evolution of RMSE, MSE, and R^2 for CPU resource prediction across successive iterations during the incremental learning process, including the trigger points of the federated aggregation mechanism.

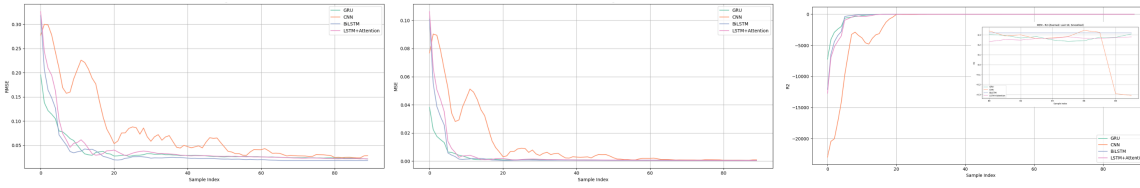


Figure 6.6: Evolution of RMSE, MSE, and R^2 for memory resource prediction across successive iterations during the incremental learning process, including the trigger points of the federated aggregation mechanism.

Table 6.2 presents the average MSE, RMSE, and R^2 values across three independent experimental runs and all three worker nodes for each evaluated architecture.

Table 6.2: Average MSE, RMSE, and R^2 across three independent experimental runs and all worker nodes, for each evaluated DL architecture.

Model	CPU MSE	CPU RMSE	CPU R^2	MEM MSE	MEM RMSE	MEM R^2
GRU	0.001135	0.032530	-0.508430	0.001605	0.033266	-129.057501
BiLSTM	0.001262	0.033668	-0.758675	0.002615	0.033191	-227.000000
CNN	0.001542	0.035956	-0.936868	0.009766	0.070557	-1139.588058
LSTM_att	0.000983	0.030771	-0.144565	0.002883	0.037053	-215.175454

The experimental results demonstrate that the LSTM with attention achieves the lowest RMSE and MSE values for CPU prediction, while the Gated Recurrent Unit (GRU) and Bi-directional LSTM architectures achieve marginally lower error for memory prediction. The attention-enhanced LSTM is selected as the production forecasting model given its superior CPU prediction accuracy and its alignment

with the operational demands of distributed edge environments. It is important to note that the negative R^2 values reported, particularly for memory forecasting, reflect two compounding factors. First, during the cold-start phase of incremental learning, models have not yet accumulated sufficient observations to converge, producing predictions that are temporarily worse than a naïve mean predictor during this transient window. Second, edge memory telemetry is inherently non-stationary, exhibiting abrupt level shifts between prolonged flat regions characteristic of garbage collection and container lifecycle events, when memory variance within short windows approaches zero, even modest residuals produce strongly negative R^2 values due to a denominator effect rather than a model failure. For these reasons, MSE and RMSE are used as the primary FedAsync activation criteria and the principal reliability indicators for the IL layer.

Table 6.3 presents a comparative analysis of the attention-enhanced LSTM and GRU architectures across key performance indicators relevant for deployment in resource-constrained edge environments, including model size, training latency, and inference overhead.

Table 6.3: Comparison of the attention-enhanced LSTM and GRU architectures across key performance indicators for resource-constrained edge deployment.

KPI	LSTM with Attention	GRU
Trainable parameters	21,954	17,730
Model size (weights)	86–88 KB	69–71 KB
Training time per update	4.6–5.3 s	4.8–5.3 s
Inference latency (warm)	2–3 s	2–3 s
Inference latency (cold)	23–28 s	21–27 s
Runtime memory (RSS)	Low hundreds MB	Low hundreds MB
Forecast accuracy	Higher	Lower

Although the attention-enhanced LSTM has approximately 4,224 more trainable parameters than the GRU — corresponding to an additional ~18 KB of stored model weights — this overhead is negligible relative to the capabilities of modern edge nodes. Both models exhibit near-identical inference latency and run-

time memory utilization. Given these findings, the attention-based LSTM is selected as the production model for the proposed orchestrator, as its ability to capture long-range temporal dependencies while maintaining a lightweight footprint aligns well with the operational demands of distributed edge computing environments.

6.4.3 Federated learning results

The FL component acts as a regulator, triggered only when local models exhibit poor convergence or unstable training dynamics. A patience parameter governs this activation: if evaluation metrics fail to improve within a predefined window, the FedAsync mechanism is initiated. Triggering conditions are monitored at both the IL layer (where prediction error metrics are continuously tracked) and the FL layer (where convergence degradation of approximately 10–15% from a stable baseline activates global aggregation). Following weight aggregation and redistribution, operation returns to local IL unless new degradation is detected.

Figures 6.7 and 6.8 illustrate the time points at which the FL mechanism was activated for the CPU and memory forecasting models respectively, together with the corresponding performance improvements following weight aggregation and model adaptation. For both resources, the figures show the global evolution of MSE across the full IL horizon alongside localized views around selected FedAsync trigger points, illustrating the pre-trigger error escalation and the post-aggregation error correction behavior.

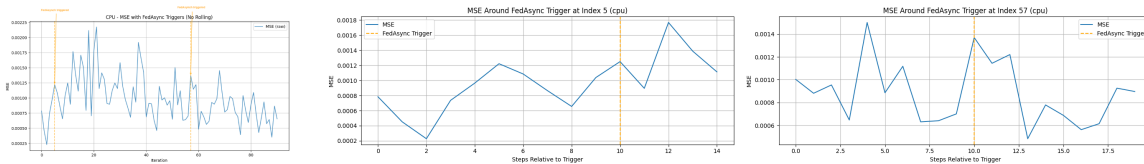


Figure 6.7: Evolution of MSE for CPU resource prediction across successive iterations, with indication of the federated aggregation trigger points and the corresponding post-aggregation error correction behavior.

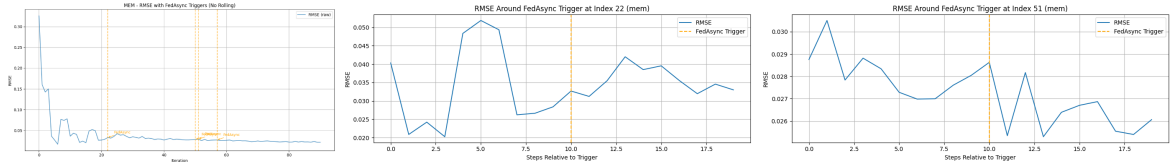


Figure 6.8: Evolution of MSE for memory resource prediction across successive iterations, with indication of the federated aggregation trigger points. The post-aggregation stabilisation of RMSE values confirms the positive impact of the FL mechanism on prediction accuracy and robustness under non-stationary workload conditions.

The results confirm that the proposed FedAsync mechanism effectively enhances the robustness and stability of resource utilization forecasting, particularly under conditions of fluctuating workloads and distributed edge deployments. The memory RMSE reduction following each FedAsync trigger — most clearly visible around the first and second activation points — demonstrates recovery from model drift and improved prediction stability that the local IL layer alone could not achieve.

6.4.4 Resource overhead of the decentralized AI agents

The computational and memory overhead introduced by the distributed AI-driven layers is assessed to confirm the suitability of the proposed architecture for resource-constrained edge environments. The IL agents exhibit moderate CPU usage averaging between 0.2–0.4 CPU cores, with short-duration peaks of 0.5–0.6 cores during incremental update steps. Memory usage typically ranges between 400–800 MB, with occasional peaks reaching 1.1 GB under worst-case training activity. The FL layer introduces minimal additional overhead, exchanging only lightweight parameter updates in an event-driven asynchronous manner. The Node Exporter monitoring component requires approximately 0.002–0.004 CPU cores and 35–40 MB of RAM per node, a negligible footprint that does not interfere with application workloads.

The overhead remains manageable because the IL layer is activated only when

sufficient new data is available, ensuring intermittent rather than continuous updates, the asynchronous FL mechanism minimizes communication by activating solely upon training degradation, and the RL decision module is invoked only upon placement events rather than continuously, limiting control-plane messaging and preserving responsiveness even as node count and geographic dispersion increase.

6.4.5 Reinforcement learning results

The selection of PPO as the production RL algorithm is informed by the preliminary results reported in [Paraskevoulakou et al., 2024], which established that PPO consistently outperforms DQN in terms of convergence speed and generalization performance as application complexity increases. The evaluation results reported below confirm and extend this conclusion to the more demanding edge–cloud placement setting. Several state-of-the-art RL algorithms are evaluated for application re-allocation under fluctuating workloads and resource constraints: Deep Q-Learning (DQN) [Hester et al., 2018], Double DQN (DDQN) [Van Hasselt et al., 2016], Advantage Actor-Critic (A2C) [Lu et al., 2024], and PPO [Schulman et al., 2017b]. All models are trained and evaluated in a custom simulation environment representing the edge–cloud continuum with one control plane and three worker nodes. Figure 6.9 presents a comparative analysis of reward convergence and normalized loss evolution across all evaluated algorithms.

PPO achieves the highest average reward with the lowest variance across all evaluated algorithms, and maintains consistently low normalized loss throughout training. Figure 6.10 further evaluates the success rate — defined as the percentage of episodes exceeding a reward threshold of 1,000 — and the number of correct scaling decisions. PPO achieves a success rate of 26.2% and 106,510 correct scaling decisions, significantly outperforming all alternatives. These results confirm that policy-gradient-based methods better handle the complexity of dynamic applica-

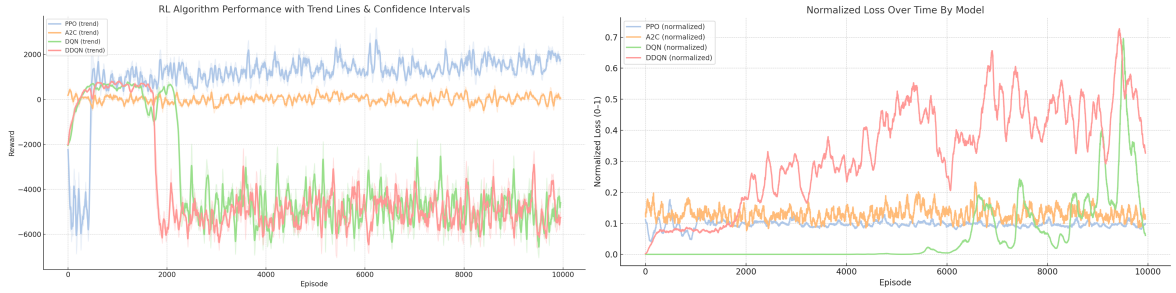


Figure 6.9: Reward convergence comparison (left) and normalized loss evolution (right) of the evaluated RL algorithms over 10,000 training episodes.

tion placement, and PPO is selected as the production inference model for the control plane.

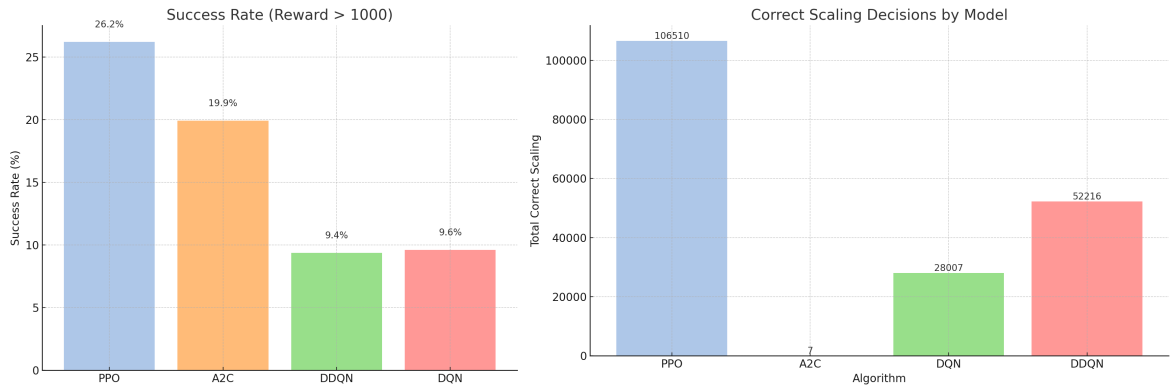


Figure 6.10: Comparison of RL models in terms of success rate (percentage of episodes exceeding a reward threshold of 1,000) and correct scaling decisions.

The PPO training configuration is presented in Table 6.4.

Reward coefficients were determined empirically on a held-out subset. The overload penalty coefficient α was selected from the set $\{0.1, 0.3, 0.5, 0.7\}$, with $\alpha = 0.3$ identified as providing the optimal balance between saturation avoidance and migration frequency. Migration coefficients β_{gain} and β_{loss} were each tuned across $\{1.0, 1.5, 2.0\}$, with 1.5 yielding stable convergence. Scaling thresholds $\rho_{1,2,3}$ and $u_{1,2,3}$ were derived from the training data distributions.

Table 6.4: Training configuration of the PPO-based RL agent as implemented in the proposed orchestration approach.

Parameter	Value
Action space	Discrete, $ \mathcal{A} = 4$
State dimension	12
Discount factor (γ)	0.99
Learning rate	2×10^{-4}
Optimizer	Adam
PPO clipping range (ϵ)	0.2
Update epochs per PPO step (K)	5
Entropy regularization coefficient	0.01
Value loss coefficient	0.5
Gradient clipping (global norm)	0.5
Policy network	Shared MLP: 2 layers $\times$ 128 (ReLU)
Actor head	128 $\rightarrow$ 128 (ReLU) $\rightarrow$ 4 (Softmax)
Critic head	128 $\rightarrow$ 128 (ReLU) $\rightarrow$ 1

6.4.6 Placement inference results

To assess the placement decision quality of the PPO agent, 100 realistic scenarios are constructed, each simulating an edge node triggering a re-adaptation request due to dynamic resource or demand changes. All scenarios are annotated using rule-based expert logic to define ground-truth optimal actions, providing a consistent benchmarking baseline. Table 6.5 presents a representative subset of the evaluation scenarios together with their ground-truth labels.

Table 6.5: Representative subset of the 100 evaluation scenarios, with node resource metrics, contextual information, and ground-truth placement labels.

Sc.	n0 CPU	n0 Mem	n0 Net	n1 CPU	n1 Mem	n1 Net	n2 CPU	n2 Mem	n2 Net	Curr.	Users	Closest	Label
2	0.73	0.18	0.24	0.26	0.05	0.18	0.39	0.73	0.55	2	64	1	scale
11	0.14	0.68	0.50	0.13	0.50	0.49	0.25	0.34	0.30	2	96	2	scale
26	0.18	0.13	0.08	0.61	0.72	0.58	0.83	0.41	0.94	2	80	0	migrate.n0
43	0.75	0.17	0.35	0.73	0.46	0.75	0.25	0.30	0.34	2	104	2	scale
54	0.23	0.09	0.07	0.71	0.44	0.44	0.92	0.64	0.95	2	402	2	migrate.n0

The PPO-based RL inference model is evaluated across all 100 scenarios and benchmarked against five supervised ML classifiers — Random Forest, Decision Tree, Lo-

gistic Regression, Naïve Bayes, and MLP — and two classical heuristic placement strategies: First-Fit (FF) and Best-Fit Bin Packing (BF-BP) [Li et al., 2014]. Table 6.6 presents the comprehensive performance comparison in terms of accuracy, precision, recall, and F1-score across all placement decision categories.

Table 6.6: Performance comparison across supervised ML models, classical heuristic baselines, and the PPO-based RL agent across all placement decision categories.

Metric	RF	DT	LR	NB	MLP	FF	BF-BP	PPO
Overall Accuracy	0.60	0.63	0.624	0.60	0.64	0.39	0.33	0.97
Migrate n0 Precision	0.57	0.53	0.55	0.54	0.55	0.36	0.20	1.00
Migrate n1 Precision	0.55	0.55	0.67	0.67	0.60	0.40	0.29	1.00
Migrate n2 Precision	0.75	0.74	0.73	0.79	0.84	0.71	0.64	0.97
Scale Precision	0.59	0.61	0.59	0.53	0.62	0.32	0.28	0.91
Migrate n0 Recall	0.71	0.67	0.71	0.62	0.67	0.17	0.04	0.95
Migrate n1 Recall	0.50	0.46	0.42	0.42	0.50	0.25	0.17	1.00
Migrate n2 Recall	0.39	0.45	0.52	0.48	0.52	0.32	0.29	0.93
Scale Recall	0.90	0.90	0.90	0.95	0.95	0.90	0.90	1.00
Migrate n0 F1	0.63	0.59	0.62	0.58	0.60	0.23	0.07	0.97
Migrate n1 F1	0.52	0.50	0.51	0.51	0.55	0.31	0.21	1.00
Migrate n2 F1	0.51	0.56	0.60	0.60	0.64	0.44	0.40	0.95
Scale F1	0.72	0.73	0.72	0.68	0.75	0.47	0.43	0.95

Figure 6.11 presents a radar chart providing a holistic visual comparison of all evaluated models across the complete set of performance metrics simultaneously. The chart makes immediately apparent the qualitative gap between the RL-based approach and all other methods: the PPO polygon consistently occupies the outermost region across every metric axis, while the supervised ML models cluster tightly together in a substantially smaller inner region, confirming that the performance advantage of PPO is not limited to any single metric category but extends uniformly across precision, recall, and F1-score for all placement decision classes.

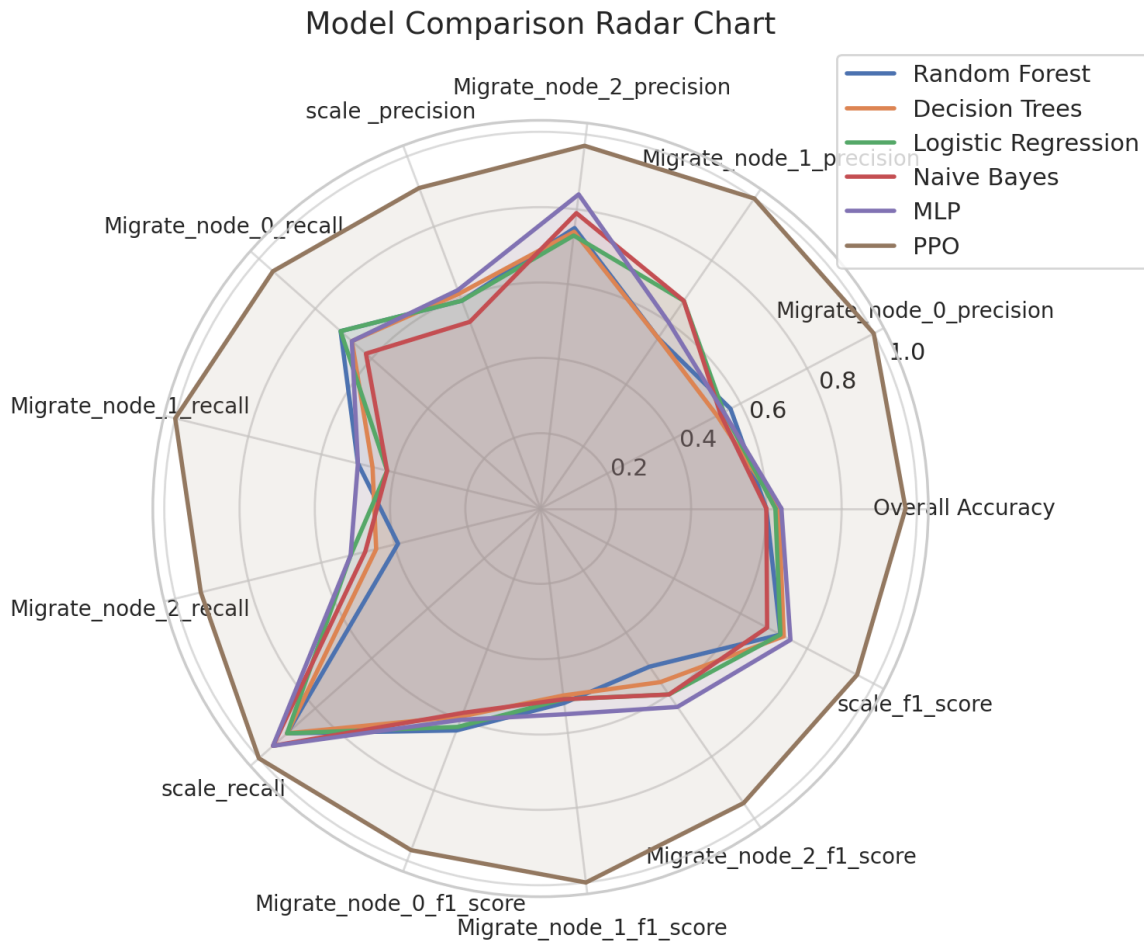


Figure 6.11: Radar chart comparing all evaluated models — Random Forest, Decision Trees, Logistic Regression, Naïve Bayes, MLP, and PPO — across the complete set of performance metrics encompassing precision, recall, and F1-score for all placement action categories and overall accuracy. The PPO polygon consistently occupies the outermost region across all axes, confirming its uniformly superior performance.

Figure 6.12 isolates the recall dimension across all models and placement action categories. Recall is particularly consequential in the placement context: a missed migration decision — classifying a scenario requiring migration as a scaling action — results in a node approaching saturation without the corrective relocation of the application workload, directly compromising both resource availability and user-perceived QoE. The figure reveals that PPO achieves perfect or near-perfect recall

across all migration categories (1.00 for `migrate_node_1`, 0.95 for `migrate_node_0`, and 0.93 for `migrate_node_2`), and perfect recall for scaling decisions (1.00), substantially outperforming all supervised ML baselines. Among the baseline models, recall for migration to `node_1` and `node_2` is particularly low (0.39–0.52), reflecting the inability of static classifiers to generalize migration decisions across the diverse resource and proximity configurations present in the evaluation scenarios.

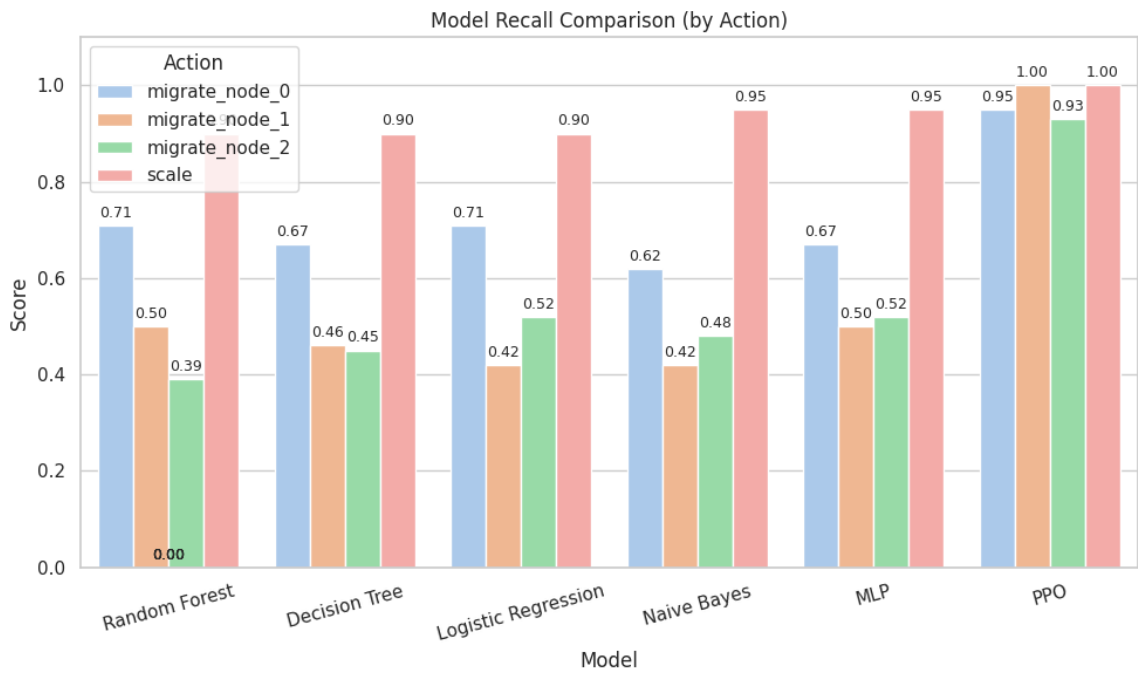


Figure 6.12: Recall comparison across all evaluated models by placement action category: `migrate_node_0`, `migrate_node_1`, `migrate_node_2`, and `scale`. PPO achieves perfect or near-perfect recall across all categories, confirming its superior sensitivity to the full spectrum of placement decision requirements.

Figure 6.13 presents the precision comparison across all models and placement action categories. Precision measures the proportion of placement decisions of a given category that were correct, and is particularly relevant for migration actions: a false positive migration — incorrectly classifying a scenario as requiring migration when scaling would suffice — results in unnecessary workload transfer overhead and transient service disruption. PPO achieves perfect precision for

migration to node_0 and node_1 (1.00), 0.97 for migration to node_2, and 0.91 for scaling decisions, consistently outperforming all supervised ML baselines across every action category. A particularly notable observation concerns the Random Forest baseline, which records a precision of 0.00 for migration to node_1 — indicating that every prediction of this action class was incorrect — a failure mode that reflects the inability of the static classifier to generalize the node_1 migration criterion across the diverse resource and proximity configurations in the evaluation scenarios. Among the supervised ML baselines, MLP achieves the highest migration to node_2 precision (0.84), yet still falls substantially short of PPO’s 0.97, confirming that the sequential decision-making capability of the RL agent provides a systematic advantage that static classification approaches cannot replicate.

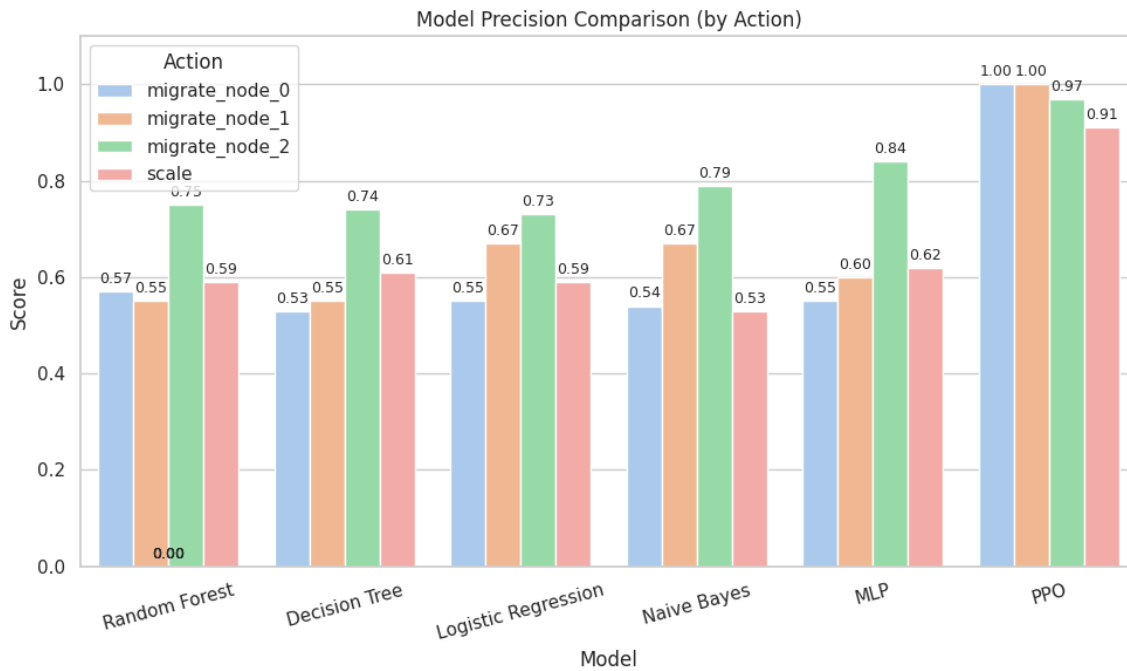


Figure 6.13: Precision comparison across all evaluated models by placement action category: migrate_node_0, migrate_node_1, migrate_node_2, and scale. PPO achieves perfect or near-perfect precision across all migration categories. The zero precision of Random Forest for migrate_node_1 highlights the systematic generalization failures of static classifiers under heterogeneous placement conditions.

PPO achieves an overall decision accuracy of 97%, substantially outperforming all supervised ML baselines (60–64%) and the classical heuristic strategies (33–39%). As illustrated in Figure 6.11, this superiority is not confined to any single metric but extends uniformly across all precision, recall, and F1-score dimensions for every placement action category. The substantially lower performance of FF and BF-BP derives from three key limitations: both rely exclusively on single-snapshot CPU and memory availability without accounting for network utilization, neither incorporates user proximity information, and their static greedy logic cannot adapt to the multi-resource constraints present in the evaluation scenarios where optimal decisions depend jointly on CPU, memory, network, user count, and closest node. As shown in Figure 6.12, the recall gap is especially pronounced for migration decisions: the supervised ML models achieve recall values of only 0.39–0.52 for migration to node_2, compared to 0.93 for PPO, reflecting the fundamental inability of static classifiers to generalize migration decisions across heterogeneous resource and proximity configurations. The precision comparison in Figure 6.13 further confirms this pattern: PPO achieves perfect precision for migration to node_0 and node_1, while the Random Forest baseline records zero precision for the node_1 migration category — a systematic generalization failure with direct operational consequences in live edge deployments. Figure 6.14 provides a further detailed analysis of the PPO model’s placement decisions, examining how the agent balances resource utilization and user proximity.

In 90.5% of cases where the geographically closest node was not selected, the model prioritized nodes with significantly lower resource utilization, thereby avoiding potential performance bottlenecks, only 9.5% of such decisions were attributable to other factors. Migration accuracy reaches 96% compared to 84% for scaling decisions, a gap attributable to the reward parameterization which makes the policy more responsive to load imbalances across nodes than to scaling triggers. In over 80% of non-closest-node selections, a resource load reduction of $\Delta > 0.25$ justified the increased network distance, demonstrating that the agent consistently favors

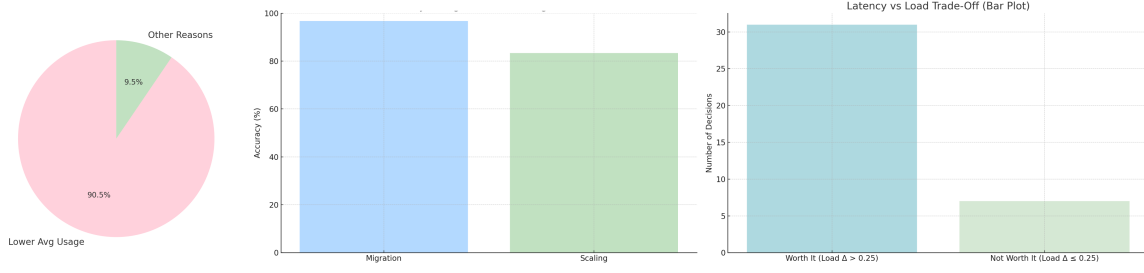


Figure 6.14: Quantitative evaluation of the PPO model's placement decisions: (left) proportion of cases where the geographically closest node was not selected and the reasoning behind such decisions, (centre) migration versus scaling decision accuracy, (right) distribution of resource load reduction Δ for non-closest-node selections.

distant nodes when load-balancing gains outweigh the added latency cost.

6.4.7 Latency–resource utilization trade-off analysis

To provide a comprehensive evaluation of placement quality beyond classification accuracy, a composite latency–resource trade-off metric is introduced:

$$J = aL_{\text{norm}} + (1 - a)U_{\text{norm}}, \quad (6.21)$$

where L_{norm} is the normalized latency derived from user proximity and placement correctness, U_{norm} is the normalized resource utilization across CPU, memory, and network, and $a = 0.5$ reflects equal importance of latency and resource balancing. Table 6.7 presents the trade-off performance comparison across PPO and the two heuristic baselines.

PPO achieves the lowest trade-off score, demonstrating approximately 12% and 13% improvement over FF and BF-BP respectively. This confirms the superior ability of the RL-based orchestrator to jointly minimize latency and maintain balanced resource utilization — an advantage that static heuristic approaches cannot replicate due to their lack of global system awareness and their inability to reason over

Table 6.7: Latency–resource trade-off performance comparison between the PPO-based RL agent and classical heuristic placement strategies. Lower values of J indicate better joint optimization of latency and resource efficiency.

Model	Category	Trade-off Score (J)
PPO	AI-based RL	0.54
First-Fit (FF)	Heuristic	0.613
Best-Fit Bin Packing (BF-BP)	Heuristic	0.621

the joint optimization objective defined in Equation 6.7.

6.5 Summary

This chapter has presented the second of two complementary AI-driven approaches proposed in this thesis for addressing the dynamic service placement problem in heterogeneous edge–cloud environments. Where Chapter 5 approached placement from the service composition level — demonstrating that the internal grouping of serverless function blocks is an independent and consequential dimension of the placement problem — this chapter approached the same problem from the infrastructure resource level, treating the edge node’s computational state as the primary input to a closed-loop, AI-driven placement decision pipeline.

The proposed three-layer orchestration approach addresses the placement problem through a hierarchical sequence of AI mechanisms, jointly addressing the resource-centric research questions of this thesis defined in Chapter 1. The IL layer provides lightweight, node-local forecasting of resource dynamics, enabling proactive re-adaptation before saturation occurs and contributing to **RQ1** and **RQ3** by demonstrating that attention-enhanced LSTM models can capture non-stationary resource dynamics with sufficient accuracy to support proactive, resource-aware placement decisions. The FL layer provides an event-driven, privacy-preserving mechanism for collaborative model refinement when local convergence degrades, contributing to **RQ5** by demonstrating that FedAsync aggregation improves pre-

diction stability across the distributed infrastructure without requiring raw telemetry exchange, thereby addressing the privacy and scalability challenges inherent to decentralized edge deployments. The RL layer provides centralized, globally informed placement decisions that jointly minimize user-perceived latency and resource utilization across the edge cluster, addressing **RQ1**, **RQ3**, and **RQ4** by demonstrating that PPO achieves 97.0% placement accuracy across 100 heterogeneous scenarios — substantially outperforming both supervised ML baselines and classical heuristic strategies, and confirming that fully AI-driven orchestration produces more effective and proactive placement decisions than traditional heuristic-based models. The RL-based placement layer builds on a preliminary cloud-specific investigation [Paraskevoulakou et al., 2024] that established the viability of PPO for application component placement under dynamic workload conditions, extending that foundation to the more demanding and heterogeneous edge–cloud continuum setting addressed in this chapter. The evaluation results establish three findings of broader significance. First, the FedAsync mechanism demonstrably improves prediction stability under non-stationary workload conditions that the IL layer alone cannot handle, validating the value of the collaborative learning layer and its event-driven activation logic. Second, the PPO agent’s consistently superior performance over classical heuristic strategies confirms that static greedy methods are fundamentally inadequate for placement problems requiring joint optimization of resource utilization, user proximity, and multi-resource constraints. Third, the latency–resource trade-off analysis demonstrates that the RL-based orchestrator achieves a 12–13% improvement over heuristic baselines in the composite trade-off metric, providing empirical grounding for the practical deployment value of the proposed approach.

Taken together, the service-centric approach of Chapter 5 and the resource-centric approach of this chapter constitute a unified, two-dimensional response to the dynamic service placement problem in edge–cloud environments. The former establishes that optimal placement cannot be reduced to infrastructure allocation

alone, the latter demonstrates that infrastructure-level AI-driven orchestration, when designed as a closed-loop pipeline integrating forecasting, federated refinement, and reinforcement decision-making, can achieve near-optimal placement accuracy under the dynamic and heterogeneous conditions characteristic of real-world edge deployments. The conclusions drawn from both approaches, together with the contributions of this thesis to the broader research questions established in Chapter 1, are synthesized in Chapter 7.

Chapter 7

Conclusions and Future Directions

Chapter Structure

This Chapter is constructed as follows:

- **Section 7.1 – Critical reflection on the proposed approaches**, offers an assessment of the extent to which the proposed approaches address the dynamic service placement problem in heterogeneous edge–cloud environments, discussing their contributions, limitations, and the assumptions under which their results were obtained.
- **Section 7.2 – Synthesis of contributions**, synthesizes the answers to the five research questions defined in Chapter 1 and consolidates the principal contributions of this thesis within the broader landscape of AI-driven edge–cloud orchestration research.
- **Section 7.3 – Future research directions**, identifies the open challenges and promising research avenues derived from the present work, outlining concrete extensions of the proposed approaches toward next-generation intelligent edge–cloud orchestration systems.

7.1 Critical reflection on the proposed approaches

This thesis set out to address the dynamic service placement problem in heterogeneous edge–cloud environments from two complementary perspectives: the service composition level and the infrastructure resource level. The two approaches developed in Chapters 5 and 6 collectively constitute a principled, empirically validated response to this problem. A critical assessment of their contributions and limitations is presented below.

7.1.1 The service-centric approach

The service-centric approach presented in Chapter 5 made two distinct contributions. The first was the demonstration that a complete ML/DL analytics workflow, encompassing large-scale data ingestion, multi-stage preprocessing, dimensionality reduction, feature engineering, and deep learning inference, can be realized as a chain of serverless functions on edge-compatible infrastructure using Apache OpenWhisk. This proof-of-concept was evaluated on a real-world financial fraud detection pipeline and established that the serverless model is a viable and practical deployment vehicle for computationally intensive, data-driven AI applications at the network edge. The proposed approach constitutes a functional, lightweight solution that enables a developer or data scientist to deploy their entire business logic as a next-generation edge application with zero infrastructure administration burden, lowering the barrier to deployment from a full-stack engineering undertaking to a focused, logic-centric development process.

The second and more consequential contribution was the AI-based Function Regulator, which addressed a previously unrecognized dimension of the service placement problem: the relationship between the internal grouping of serverless function blocks and the end-to-end response latency of the pipeline. The empirical finding that this relationship is non-monotonic, that a 3-function pipeline con-

sistently outperforms a 2-function pipeline despite having more inter-function boundaries, directly refutes the intuitive assumption that fewer functions necessarily means lower latency. The Function Regulator's ability to identify the optimal partitioning automatically, without requiring any manual experimentation from the developer, validates the practical feasibility of treating function composition as an independent and addressable dimension of the placement problem. The resource independence finding, which demonstrated that response time is effectively invariant to the computational capacity of the hosting node across two distinct hardware configurations, provides the strongest empirical argument for the service-centric perspective: if latency is independent of infrastructure, then infrastructure optimization alone is provably insufficient to guarantee QoE.

The approach was evaluated on a single application domain, financial fraud detection, using a single serverless platform, Apache OpenWhisk. The generalizability of the non-monotonic composition finding to other ML/DL pipeline types, other FaaS platforms such as OpenFaaS or AWS Lambda, and other application domains remains an open empirical question. The cold-start penalty, which reached 25 seconds for the full 3-function pipeline on the first invocation, represents a significant practical limitation for latency-sensitive applications that cannot tolerate this initialization overhead. Furthermore, since the regulator enables a trade-off between modularity and performance efficiency, the current implementation does not yet account for the correlations between different pipeline partitions and response time values in a holistic manner, nor does it explicitly model how the compute, I/O, and data intensity of each pipeline step affects the optimal composition under end-to-end scenario constraints.

7.1.2 The resource-centric approach

The resource-centric approach presented in Chapter 6 addressed the infrastructure-level dimension of the placement problem through a closed-loop, three-layer AI-

driven orchestration architecture integrating IL, FL, and RL. Built on Kubernetes-in-Docker, the proposed solution employs a distributed multi-layered AI architecture that processes real-time telemetry streams encompassing CPU, memory, and network metrics, feeding them into incremental forecasting models that enable proactive re-adaptation before resource saturation occurs. The approach's principal strength lies in its hierarchical design, which separates the forecasting, collaborative refinement, and placement decision responsibilities across three distinct layers, each activated under specific conditions and each contributing a capability that the others cannot provide.

The attention-enhanced LSTM model selected for the IL layer demonstrated the lowest MSE and RMSE values for CPU prediction among all evaluated architectures, confirming that neural network-based forecasting is appropriate for the non-linear, non-stationary telemetry streams produced by edge nodes. The negative R^2 values observed for memory prediction during the cold-start phase reflect a genuine limitation of incremental learning under conditions where the target variable exhibits near-zero variance within short windows, and the decision to use MSE and RMSE as primary FedAsync activation criteria rather than R^2 represents a pragmatic and well-motivated response to this limitation.

The FedAsync mechanism demonstrated clear empirical value, consistently restoring prediction stability following model drift events that the local IL layer alone could not recover from. Its event-driven, asynchronous design ensures that communication overhead is incurred only when necessary, reducing synchronization overhead while preserving data privacy, as raw telemetry never leaves the local node. This privacy-preserving property is a non-trivial operational advantage in deployments spanning multiple administrative domains or subject to data residency regulations.

The PPO-based placement agent achieved 97% overall accuracy across 100 heterogeneous evaluation scenarios, outperforming supervised ML baselines by 33 to

37 percentage points and heuristic strategies by 58 to 64 percentage points. The latency–resource trade-off analysis confirmed that PPO achieves a 12 to 13% improvement over First-Fit and Best-Fit Bin Packing in the composite trade-off metric. Importantly, scalability-related properties of the design, including the lightweight nature of control-plane messaging, the asynchronous FL design reducing synchronization overhead, and the RL module being invoked only upon placement events, are architecturally motivated rather than empirically validated at larger scales. The experimental evaluation was conducted in a controlled KinD-based testbed limited to a 3-node configuration, and all quantitative results are empirically bounded to this scale. Workload generation was based on synthetic traffic patterns following a Gaussian distribution with up to 300 simulated requests, a range selected to evaluate decision-making quality under controlled dynamic workload conditions rather than to characterize raw infrastructure scalability limits. The centralized control-plane design of the RL decision module may additionally require adaptation for very large-scale deployments where a single decision agent becomes a bottleneck.

7.2 Synthesis of contributions

The research conducted in this thesis was organized around five research questions, defined in Chapter 1, each corresponding to a specific capability gap in the existing literature on AI-driven edge–cloud orchestration. The answers to these questions, as established by the empirical evidence presented in Chapters 5 and 6, are synthesized below.

RQ1 asked whether the dynamic service placement problem in heterogeneous edge environments can be effectively addressed through AI-driven orchestration mechanisms that outperform conventional optimization-based approaches under non-stationary workload conditions. The empirical evidence from both proposed approaches answers this question affirmatively. The service-centric approach

of Chapter 5 demonstrates that an AI-driven mechanism identifies optimal serverless function compositions that deterministic, intuition-based partitioning fails to discover, while the resource-centric approach of Chapter 6 demonstrates that a closed-loop, continuously adapting orchestration pipeline substantially outperforms both static classifiers and classical heuristics under dynamically evolving conditions, confirming that learning-based orchestration is more effective than conventional optimization for this problem class.

RQ2 asked whether an AI-driven approach operating at the service level, by exploiting the serverless FaaS paradigm, constitutes a resource-efficient and QoE-aware solution for dynamic service placement at the edge. The AI-based Function Regulator introduced in Chapter 5 answers this: by automatically identifying the function composition that minimizes end-to-end response time, it directly optimizes the primary determinant of user QoE while preserving the invocation-based resource efficiency of the serverless model. The empirical finding that response time is non-monotonically related to pipeline cardinality, and that it is effectively independent of the computational capacity of the hosting node across two distinct hardware configurations, establishes that service-level composition is a necessary and addressable dimension of the placement problem in its own right — one that no prior work in the serverless ML deployment literature had systematically addressed, and which reframes the placement problem from a two-dimensional question of *what* to deploy and *where* to a three-dimensional question that additionally asks *how* the service’s internal pipeline should be structured.

RQ3 asked, from the complementary perspective, whether an AI-driven approach that targets edge node resources directly constitutes a resource-efficient and QoE-aware solution for dynamic service placement. The three-layer orchestration approach of Chapter 6 answers this affirmatively. The IL layer, built on attention-enhanced LSTM models that achieved the lowest MSE and RMSE values among all evaluated architectures for CPU prediction, continuously forecasts node-level

resource dynamics and triggers proactive re-adaptation before saturation occurs, while the PPO-based decision layer produces placement actions that jointly minimize user-perceived latency and maintain balanced resource utilization, confirming that resource-centric AI-driven orchestration satisfies both resource efficiency and QoE objectives.

RQ4 asked whether, given the NP-hard nature of the placement problem, fully AI-driven orchestration mechanisms can produce more effective and proactive placement decisions than traditional heuristic-based models. The comparative evaluation of Chapter 6 answers this affirmatively and quantitatively: the PPO agent achieved 97% overall placement accuracy across 100 heterogeneous scenarios, outperforming supervised ML baselines by 33% to 37% and the First-Fit and Best-Fit Bin Packing heuristics by 58% to 64%, while the latency–resource trade-off analysis confirmed a 12 to 13% improvement over both heuristics in the composite trade-off metric. Crucially, the IL-based forecasting layer enables the orchestrator to anticipate resource exhaustion and act proactively, rather than reacting to constraint violations after they occur as greedy heuristics necessarily do.

RQ5 asked whether the proposed approaches adequately address scalability, user mobility, and security — the principal challenges of edge computing — while exploiting global observability for coordinated placement decisions. The proposed approaches address these challenges along three fronts. Scalability in the first proposed approach is supported by the invocation-based resource model of the serverless paradigm and by the event-driven, asynchronous activation of the FL and RL layers, which limit communication and control-plane overhead to the moments at which they are genuinely required. For the second proposed approach, a key enabler of scalability is the decentralized design of the first layer, in which each edge node hosts an autonomous IL agent that collects telemetry, forecasts its own resource dynamics, and adapts locally without depending on coordination with other nodes. Because these node-level agents operate independently

and in parallel, the forecasting workload grows linearly with the number of nodes rather than imposing a centralized bottleneck, allowing the architecture to accommodate edge ecosystems comprising hundreds of nodes without a corresponding increase in coordination overhead. The FL and RL layers are invoked only upon convergence degradation or placement events respectively, ensuring that the centralized components are engaged selectively rather than continuously as the node count grows. User mobility is incorporated directly into the RL state representation through the current hosting node and the closest node to the requesting user population, enabling mobility-aware migration decisions. Security and privacy are preserved by the FedAsync mechanism, which exchanges only model parameters rather than raw telemetry, respecting the bandwidth and data-residency constraints inherent to decentralized, multi-domain edge deployments. It should be noted, however, that the scalability properties of the design are architecturally motivated rather than empirically validated at large scale, as discussed in Section 7.1.2.

Combined, the two approaches constitute a unified response to the dynamic service placement problem that spans both its service composition and resource allocation dimensions. The two operate at different and complementary levels of granularity: the service-centric approach governs *how* the constituent tasks of an application are grouped into serverless functions, optimizing the internal structure of the service, whereas the resource-centric approach governs *where* the application is placed across the available edge nodes, optimizing its assignment to the underlying infrastructure.

7.3 Future research directions

The contributions of this thesis open several concrete directions for future research, organized around four themes: extending the service-centric approach, extending the resource-centric approach, integrating the two approaches into a unified

orchestration system, and extending the scope of the research to emerging challenges in next-generation edge–cloud environments.

7.3.1 Extensions of the service-centric approach

The AI-based Function Regulator was evaluated on a single ML/DL pipeline realized on Apache OpenWhisk. A natural and important extension is the generalization of the composition optimization approach to other pipeline types, including multi-branch DAG-structured pipelines whose execution topology is not strictly sequential, and to other FaaS platforms including OpenFaaS, AWS Lambda, and Knative [Knative Authors, 2024]. The non-monotonic relationship between pipeline cardinality and response time was established empirically for a specific fraud detection workload. Whether this property generalizes across application domains and pipeline structures is an open empirical question whose answer would substantially strengthen the theoretical foundations of the service-centric placement perspective.

A particularly promising direction is the extension of the Function Regulator to reason about the correlations between different pipeline partitions and their associated response time profiles in a holistic manner, moving beyond the current approach of independently evaluating each partitioning configuration. This would enable the regulator to explicitly model how the compute, I/O, and data intensity of each pipeline step, both individually and in combination, influences the optimal function grouping in end-to-end deployment scenarios. Future work should also investigate how rules, triggers, and connections to external data sources such as IoT devices affect pipeline composition decisions, broadening the scope of the pipelining approach beyond static, batch-oriented ML/DL workflows.

The cold-start penalty identified in Chapter 5 represents a significant practical barrier for latency-sensitive edge applications. Future work could investigate warm

container pool management strategies that pre-initialize a configurable number of runtime containers to eliminate cold-start latency for high-priority pipelines, and could integrate cold-start prediction models into the Function Regulator to account for initialization overhead when selecting the optimal pipeline composition.

7.3.2 Extensions of the resource-centric approach

The resource-centric approach was validated in a simulated environment with three worker nodes. Large-scale validation using production-grade edge infrastructure with a higher number of geographically distributed nodes and higher client request volumes represents the most immediate and important extension of this work. Scaling the approach to deployments with tens or hundreds of nodes introduces new challenges in federated aggregation convergence, RL state space dimensionality, and control-plane communication overhead, and future work should investigate hierarchical aggregation strategies for the FL layer that reduce communication overhead while preserving collaborative learning effectiveness at scale.

Replacing the current synthetic workload generation with real-world telemetry traces from production edge deployments is equally important, as realistic workload patterns may exhibit temporal dependencies and distributional properties that synthetic Gaussian distributions do not capture. Validating the approach under real-world conditions would provide a more rigorous empirical foundation for the scalability and adaptability claims made in Chapter 6.

The RL decision module follows a centralized control-plane design which may require adaptation for very large-scale deployments. Future work should investigate the extension of the placement layer towards multi-agent reinforcement learning (Multi-Agent Reinforcement Learning (MARL)) approaches, in which multiple cooperative agents share the placement responsibility across geographically dis-

tributed sub-clusters, reducing the state space dimensionality of each individual agent while preserving global coordination through inter-agent communication. The reward function design also motivates future exploration of multi-objective RL formulations that produce a Pareto front of placement policies rather than a single weighted scalarization, enabling operators to select policies according to their specific trade-off preferences at runtime.

Energy efficiency was not explicitly modeled as an optimization objective in the current approach. The growing operational and environmental cost of edge infrastructure makes energy-aware placement an increasingly important consideration, and future work should extend the reward function and state representation to incorporate node-level power consumption metrics alongside the existing CPU, memory, and network utilization signals, together with data locality and service dependency awareness as additional optimization dimensions.

7.3.3 Integration and joint optimization

The two approaches proposed in this thesis address the service composition and resource allocation dimensions of the placement problem independently. A natural and important direction for future work is their integration into a unified orchestration system that jointly optimizes pipeline composition and node assignment within a single closed-loop decision pipeline. Such a system would require a combined state representation that captures both the internal structure of the service pipeline and the resource state of the candidate hosting nodes, and a joint optimization objective that accounts for the interaction between inter-function data transfer overhead and inter-node communication latency. In the two approaches as presented in this thesis, the two decisions are taken sequentially and independently: the Function Regulator first determines the optimal grouping of an application's tasks into serverless functions, and the resulting application is subsequently placed, as a single unit, on an edge node selected by the orchestrator. A

fully integrated system would instead allow the two decisions to inform one another. Consider, for example, an ML/DL fraud-detection pipeline deployed across a multi-node edge cluster: when sufficient resources are available on a single node, co-locating all functions there minimizes inter-function data transfer and yields the composition identified by the Function Regulator. When no single node can accommodate the full pipeline, however, the functions must be distributed across multiple nodes, introducing inter-node communication latency that alters which function grouping is in fact optimal. A joint optimizer would weigh these two latency sources against each other, selecting the function composition and the node assignment together rather than in sequence. Realizing this coupling, in which function partitioning and node placement are co-optimized within a single closed-loop decision process, constitutes the central objective of this future research direction.

The integration of Large Language Model (LLM)-based reasoning into the orchestration pipeline represents a promising emerging direction. LLM agents could provide high-level semantic reasoning about application requirements, infrastructure constraints, and operator preferences, complementing the data-driven optimization of the IL, FL, and RL layers with a natural language interface for policy specification and constraint articulation. This direction aligns with the broader trend toward LLM-augmented AI systems and could substantially lower the barrier to deploying and configuring intelligent edge–cloud orchestration approaches in operational settings.

7.3.4 Broader research challenges

Several broader challenges in next-generation edge–cloud orchestration are directly motivated by the limitations and open questions identified in this thesis. The extension of the proposed approaches to multi-tenant edge environments, where multiple independent operators share the same physical infrastructure and

placement decisions made by one tenant affect the resource availability for others, introduces game-theoretic dimensions that neither approach currently addresses. The incorporation of user mobility prediction models, which could anticipate future user locations and trigger proactive migration before latency degradation occurs rather than in response to it, would extend the reactive mobility awareness of the current RL state representation toward a fully predictive placement paradigm.

The growing prevalence of Sixth Generation Mobile Network (6G) networks, with their promise of sub-millisecond latency, terabit-scale bandwidth, and native support for edge intelligence will fundamentally alter the constraints and opportunities of edge–cloud orchestration. The approaches proposed in this thesis were designed and evaluated under current network conditions, their adaptation to the 6G context, where the latency and bandwidth constraints that currently motivate edge-local processing may be relaxed for some application classes while remaining critical for others, represents a substantial and timely research agenda.

Finally, the deployment of AI-driven orchestration approaches in regulated industries such as healthcare, finance, and public safety introduces accountability, explainability, and auditability requirements that the current frameworks struggle to address. Future work should investigate the integration of explainable AI mechanisms into the placement decision pipeline, enabling operators and regulators to understand and audit the reasoning behind individual placement decisions in terms of observable system metrics and policy objectives.

Bibliography

- [Abbas et al., 2017] Abbas, N., Zhang, Y., Taherkordi, A., and Skeie, T. (2017). Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465.
- [Abd Alnabe and Zeebaree, 2024] Abd Alnabe, N. and Zeebaree, S. R. (2024). Distributed systems for real-time computing in cloud environment: A review of low-latency and time sensitive applications. *The Indonesian Journal of Computer Science*, 13(2).
- [Abdi and Williams, 2010] Abdi, H. and Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459.
- [Agarap, 2018] Agarap, A. F. (2018). Deep learning using rectified linear units (ReLU). *arXiv preprint arXiv:1803.08375*.
- [Ahmadvand and Foroutan, 2025] Ahmadvand, H. and Foroutan, F. (2025). Latency and privacy-aware resource allocation in vehicular edge computing. *arXiv preprint arXiv:2501.02804*.
- [Ajax et al., 2025] Ajax, R., Wasiu, R., and Daniel, S. (2025). Deep reinforcement learning for resource allocation in edge-cloud environments.
- [Akhtar et al., 2020] Akhtar, N., Raza, A., Ishakian, V., and Matta, I. (2020). COSE: Configuring serverless functions using statistical learning. In *Proceedings of the IEEE INFOCOM Conference on Computer Communications*, pages 129–138.
- [Al-Dulaimy et al., 2024] Al-Dulaimy, A., Jansen, M., Johansson, B., Trivedi, A., Iosup, A., Ashjaei, M., Galletta, A., Kimovski, D., Prodan, R., Tserpes, K., et al.

- (2024). The computing continuum: From iot to the cloud. *Internet of Things*, 27:101272.
- [Al-Zobaie et al., 2025] Al-Zobaie, D., Kazmi, A., and Clarke, S. (2025). Scalable mobility-aware dynamic service placement in edge computing. In *2025 IEEE 11th International Conference on Edge Computing and Scalable Cloud (Edge-Com)*, pages 52–61. IEEE.
- [Ali-Eldin et al., 2021] Ali-Eldin, A., Wang, B., and Shenoy, P. (2021). The hidden cost of the edge: A performance comparison of edge and cloud latencies. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*, St. Louis, MO, USA. ACM.
- [Alizadeh et al., 2020] Alizadeh, M. R., Khajehvand, V., Rahmani, A. M., and Akbari, E. (2020). Task scheduling approaches in fog computing: A systematic review. *International Journal of Communication Systems*, 33(16):e4583.
- [Altalhan et al., 2025] Altalhan, M., Algarni, A., and Alouane, M. T.-H. (2025). Imbalanced data problem in machine learning: A review. *IEEE access*, 13:13686–13699.
- [Alwarafy et al., 2020] Alwarafy, A., Al-Thelaya, K. A., Abdallah, M., Schneider, J., and Hamdi, M. (2020). A survey on security and privacy issues in edge-computing-assisted internet of things. *IEEE Internet of Things Journal*, 8(6):4004–4022.
- [Amazon Web Services, 2024a] Amazon Web Services (2024a). Amazon S3 — object storage built to retrieve any amount of data. <https://aws.amazon.com/s3/>.
- [Amazon Web Services, 2024b] Amazon Web Services (2024b). AWS Lambda — serverless compute service. <https://aws.amazon.com/lambda/>.
- [Amazon Web Services, 2026] Amazon Web Services (2026). Aws step functions. Accessed: 2026-05-15.

- [Apache Software Foundation, 2021] Apache Software Foundation (2021). Apache OpenWhisk — creating and invoking actions. <https://github.com/apache/openwhisk/blob/master/docs/actions-new.md>. Accessed: May 2026.
- [Apache Software Foundation, 2024a] Apache Software Foundation (2024a). Apache CouchDB: Open source database software. <https://couchdb.apache.org>. Accessed: May 2026.
- [Apache Software Foundation, 2024b] Apache Software Foundation (2024b). Apache OpenWhisk: An open source serverless cloud platform. <https://openwhisk.apache.org>. Accessed: May 2026.
- [Arnold et al., 2019] Arnold, E., Al-Jarrah, O. Y., Dianati, M., Fallah, S., Oxtoby, D., and Mouzakitis, A. (2019). A survey on 3d object detection methods for autonomous driving applications. *IEEE Transactions on Intelligent Transportation Systems*, 20(10):3782–3795.
- [Asgarian et al., 2024] Asgarian, M., Jamshidi, K., and Bohlooli, A. (2024). An efficient approximation algorithm for service function chaining placement in edge–cloud computing industrial internet of things. *IEEE Internet of Things Journal*, 11(7):12815–12822.
- [Bagher et al., 2021] Bagher, K., Khalil, I., Alabdulatif, A., and Atiquzzaman, M. (2021). Edgesom: Distributed hierarchical edge-driven iot data analytics framework. *Computer Communications*, 172:64–74.
- [Bahrami et al., 2023] Bahrami, B., Khayyambashi, M. R., and Mirjalili, S. (2023). Edge server placement problem in multi-access edge computing environment: models, techniques, and applications. *Cluster Computing*, 26(5):3237–3262.
- [Bahtovski and Gusev, 2014] Bahtovski, A. and Gusev, M. (2014). Cloudlet challenges. *Procedia Engineering*, 69:704–711.

- [Balakrishnan et al., 2025] Balakrishnan, K., Velusamy, D., Hinkle, H. E., Li, Z., Ramasamy, K., Khan, H., Ramaswamy, S., and Shah, P. M. (2025). Artificial intelligence in rural healthcare delivery: Bridging gaps and enhancing equity through innovation. *arXiv preprint arXiv:2508.11738*.
- [Baldini et al., 2017] Baldini, I., Cheng, P., Fink, S. J., Mitchell, N., Muthusamy, V., Rabbah, R., Suter, P., and Tardieu, O. (2017). The serverless trilemma: Function composition for serverless computing. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 89–103.
- [Barshan and Sharafat, 2017] Barshan, M. and Sharafat, A. R. (2017). Algorithms for application component placement in cloud computing environments. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7.
- [Batra and Clark, 2024] Batra, H. and Clark, R. (2024). EVCL: Elastic variational continual learning with weight consolidation. *arXiv preprint arXiv:2406.15972*.
- [Beckman et al., 2020] Beckman, P., Dongarra, J., Ferrier, N., Fox, G., Moore, T., Reed, D., and Beck, M. (2020). Harnessing the computing continuum for programming our world. In *Fog Computing: Theory and Practice*. Wiley.
- [Bhattacharjee et al., 2019a] Bhattacharjee, A., Barve, Y., Khare, S., Bao, S., Gokhale, A., and Damiano, T. (2019a). Stratum: A serverless framework for the lifecycle management of machine learning-based data analytics tasks. In *2019 USENIX conference on operational machine learning (OpML 19)*, pages 59–61.
- [Bhattacharjee et al., 2019b] Bhattacharjee, A., Chhokra, A. D., Kang, Z., Sun, H., Gokhale, A., and Karsai, G. (2019b). Barista: Efficient and scalable serverless serving system for deep learning prediction services. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pages 23–33. IEEE.
- [Bhuiyan et al., 2023] Bhuiyan, T., Kästner, L., Hu, Y., Kutschank, B., and Lambrecht, J. (2023). Deep-reinforcement-learning-based path planning for

- industrial robots using distance sensors as observation. *arXiv preprint arXiv:2301.05980*.
- [Bini, 2018] Bini, S. A. (2018). Artificial intelligence, machine learning, deep learning, and cognitive computing: what do these terms mean and how will they impact health care? *The Journal of arthroplasty*, 33(8):2358–2361.
- [Bittencourt et al., 2018] Bittencourt, L. F., Immich, R., Sakellariou, R., Fonseca, N. L. S., Madeira, E. R. M., Curado, M., Villas, L. A., da Silva, F., Rana, O., and Buyya, R. (2018). The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*, 3–4:134–155.
- [Boehm et al., 1995] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., and Selby, R. (1995). Cost models for future software life cycle processes: Como 2.0. *Annals of software engineering*, 1(1):57–94.
- [Boehm, 1984] Boehm, B. W. (1984). Software engineering economics. *IEEE transactions on Software Engineering*, (1):4–21.
- [Bouziane et al., 2010] Bouziane, H. L., Pérez, C., and Priol, T. (2010). Extending software component models with the master–worker paradigm. *Parallel Computing*, 36(2-3):86–103.
- [Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.
- [Burns et al., 2016] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. (2016). Borg, Omega, and Kubernetes. *Communications of the ACM*, 59(5):50–57.
- [Calcote and Butcher, 2019] Calcote, L. and Butcher, Z. (2019). *Istio: Up and Running*. O’Reilly Media.
- [Cao et al., 2020] Cao, K., Liu, Y., Meng, G., and Sun, Q. (2020). An overview on edge computing research. *IEEE Access*, 8:85714–85728.

- [Cappa et al., 2021] Cappa, F., Oriani, R., Peruffo, E., and McCarthy, I. (2021). Big data for creating and capturing value in the digitalized environment: unpacking the effects of volume, variety, and veracity on firm performance. *Journal of Product Innovation Management*, 38(1):49–67.
- [Carrión, 2022] Carrión, C. (2022). Kubernetes scheduling: Taxonomy, ongoing issues and challenges. *ACM Computing Surveys*, 55(7):1–37.
- [Caviglione et al., 2021] Caviglione, L., Gaggero, M., Paolucci, M., and Repetto, M. (2021). Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters. *Soft Computing*, 25(19):12569–12588.
- [Chataut et al., 2024] Chataut, R., Nankya, M., and Akl, R. (2024). 6g networks and the ai revolution—exploring technologies, applications, and emerging challenges. *Sensors*, 24(6).
- [Chawla et al., 2002] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.
- [Chen et al., 2021a] Chen, C.-L., Brinton, C. G., and Aggarwal, V. (2021a). Latency minimization for mobile edge computing networks. *IEEE Transactions on Mobile Computing*, 22(4):2233–2247.
- [Chen et al., 2016] Chen, J.-F., Chen, W.-L., Huang, C.-P., Huang, S.-H., and Chen, A.-P. (2016). Financial time-series data analysis using deep convolutional neural networks. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pages 87–92.
- [Chen et al., 2021b] Chen, Y., Lin, Y., Zheng, Z., Yu, P., Shen, J., and Guo, M. (2021b). Preference-aware edge server placement in the internet of things. *IEEE Internet of Things Journal*, 9(2):1289–1299.

- [Chen et al., 2020] Chen, Y., Ning, Y., Slawski, M., and Rangwala, H. (2020). Asynchronous online federated learning for edge devices with non-IID data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24.
- [Chen and Huang, 2017] Chen, Z. and Huang, X. (2017). End-to-end learning for lane keeping of self-driving cars. In *2017 IEEE intelligent vehicles symposium (IV)*, pages 1856–1860. IEEE.
- [Chiang et al., 2023] Chiang, Y., Zhang, Y., Luo, H., Chen, T.-Y., Chen, G.-H., Chen, H.-T., Wang, Y.-J., Wei, H.-Y., and Chou, C.-T. (2023). Management and orchestration of edge computing for iot: A comprehensive survey. *IEEE Internet of Things Journal*, 10(16):14307–14331.
- [Chima Ogbuachi et al., 2020] Chima Ogbuachi, M., Reale, A., Suskovics, P., and Kovács, B. (2020). Context-aware kubernetes scheduler for edge-native applications on 5g. *Journal of communications software and systems*, 16(1):85–94.
- [Christidis et al., 2019] Christidis, A., Davies, R., and Moschoyiannis, S. (2019). Serving machine learning workloads in resource constrained environments: A serverless deployment example. In *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*, pages 55–63. IEEE.
- [Cisco,] Cisco. What is edge computing – distributed architecture. <https://www.cisco.com/site/us/en/learn/topics/computing/what-is-edge-computing.html>. Accessed: 2026-04-20.
- [CNCF, 2024] CNCF (2024). Jaeger: Open source, end-to-end distributed tracing. <https://www.jaegertracing.io>. Accessed: May 2026.
- [Coffman et al., 2013] Coffman, E. G., Csirik, J., Galambos, G., Martello, S., Vigo, D., et al. (2013). Bin packing approximation algorithms: Survey and classification. In *Handbook of combinatorial optimization*, pages 455–531. Springer.

- [Coffman et al., 1999] Coffman, E. G., Galambos, G., Martello, S., and Vigo, D. (1999). Bin packing approximation algorithms: Combinatorial analysis. In *Handbook of Combinatorial Optimization: Supplement Volume A*, pages 151–207. Springer.
- [Decker et al., 2022] Decker, J., Kasprzak, P., and Kunkel, J. M. (2022). Performance evaluation of open-source serverless platforms for kubernetes. *Algorithms*, 15(7):234.
- [Donoho, 2017] Donoho, D. (2017). 50 years of data science. *Journal of computational and graphical statistics*, 26(4):745–766.
- [Douch et al., 2022] Douch, S., Abid, M. R., Zine-Dine, K., Bouzidi, D., and Benhaddou, D. (2022). Edge computing technology enablers: A systematic lecture study. *IEEE Access*, 10:69264–69302.
- [Dustdar et al., 2023] Dustdar, S., Casamayor Pujol, V., and Donta, P. K. (2023). On distributed computing continuum systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):4201–4215.
- [Eismann et al., 2021] Eismann, S., Bui, L., Grohmann, J., Abad, C., Herbst, N., and Kounev, S. (2021). Sizeless: Predicting the optimal size of serverless functions. In *Proceedings of the 22nd International Middleware Conference*, pages 248–259.
- [Eismann et al., 2020] Eismann, S., Scheuner, J., Van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., Abad, C. L., and Iosup, A. (2020). A review of serverless use cases and their characteristics. *arXiv preprint arXiv:2008.11110*.
- [Emmanuel et al., 2021] Emmanuel, T., Maupong, T., Mpoeleng, D., Semong, T., Mphago, B., and Tabona, O. (2021). A survey on missing data in machine learning. *Journal of Big data*, 8(1):140.
- [ETSI, 2024] ETSI (2024). ETSI GS MEC 003 V3.2.1: Multi-access Edge Computing (MEC); Framework and Reference Architecture. Group specification,

- European Telecommunications Standards Institute. https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/.
- [Fluent Bit Authors, 2024] Fluent Bit Authors (2024). Fluent bit — fast and lightweight telemetry agent. <https://fluentbit.io>. Accessed: May 2026.
- [Gao et al., 2020] Gao, M., Shen, R., Li, J., Yan, S., Li, Y., Shi, J., Han, Z., and Zhuo, L. (2020). Computation offloading with instantaneous load billing for mobile edge computing. *IEEE Transactions on Services Computing*, 15(3):1473–1485.
- [Gardner Jr, 1985] Gardner Jr, E. S. (1985). Exponential smoothing: The state of the art. *Journal of Forecasting*, 4(1):1–28.
- [Gepperth and Hammer, 2016] Gepperth, A. and Hammer, B. (2016). Incremental learning algorithms and applications. In *European Symposium on Artificial Neural Networks (ESANN)*.
- [Geurts et al., 2006] Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, 63(1):3–42.
- [Ghaseminya et al., 2025] Ghaseminya, M. M., Eslami, E., Shahzadeh Fazeli, S. A., Abouei, J., Abbasi, E., and Karbassi, S. M. (2025). Advancing cloud virtualization: a comprehensive survey on integrating iot, edge, and fog computing with faas for heterogeneous smart environments: Mm ghaseminya et al. *The Journal of Supercomputing*, 81(14):1303.
- [Goethals et al., 2020] Goethals, T., De Turck, F., and Volckaert, B. (2020). Extending kubernetes clusters to low-resource edge devices using virtual kubelets. *IEEE Transactions on Cloud Computing*.
- [Goethals et al., 2022] Goethals, T., De Turck, F., and Volckaert, B. (2022). Extending Kubernetes clusters to low-resource edge devices using virtual Kubelets. *IEEE Transactions on Cloud Computing*, 10(4):2623–2636.

- [Grigorescu et al., 2020] Grigorescu, S., Trasnea, B., Cocias, T., and Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of field robotics*, 37(3):362–386.
- [Han et al., 2021] Han, Y., Shen, S., Wang, X., Wang, S., and Leung, V. C. M. (2021). Tailored learning-based scheduling for Kubernetes-oriented edge–cloud system. In *IEEE INFOCOM 2021 — IEEE Conference on Computer Communications*, pages 1–10.
- [Hancock and Khoshgoftaar, 2020] Hancock, J. T. and Khoshgoftaar, T. M. (2020). Survey on categorical data for neural networks. *Journal of big data*, 7(1):28.
- [Hao et al., 2020] Hao, Y., Chen, M., Gharavi, H., Zhang, Y., and Hwang, K. (2020). Deep reinforcement learning for edge service placement in softwarized industrial cyber-physical system. *IEEE Transactions on Industrial Informatics*, 17(8):5552–5561.
- [He et al., 2018] He, T., Khamfroush, H., Wang, S., La Porta, T., and Stein, S. (2018). It’s hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 365–375. IEEE.
- [Hester et al., 2018] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. (2018). Deep Q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.
- [Hong and Varghese, 2019] Hong, C.-H. and Varghese, B. (2019). Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. *ACM computing surveys (csur)*, 52(5):1–37.
- [Hou et al., 2024] Hou, H., Agos Jawaddi, S. N., and Ismail, A. (2024). Energy efficient task scheduling based on deep reinforcement learning in cloud environment: A specialized review. *Future Generation Computer Systems*, 151:214–231.

- [Hou et al., 2021] Hou, J., Liu, H., Liu, Y., Wang, Y., Wan, P.-J., and Li, X.-Y. (2021). Model protection: Real-time privacy-preserving inference service for model privacy at the edge. *IEEE Transactions on Dependable and Secure Computing*, 19(6):4270–4284.
- [Hu et al., 2015] Hu, Y. C., Patel, M., Sabella, D., Sprecher, N., and Young, V. (2015). Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16.
- [Huang et al., 2020] Huang, T., Lin, W., Xiong, C., Pan, R., and Huang, J. (2020). An ant colony optimization-based multiobjective service replicas placement strategy for fog computing. *IEEE Transactions on Cybernetics*, 51(11):5595–5608.
- [Ibanez et al., 2019] Ibanez, S., Shahbaz, M., and McKeown, N. (2019). The case for a network fast path to the cpu. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pages 52–59.
- [IBM Cloud Documentation, 2022] IBM Cloud Documentation (2022). Apache OpenWhisk system details and limits. <https://github.com/ibm-cloud-docs/openwhisk/blob/master/limits.md>. Accessed: May 2026.
- [Jiang et al., 2021] Jiang, J., Gan, S., Liu, Y., Wang, F., Alonso, G., Klimovic, A., Singla, A., Wu, W., and Zhang, C. (2021). Towards demystifying serverless machine learning training. In *Proceedings of the 2021 international conference on management of data*, pages 857–871.
- [Jie et al., 2020] Jie, X., Liu, T., Gao, H., Cao, C., Wang, P., and Tong, W. (2020). A dqn-based approach for online service placement in mobile edge computing. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 169–183. Springer.
- [Jonas et al., 2019] Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N., et al. (2019).

- Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*.
- [Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.
- [Kaggle, 2019] Kaggle (2019). Ieee-cis fraud detection. Accessed: 2026-05-16.
- [Karlik and Olgac, 2011] Karlik, B. and Olgac, A. V. (2011). Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122.
- [Katsoulakis et al., 2024] Katsoulakis, E., Wang, Q., Wu, H., Shahriyari, L., Fletcher, R., Liu, J., Achenie, L., Liu, H., Jackson, P., Xiao, Y., et al. (2024). Digital twins for health: a scoping review. *NPJ digital medicine*, 7(1):77.
- [Kayal, 2020] Kayal, P. (2020). Kubernetes in fog computing: Feasibility, limitations and improvement scope. In *Proceedings of the IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–6. IEEE.
- [Kazmi et al., 2024] Kazmi, A. H., Staffolani, A., Zhang, T., Cabrera, C., and Clarke, S. (2024). Dynamic service placement in edge computing: a comparative evaluation of nature-inspired algorithms. *IEEE Access*, 13:2653–2670.
- [Kazmi et al., 2025] Kazmi, A. H., Staffolani, A., Zhang, T., Cabrera, C., and Clarke, S. (2025). Dynamic service placement in edge computing: A comparative evaluation of nature-inspired algorithms. *IEEE Access*, 13:2653–2670.
- [Kenner et al., 2021] Kenner, B., Chari, S. T., Kelsen, D., Klimstra, D. S., Pandol, S. J., Rosenthal, M., Rustgi, A. K., Taylor, J. A., Yala, A., Abul-Husn, N., et al. (2021). Artificial intelligence and early detection of pancreatic cancer: 2020 summative review. *Pancreas*, 50(3):251–279.

- [Keras Contributors, 2019] Keras Contributors (2019). Keras: The Python deep learning API. <https://keras.io>. Accessed: May 2026.
- [Keshavarz Haddadha et al., 2024] Keshavarz Haddadha, P., Rezvani, M. H., MollaMotalibi, M., and Shankar, A. (2024). Machine learning methods for service placement: a systematic review. *Artificial Intelligence Review*, 57(3):61.
- [Khan et al., 2019] Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., and Ahmed, A. (2019). Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235.
- [Kitaoka et al., 2025] Kitaoka, Y., Uchihashi, T., Kawata, S., Nishiura, A., Yamamoto, T., Hiraoka, S.-i., Yokota, Y., Isomura, E. T., Kogo, M., Tanaka, S., et al. (2025). Role and potential of artificial intelligence in biomarker discovery and development of treatment strategies for amyotrophic lateral sclerosis. *International Journal of Molecular Sciences*, 26(9):4346.
- [Knative Authors, 2024] Knative Authors (2024). Knative — serverless and event-driven applications on Kubernetes. <https://knative.dev/>. Accessed: May 2026.
- [Konda and Tsitsiklis, 1999] Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.
- [Kosaraju et al., 2023] Kosaraju, N., Sankepally, S. R., and Mallikharjuna Rao, K. (2023). Categorical data: Need, encoding, selection of encoding method and its emergence in machine learning models—a practical review study on heart disease prediction dataset using pearson correlation. In *Proceedings of International Conference on Data Science and Applications: ICDSA 2022, Volume 1*, pages 369–382. Springer.
- [Krislata et al., 2020] Krislata, I., Katrenko, A., Lytvyn, V., Vysotska, V., and Burov, Y. (2020). Traffic flows system development for smart city. In *ITPM*, pages 280–294.

- [Kritikos and Skrzypek, 2018] Kritikos, K. and Skrzypek, P. (2018). A review of serverless frameworks. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 161–168. IEEE.
- [Kubernetes,] Kubernetes. Production-grade container orchestration. <https://kubernetes.io>. Accessed: May 2026.
- [Kubernetes SIGs,] Kubernetes SIGs. kind (Kubernetes in Docker). <https://kind.sigs.k8s.io>. Accessed: May 2026.
- [KUMAR et al., 2025] KUMAR, D. L. S., PRASHANTHI, D., SURYANARAYANA, S., and ROSHINI, D. (2025). 5g and beyond advancing ultra low latency communication for the next generation of autonomous systems. *Journal of Theoretical and Applied Information Technology*, 103(24).
- [Kumar et al., 2010] Kumar, M., Husain, M., Upreti, N., and Gupta, D. (2010). Genetic algorithm: Review and application. *SSRN Electronic Journal*.
- [Kumar, 2025] Kumar, T. V. (2025). Scalable kubernetes workload orchestration for multi-cloud environments.
- [Kumari et al., 2026] Kumari, A., Behera, R. K., Sahoo, K. S., Sahoo, B., and Gandomi, A. H. (2026). Emergence of serverless computing in cloud: A state-of-the-art review of challenges and opportunities. *Machine Learning with Applications*, page 100862.
- [Li et al., 2020] Li, L., Fan, Y., Tse, M., and Lin, K.-Y. (2020). A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854.
- [Li, 2017] Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.
- [Li et al., 2022] Li, Y., Lin, Y., Wang, Y., Ye, K., and Xu, C. (2022). Serverless computing: state-of-the-art, challenges and opportunities. *IEEE Transactions on Services Computing*, 16(2):1522–1539.

- [Li et al., 2014] Li, Y., Tang, X., and Cai, W. (2014). On dynamic bin packing for resource allocation in the cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 2–11.
- [Lin et al., 2023] Lin, R., Guo, X., Luo, S., Xiao, Y., Moran, B., and Zukerman, M. (2023). Application-aware computation offloading in edge computing networks. *Future Generation Computer Systems*, 146:86–97.
- [Lindemann et al., 2021] Lindemann, B., Müller, T., Vietz, H., Jazdi, N., and Weyrich, M. (2021). A survey on long short-term memory networks for time series prediction. *Procedia CIRP*, 99:650–655.
- [Liu and Zhang, 2024] Liu, J. and Zhang, Q. (2024). Multi-objective optimization based service pre-deployment in edge computing. In *2024 IEEE 15th International Conference on Software Engineering and Service Science (ICSESS)*, pages 74–78. IEEE.
- [Liu et al., 2020a] Liu, L., Kuang, Z., Chen, Y., Xue, J.-H., Yang, W., and Zhang, W. (2020a). IncDet: In defense of elastic weight consolidation for incremental object detection. *IEEE Transactions on Neural Networks and Learning Systems*, 32(6):2306–2319.
- [Liu et al., 2019] Liu, L., Tan, H., Jiang, S. H.-C., Han, Z., Li, X.-Y., and Huang, H. (2019). Dependent task placement and scheduling with function configuration in edge computing. In *Proceedings of the International Symposium on Quality of Service*, pages 1–10.
- [Liu et al., 2018] Liu, X., Masana, M., Herranz, L., Van de Weijer, J., López, A. M., and Bagdanov, A. D. (2018). Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2262–2268.
- [Liu et al., 2020b] Liu, Y., He, Q., Zheng, D., Xia, X., Chen, F., and Zhang, B. (2020b). Data caching optimization in the edge computing environment. *IEEE transac-*

- tions on services computing*, 15(4):2074–2085.
- [Lu et al., 2018] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G. (2018). Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering*, 31(12):2346–2363.
- [Lu et al., 2019] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G. (2019). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363.
- [Lu et al., 2024] Lu, J., Yang, J., Li, S., Li, Y., Jiang, W., Dai, J., and Hu, J. (2024). A2C-DRL: Dynamic scheduling for stochastic edge–cloud environments using A2C and deep reinforcement learning. *IEEE Internet of Things Journal*, 11(9):16915–16927.
- [Luo et al., 2022] Luo, F., Zheng, S., Ding, W., Fuentes, J., and Li, Y. (2022). An edge server placement method based on reinforcement learning. *Entropy*, 24(3):317.
- [Mahjoubi et al., 2024] Mahjoubi, A., Ramaswamy, A., and Grinnemo, K.-J. (2024). An online simulated annealing-based task offloading strategy for a mobile edge architecture. *IEEE Access*, 12:70707–70718.
- [Mahmoudi et al., 2018] Mahmoudi, C., Mourlin, F., and Battou, A. (2018). Formal definition of edge computing: An emphasis on mobile cloud and IoT composition. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 34–42. IEEE.
- [Maia et al., 2020] Maia, A. M., Ghamri-Doudane, Y., Vieira, D., and de Castro, M. F. (2020). Dynamic service placement and load distribution in edge computing. In *Proceedings of the IFIP Networking Conference (IFIP Networking)*, pages 1–9. IFIP.
- [Malazi et al., 2022] Malazi, H. T., Chaudhry, S. R., Kazmi, A., Palade, A., Cabrera, C., White, G., and Clarke, S. (2022). Dynamic service placement in multi-access

- edge computing: A systematic literature review. *IEEE Access*, 10:32639–32688.
- [Mansouri and Babar, 2021] Mansouri, Y. and Babar, M. A. (2021). A review of edge computing: Features and resource virtualization. *Journal of Parallel and Distributed Computing*, 150:155–183.
- [Matetić et al., 2022] Matetić, I., Štajduhar, I., Wolf, I., and Ljubic, S. (2022). A review of data-driven approaches and techniques for fault detection and diagnosis in hvac systems. *Sensors*, 23(1):1.
- [McMahan et al., 2017] McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. Pmlr.
- [Mehdipour et al., 2019] Mehdipour, F., Javadi, B., Mahanti, A., and Ramirez-Prado, G. (2019). Fog computing realization for big data analytics. *Fog and edge computing: Principles and paradigms*, pages 259–290.
- [Menon et al., 2024] Menon, S., Addula, S. R., Parkavi, A., Subbalakshmi, C., Dhandayuthapani, V. B., Pokkuluri, K. S., and Soni, A. (2024). Streamlining task planning systems for improved enactment in contemporary computing surroundings. *SN Computer Science*, 5(8):993.
- [Meyes et al., 2017] Meyes, R., Tercan, H., Roggendorf, S., Thiele, T., Büscher, C., Obdenbusch, M., Brecher, C., Jeschke, S., and Meisen, T. (2017). Motion planning for industrial robots using reinforcement learning. *Procedia CIRP*, 63:107–112.
- [Miraz et al., 2015] Miraz, M. H., Ali, M., Excell, P. S., and Picking, R. (2015). A review on internet of things (iot), internet of everything (ioe) and internet of nano things (iont). *2015 Internet Technologies and Applications (ITA)*, pages 219–224.

- [Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- [Mohan et al., 2019] Mohan, S., Mullapudi, S., Sammeta, S., Vijayvergia, P., and Anastasiu, D. C. (2019). Stock price prediction using news sentiment analysis. In *2019 IEEE fifth international conference on big data computing service and applications (BigDataService)*, pages 205–208. IEEE Computer Society.
- [Morenets and Shabinskiy, 2020] Morenets, I. and Shabinskiy, A. (2020). Serverless event-driven applications development tools and techniques. *’*, 3:36–41.
- [Moreschini et al., 2022] Moreschini, S., Pecorelli, F., Li, X., Naz, S., Hästbacka, D., and Taibi, D. (2022). Cloud continuum: The definition. *IEEE Access*, 10:131876–131886.
- [Mouradian et al., 2019] Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., and Polakos, P. A. (2019). Application component placement in nfv-based hybrid cloud/fog systems with mobile fog nodes. In *IEEE Journal on Selected Areas in Communications*, volume 37, pages 1130–1143.
- [Murti et al., 2019] Murti, D. M. P., Pujiyanto, U., Wibawa, A. P., and Akbar, M. I. (2019). K-Nearest Neighbor (K-NN) based missing data imputation. In *2019 5th International Conference on Science in Information Technology (ICSITech)*, pages 83–88.
- [Naeem et al., 2020] Naeem, M., Rizvi, S. T. H., and Coronato, A. (2020). A gentle introduction to reinforcement learning and its application in different fields. *IEEE access*, 8:209320–209344.
- [Netdata Inc., 2024] Netdata Inc. (2024). Netdata — real-time infrastructure monitoring. <https://www.netdata.cloud>. Accessed: May 2026.

- [Newbold, 1983] Newbold, P. (1983). ARIMA model building and the time series analysis approach to forecasting. *Journal of Forecasting*, 2(1):23–35.
- [NGINX, Inc., 2024] NGINX, Inc. (2024). NGINX: High performance load balancer, web server, and reverse proxy. <https://nginx.org>. Accessed: May 2026.
- [Ning et al., 2020] Ning, Z., Dong, P., Wang, X., Wang, S., Hu, X., Guo, S., Qiu, T., Hu, B., and Kwok, R. Y. (2020). Distributed and dynamic service placement in pervasive edge computing networks. *IEEE Transactions on Parallel and Distributed Systems*, 32(6):1277–1292.
- [Nygren et al., 2010] Nygren, E., Sitaraman, R. K., and Sun, J. (2010). The Akamai network: A platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19.
- [Oh et al., 2020] Oh, C., Han, S., and Jeong, J. (2020). Time-series data augmentation based on interpolation. *Procedia Computer Science*, 175:64–71.
- [Open Container Initiative, 2024] Open Container Initiative (2024). OCI Runtime Specification, Image Format Specification, and Distribution Specification. <https://opencontainers.org>. Version 1.1.0/1.2.0. Accessed: 2025.
- [OpenFaaS Ltd., 2024] OpenFaaS Ltd. (2024). OpenFaaS: Serverless functions made simple. <https://github.com/openfaas/faas>. Accessed: May 2026.
- [OpenTelemetry Authors, 2024] OpenTelemetry Authors (2024). OpenTelemetry — high-quality, ubiquitous, and portable telemetry. <https://opentelemetry.io>. Accessed: May 2026.
- [OpenZipkin, 2024] OpenZipkin (2024). Zipkin — a distributed tracing system. <https://zipkin.io>. Accessed: May 2026.
- [Ozdemir, 2024] Ozdemir, Z. (2024). Chaining serverless functions for stateful workflows: AWS step functions using adapter pattern. <https://medium.com/@ozdemir.zynl/>

chaining-serverless-functions-for-stateful-workflows-aws-step-functions-using-adapt
Accessed: May 2026.

- [Pahl and Lee, 2015] Pahl, C. and Lee, B. (2015). Containers and clusters for edge cloud architectures – a technology review. In *Proceedings of the 3rd IEEE International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 379–386. IEEE.
- [Pan and McElhannon, 2018] Pan, J. and McElhannon, J. (2018). Future edge cloud and edge computing for internet of things applications. *IEEE Internet of Things Journal*, 5(1):439–449.
- [Paraskevoulakou and Kyriazis, 2021] Paraskevoulakou, E. and Kyriazis, D. (2021). Leveraging the serverless paradigm for realizing machine learning pipelines across the edge-cloud continuum. In *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 110–117. IEEE.
- [Paraskevoulakou and Kyriazis, 2023] Paraskevoulakou, E. and Kyriazis, D. (2023). ML-faas: Toward exploiting the serverless paradigm to facilitate machine learning functions as a service. *IEEE Transactions on Network and Service Management*, 20(3):2110–2123.
- [Paraskevoulakou et al., 2026] Paraskevoulakou, E., Symvoulidis, C., Karamolegkos, P., Kousiouris, G., and Kyriazis, D. (2026). A three-tier ai-based approach for dynamic application placement in cloud-edge environments. *Future Generation Computer Systems*, page 108511.
- [Paraskevoulakou et al., 2024] Paraskevoulakou, E., Tom-Ata, J.-D. T., Symvoulidis, C., and Kyriazis, D. (2024). Enhancing cloud-based application component placement with AI-driven operations. In *2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0687–0694.
- [Pearson, 2002] Pearson, R. K. (2002). Outliers in process modeling and identification. *IEEE Transactions on Control Systems Technology*, 10(1):55–63.

- [Peter, 2015] Peter, N. (2015). Fog computing and its real time applications. *Int. J. Emerg. Technol. Adv. Eng*, 5(6):266–269.
- [Petrucchi et al., 2025] Petrucci, L., Zanini, S., Palamà, I., Melazzi, N. B., and Bartoletti, S. (2025). Localization in 5g and beyond: A multi-objective approach for accuracy, latency, and resilience. *IEEE Transactions on Mobile Computing*.
- [Popescu et al., 2009] Popescu, M.-C., Balas, V. E., Perescu-Popescu, L., and Mastorakis, N. (2009). Multilayer perceptron and neural networks. *WSEAS transactions on circuits and systems*, 8(7):579–588.
- [Raban and Gordon, 2020] Raban, D. R. and Gordon, A. (2020). The evolution of data science and big data research: A bibliometric analysis. *Scientometrics*, 122(3):1563–1581.
- [Rahman et al., 2019] Rahman, M. O., Hossain, M. S., Junaid, T.-S., Forhad, M. S. A., and Hossen, M. K. (2019). Predicting prices of stock market using gated recurrent units (GRUs) neural networks. *International Journal of Computer Science and Network Security*, 19(1):213–222.
- [Raschka, 2018] Raschka, S. (2018). Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*.
- [Rausch et al., 2019] Rausch, T., Hummer, W., Muthusamy, V., Rashed, A., and Dustdar, S. (2019). Towards a serverless platform for edge {AI}. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*.
- [Rodríguez et al., 2021] Rodríguez, O. R. C., Pahl, C., El Ioini, N., Barzegar, H. R., et al. (2021). Improvement of edge computing workload placement using multi objective particle swarm optimization. In *2021 8th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 1–8. IEEE.

- [Rodríguez et al., 2018] Rodríguez, P., Bautista, M. A., Gonzalez, J., and Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31.
- [Rosendo et al., 2022] Rosendo, D., Costan, A., Valduriez, P., and Antoniu, G. (2022). Distributed intelligence on the edge-to-cloud continuum: A systematic literature review. *Journal of Parallel and Distributed Computing*, 166:71–94.
- [Rui et al., 2021] Rui, L., Qiu, X., Huang, H., Tang, Y., and Wu, G. (2021). Service migration method based on deep reinforcement learning in mobile edge computing. *IEEE Access*, 9:17356–17364.
- [Sadouk, 2019] Sadouk, L. (2019). CNN approaches for time series classification. In *Time Series Analysis — Data, Methods, and Applications*, volume 5. InTech.
- [Salem et al., 2024] Salem, A. H., Azzam, S. M., Emam, O. E., and Abohany, A. A. (2024). Advancing cybersecurity: a comprehensive review of ai-driven detection techniques. *Journal of Big Data*, 11(1):105.
- [Sampé et al., 2018] Sampé, J., Vernik, G., Sánchez-Artigas, M., and García-López, P. (2018). Serverless data analytics in the ibm cloud. In *Proceedings of the 19th international middleware conference industry*, pages 1–8.
- [Sari et al., 2020] Sari, A., Lekidis, A., and Butun, I. (2020). Industrial networks and iiot: Now and future trends. In *Industrial IoT: Challenges, Design Principles, Applications, and Security*, pages 3–55. Springer.
- [Sathiyamurthy and Naidu, 2025] Sathiyamurthy, A. and Naidu, S. G. S. (2025). Designing edge computing solutions for real-time vessel tracking and collision avoidance. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 16(2):45–62.
- [Satyanarayanan, 2017] Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1):30–39.

- [Satyanarayanan et al., 2009] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4):14–23.
- [Schulman et al., 2017a] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017a). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- [Schulman et al., 2017b] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- [Schuster and Paliwal, 1997] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.
- [scikit-learn Developers, 2019] scikit-learn Developers (2019). StratifiedKFold — scikit-learn documentation. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html. Accessed: May 2026.
- [Sewak and Singh, 2018] Sewak, M. and Singh, S. (2018). Winning in the era of serverless computing and function as a service. In *2018 3rd International Conference for Convergence in Technology (I2CT)*, pages 1–5. IEEE.
- [Sfafi et al., 2024] Sfafi, H., Lahyani, I., Yangui, S., and Torjmen, M. (2024). Latency-aware and proactive service placement for edge computing. *IEEE Transactions on Network and Service Management*, 21(4):4243–4254.
- [Shafiei et al., 2022] Shafiei, H., Khonsari, A., and Mousavi, P. (2022). Serverless computing: a survey of opportunities, challenges, and applications. *ACM Computing Surveys*, 54(11s):1–32.

- [Shah et al., 2018] Shah, S. D. A., Gregory, M. A., and Li, S. (2018). An overview of multi-access edge computing: Standards, opportunities, use cases and open challenges. *IEEE Access*, 6:68048–68068.
- [Shaukat et al., 2016] Shaukat, U., Ahmed, E., Anwar, Z., and Xia, F. (2016). Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges. *Journal of Network and Computer Applications*, 62:18–40.
- [Sheikh et al., 2025] Sheikh, A. M., Islam, M. R., Habaebi, M. H., Zabidi, S. A., Bin Najeeb, A. R., and Kabbani, A. (2025). A survey on edge computing (ec) security challenges: Classification, threats, and mitigation strategies. *Future Internet*, 17(4):175.
- [Shi et al., 2025] Shi, T., Zhang, T., Xu, X., Liu, Y., and Huang, R. (2025). Multi-modal mobile edge computing: Multi-objective optimization with synchronization constraint. *IEEE Transactions on Mobile Computing*.
- [Shi et al., 2016] Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646.
- [Siامي-Namini et al., 2019a] Siامي-Namini, S., Tavakoli, N., and Namin, A. S. (2019a). The performance of lstm and bilstm in forecasting time series. In *2019 IEEE International conference on big data (Big Data)*, pages 3285–3292. IEEE.
- [Siامي-Namini et al., 2019b] Siامي-Namini, S., Tavakoli, N., and Namin, A. S. (2019b). The performance of LSTM and BiLSTM in forecasting time series. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3285–3292.
- [Sigelman et al., 2010] Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspán, S., and Shanbhag, C. (2010). Dapper, a large-scale distributed systems tracing infrastructure. Technical Report Google Technical Report dapper-2010-1, Google. <https://research.google/pubs/dapper-a-large-scale-distributed-systems-tracing-infrastructure/>.

- [Sofia et al., 2026] Sofia, R. C., Salomon, J., Carrol, R., Garcés-Erice, L., Urbanetz, P., Gesswein, J., Touma, R., Espinosa, A., Contreras, L. M., Theodorou, V., et al. (2026). Towards scalable federated container orchestration: The codeco approach. *arXiv preprint arXiv:2601.13351*.
- [Sofia et al., 2024] Sofia, R. C., Salomon, J., Ferlin-Reiter, S., et al. (2024). A framework for cognitive, decentralized container orchestration. *IEEE Access*, 12:79978–80008.
- [Somesula et al., 2023] Somesula, M. K., Mothku, S. K., and Annadanam, S. C. (2023). Cooperative service placement and request routing in mobile edge networks for latency-sensitive applications. *IEEE Systems Journal*, 17(3):4050–4061.
- [Songhorabadi et al., 2023] Songhorabadi, M., Rahimi, M., MoghadamFarid, A., and Kashani, M. H. (2023). Fog computing approaches in iot-enabled smart cities. *Journal of Network and Computer Applications*, 211:103557.
- [Sridharan, 2018] Sridharan, C. (2018). *Distributed Systems Observability*. O'Reilly Media.
- [Stoica and Shenker, 2021] Stoica, I. and Shenker, S. (2021). From cloud computing to sky computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*, pages 26–32. ACM.
- [Supraja et al., 2025] Supraja, T., Chawla, P., and Gill, S. S. (2025). Ai-driven service placement in fog and edge computing environments: a systematic review, taxonomy and future directions. *Cluster Computing*, 28(16):1–39.
- [Svorobej et al., 2020] Svorobej, S., Bendeche, M., Griesinger, F., and Domaschka, J. (2020). Orchestration from the cloud to the edge. In *The Cloud-to-Thing Continuum: Opportunities and Challenges in Cloud, Fog and Edge Computing*, pages 61–77. Springer International Publishing, Cham.

- [Symvoulidis et al., 2024] Symvoulidis, C., Paraskevoulakou, E., Kiourtis, A., Mavrogiorgou, A., and Kyriazis, D. (2024). Dynamic resource allocation on the edge: A causal and contextually-aware machine learning approach. In *Intelligent Systems Conference*, pages 300–313. Springer.
- [Taleb et al., 2025] Taleb, I., Guillaume, J.-L., and Duthil, B. (2025). A survey on services placement algorithms in integrated cloud-fog/edge computing. *ACM Computing Surveys*, 57(11):1–36.
- [Taleb et al., 2017] Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., and Sabella, D. (2017). On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681.
- [Thorne and Vlachos, 2020] Thorne, J. and Vlachos, A. (2020). Elastic weight consolidation for better bias inoculation. *arXiv preprint arXiv:2004.14366*.
- [Tymplalexis et al., 2025] Tymplalexis, N., Kolomvatsos, K., and Anagnostopoulos, C. (2025). A demand aware services placement model in pervasive edge computing. *Computer Networks*, 264:111254.
- [Vacalopoulou et al., 2024] Vacalopoulou, A., Gardelli, V., Karafyllidis, T., Liwicki, F., Mokayed, H., Papaevripidou, M., Paraskevopoulos, G., Stamouli, S., Katsamanis, A., and Katsouros, V. (2024). Ai4edu: An innovative conversational ai assistant for teaching and learning. In *INTED2024 Proceedings*, pages 7119–7127. IATED.
- [Vahidinia et al., 2020] Vahidinia, P., Farahani, B., and Aliee, F. S. (2020). Cold start in serverless computing: Current trends and mitigation strategies. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–7. IEEE.
- [Van de Ven et al., 2024] Van de Ven, G. M., Soures, N., and Kudithipudi, D. (2024). Continual learning and catastrophic forgetting. *arXiv preprint*

- arXiv:2403.05175*.
- [Van Hasselt et al., 2016] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.
- [Vaquero and Roder-Merino, 2014] Vaquero, L. M. and Roder-Merino, L. (2014). Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM computer communication Review*, 44(5):27–32.
- [Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.
- [Verma et al., 2015] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. (2015). Large-scale cluster management at Google with Borg. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys)*, pages 1–17. ACM.
- [Villari et al., 2016] Villari, M., Fazio, M., Dustdar, S., Rana, O., and Ranjan, R. (2016). Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83.
- [Volz and Rabenstein, 2015] Volz, J. and Rabenstein, B. (2015). Prometheus: A next-generation monitoring system. In *Proceedings of the USENIX SREcon15 Europe*. USENIX Association.
- [Wang et al., 2019] Wang, H., Niu, D., and Li, B. (2019). Distributed machine learning with a serverless architecture. In *IEEE INFOCOM 2019-IEEE conference on computer communications*, pages 1288–1296. IEEE.
- [Wang et al., 2018] Wang, S., Xu, J., Zhang, N., and Liu, Y. (2018). A survey on service migration in mobile edge computing. *IEEE Access*, 6:23511–23528.

- [Weissberger, 2014] Weissberger, A. (2014). TiECon 2014 Summary — Part I: Qualcomm Keynote & IoT Track Overview. IEEE ComSoc Blog.
- [Wen et al., 2021] Wen, J., Chen, Z., Liu, Y., Lou, Y., Ma, Y., Huang, G., Jin, X., and Liu, X. (2021). An empirical study on challenges of application development in serverless computing. In *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pages 416–428.
- [Widodo et al., 2022] Widodo, S., Brawijaya, H., and Samudi, S. (2022). Stratified k-fold cross validation optimization on machine learning for prediction. *Sinkron: jurnal dan penelitian teknik informatika*, 6(4):2407–2414.
- [Wolff, 2016] Wolff, E. (2016). *Microservices: flexible software architecture*. Addison-Wesley Professional.
- [Xiao et al., 2025] Xiao, W., Shi, C., Chen, M., Liu, Z., Chen, M., and Song, H. H. (2025). GraphEdge: Dynamic graph partition and task scheduling for GNNs computing in edge network. *Information Fusion*, page 103329.
- [Xiong et al., 2018] Xiong, Y., Sun, Y., Xing, L., and Huang, Y. (2018). Extend cloud to edge with KubeEdge. In *Proceedings of the IEEE/ACM Symposium on Edge Computing (SEC)*, pages 373–377. IEEE.
- [Xu et al., 2023] Xu, C., Qu, Y., Xiang, Y., and Gao, L. (2023). Asynchronous federated learning on heterogeneous devices: A survey. *Computer Science Review*, 50:100595.
- [Xu et al., 2018] Xu, J., Chen, L., and Zhou, P. (2018). Joint service caching and task offloading for mobile edge computing in dense networks. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 207–215. IEEE.
- [Yan and Zhang, 2021] Yan, J. and Zhang, M. (2021). A transfer-learning based energy consumption modeling method for industrial robots. *Journal of Cleaner*

- Production*, 325:129299.
- [Yi et al., 2015] Yi, S., Li, C., and Li, Q. (2015). A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data*, pages 37–42.
- [Yu et al., 2020] Yu, S., Chen, X., Yang, L., Wu, D., Bennis, M., and Zhang, J. (2020). Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading. *IEEE Wireless Communications*, 27(1):92–99.
- [Yu et al., 2019] Yu, Y., Si, X., Hu, C., and Zhang, J. (2019). A review of recurrent neural networks: LSTM cells and network architectures. *Neural Computation*, 31(7):1235–1270.
- [Zarei et al., 2025] Zarei, S., Azizi, S., and Ahmed, A. (2025). Optimizing edge server placement and load distribution in mobile edge computing using aco and heuristic algorithms. *The Journal of Supercomputing*, 81(1):257.
- [Zhang et al., 2021] Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., and Gao, Y. (2021). A survey on federated learning. *Knowledge-Based Systems*, 216:106775.
- [Zhang et al., 2020] Zhang, M., Krintz, C., and Wolski, R. (2020). Stoic: Serverless teleoperable hybrid cloud for machine learning applications on edge device. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 1–6. IEEE.
- [Zhao and Tang, 2020] Zhao, H. and Tang, J. (2020). A data streaming process framework for autonomous driving by edge. *arXiv preprint arXiv:2006.05817*.
- [Zheng and Casari, 2018] Zheng, A. and Casari, A. (2018). *Feature engineering for machine learning: principles and techniques for data scientists*. ” O’Reilly Media, Inc.”.
- [Zheng et al., 2024] Zheng, R., Xu, J., Wang, X., Liu, M., and Zhu, J. (2024). Service placement strategies in mobile edge computing based on an improved genetic

algorithm. *Pervasive and Mobile Computing*, 105:101986.

[Zhou et al., 2023] Zhou, B., Lu, B., and Zhang, Z. (2023). Placement of edge servers in mobile cloud computing using artificial bee colony algorithm. *International Journal of Advanced Computer Science and Applications*, 14(2).

[Zhu et al., 2023] Zhu, G. et al. (2023). Machine learning for large-scale optimization in 6G wireless networks. *IEEE Communications Surveys & Tutorials*.

[Zimmermann and Koziolok, 2020] Zimmermann, M. and Koziolok, A. (2020). Data-flow-aware application component placement in cloud computing environments. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 91–100.

