Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων Πληροφορικης

Εργαστηριο Δικτυων Επικοινωνιας Υψηλης Ταχυτητας

Deep Reinforcement Learning Mechanisms for Efficient Dynamic Resource Management in Cloud-Native Applications

Diploma Thesis

by

Georgios Baris

Supervisor: EMMANOUEL VARVARIGOS
Professor NTUA

Athens, June 2025

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής
Εργαστήριο Δικτύων Επικοινωνίας Υψηλής Ταχύτητας

Deep Reinforcement Learning Mechanisms for Efficient Dynamic Resource Management in Cloud-Native Applications

Diploma Thesis

by

Georgios Baris

Supervisor: EMMANOUEL VARVARIGOS
Professor E.M.Π.

Approved by the three-member examination committee on 27η June, 2025.

.....	.....	.....
EMMANOUEL VARVARIGOS	THEODORA VARVARIGOU	HERCULES AVRAMOPOULOS
Professor NTUA	Professor NTUA	Professor NTUA

Athens, June 2025

.....

Γεωργιος Μπαρης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Οι υποδομές που λειτουργούν σε περιβάλλοντα υπολογιστικού νέφους, που χαρακτηρίζονται από τον ετερογενή φόρτο εργασίας τους, θέτουν σημαντικές προκλήσεις στην αποτελεσματική διαχείριση των υπολογιστικών πόρων σε διαφορετικά περιβάλλοντα εκτέλεσης. Σε αυτή τη διπλωματική, προτείνουμε ένα πλαίσιο δύο σταδίων που συνδυάζει την πρόβλεψη φόρτου εργασίας με χρήση βαθιάς μάθησης με την βαθιά ενισχυτική μάθηση (DRL) για την έξυπνη κατανομή πόρων σε ιεραρχικά συστήματα edge-cloud.

Στην πρώτη φάση, χρησιμοποιούνται μοντέλα νευρωνικών δικτύων —συγκεκριμένα αρχιτεκτονικές Μακράς Βραχύχρονης Μνήμη (LSTM) και Transformer—τα οποία εκπαιδεύονται στο σύνολο δεδομένων Alibaba Cloud Trace για την πρόβλεψη μελλοντικού φόρτου εργασιών. Πειραματικές συγκρίσεις βάσει μετρικών παλινδρόμησης (RMSE, MAE, και R^2) δείχνουν ότι το μοντέλο Transformer υπερτερεί σταθερά του LSTM ως προς την ακρίβεια και τη χρονική συνέπεια. Με ακρίβεια πρόβλεψης $R^2 = 0.850$ και ελάχιστη διακύμανση μεταξύ διαφορετικών συνθηκών δοκιμής, το Transformer αναδεικνύεται ως ισχυρό εργαλείο. Το δεύτερο στάδιο διατυπώνει το πρόβλημα κατανομής πόρων ως διαδικασία απόφασης Markov (MDP) και εφαρμόζει Deep Q-Network (DQN) για να μάθει βελτιστοποιημένες πολιτικές κατανομής πόρων. Το προσαρμοσμένο περιβάλλον προσομοίωσης, που έχει δημιουργηθεί χρησιμοποιώντας το Gymnasium και το Ray RLlib, αποτυπώνει την πολυπλοκότητα της πραγματικής υποδομής με συνδυασμό near edge, far edge και cloud υποδομών, το καθένα με ξεχωριστά χαρακτηριστικά καθυστέρησης, κόστους και ενέργειας. Το περιβάλλον λαμβάνει υπόψη την προτεραιότητα με βάση την ανοχή στην καθυστέρηση που έχει η κάθε εφαρμογή, τον προγραμματισμό με επίγνωση των προτεραιοτήτων και τα δυναμικά μοτίβα άφιξης εργασίας. Για να διασφαλιστεί η επεκτασιμότητα και ο ρεαλισμός, το μοντέλο επεκτείνεται περαιτέρω σε ένα περιβάλλον πολλαπλών πρακτόρων, όπου ανεξάρτητες πύλες δικτύου αλληλεπιδρούν μέσω κοινής υποδομής. Η αξιολόγηση του συστήματος, τόσο με πραγματικά όσο και με συνθετικά δεδομένα παραγόμενα από Transformer, καταδεικνύει υψηλά ποσοστά επιτυχούς κατανομής, αυξημένη χρήση των πόρων στα επίπεδα edge, χαμηλό μέσο κόστος ανά εργασία και σημαντική ενεργειακή αποδοτικότητα. Τα συνθετικά φορτία του Transformer παρουσιάζουν ικανοποιητική ποιότητα, με βελτίωση της σταθερότητας της πολιτικής κατά 20–25% και πιο ρεαλιστικά χαρακτηριστικά φόρτου σε σχέση με άλλες μεθόδους παραγωγής. Δείκτες όπως ο λόγος ανταμοιβής προς κόστος και η ανταμοιβή ανά κιλοβατώρα επιβεβαιώνουν την ευφυή και οικονομικά αποδοτική λειτουργία της προσέγγισης DRL.

Συνολικά, το προτεινόμενο σύστημα παρουσιάζει τη δυνατότητα συνδυασμού προγνωστικών μοντέλων και DRL για διαχείριση πόρων σε πραγματικό χρόνο σε υπολογιστές εγγενούς νέφους. Τα αποτελέσματα αυτής της εργασίας υποστηρίζουν την ανάπτυξη έξυπνων μηχανισμών που μπορούν να ικανοποιήσουν τις απαιτήσεις καθυστέρησης, κόστους και ενέργειας των εφαρμογών.

Λέξεις-κλειδιά — Βαθιάς Ενισχυτική Μάθηση, κατανομή πόρων, Υπολογιστικό Νέφος, Δίκτυα Μακράς Βραχύχρονης Μνήμης(LSTM), Transformer,Συστήματα Πολλαπλών Πρακτόρων, Βαθία Ενισχυτική Μάθηση, Πρόβλεψη φόρτου εργασίας, Alibaba Cloud Trace

Abstract

Effectively managing computational resources in modern cloud-native infrastructures is a challenging task due to their elastic scalability and the heterogeneous nature of their workloads. This thesis introduces a two-phase framework for intelligent resource allocation in hierarchical edge-cloud systems, integrating deep reinforcement learning (DRL) with predictive modeling using deep learning techniques.

In the first phase, neural sequence models—specifically Long Short-Term Memory (LSTM) and Transformer architectures—are trained on the Alibaba Cloud Trace dataset to forecast workload telemetry, such as CPU utilization, in containerized batch workloads. Experimental comparisons across standard regression metrics (RMSE, MAE, and R^2) consistently show that the Transformer model outperforms LSTM in both accuracy and temporal consistency. With a forecasting accuracy of $R^2 = 0.850$ and minimal variance across test scenarios, the Transformer demonstrates strong capabilities in capturing complex temporal patterns, making it highly suitable for proactive autoscaling strategies.

The second phase formulates the resource allocation task as a Markov Decision Process (MDP) and utilizes a Deep Q-Network (DQN) agent to learn optimal job placement policies. A custom simulation environment—developed using Gymnasium and Ray RLlib—models real-world infrastructure with near-edge, far-edge, and cloud clusters, each characterized by unique latency, cost, and energy profiles. The environment incorporates dynamic job arrivals, priority-aware scheduling, and latency-sensitive reward shaping. To ensure scalability and reflect distributed system conditions, the framework is extended into a multi-agent system where independent gateways operate concurrently over shared infrastructure.

Comprehensive evaluation of the system under both real and Transformer-generated synthetic workloads demonstrates high allocation success rates, efficient edge resource utilization, low average job cost, and strong energy performance. Notably, Transformer-generated workloads yield enhanced policy stability (20–25% improvement) and more realistic workload characteristics compared to other synthetic generation methods. Performance indicators such as reward-to-cost ratio and reward-per-kWh validate the economic and operational advantages of the DRL-based approach under synthetic testing.

Overall, the proposed framework showcases the potential of integrating advanced Transformer-based workload forecasting with DRL for real-time, adaptive resource management in cloud-native environments. The results support the development of intelligent autoscaling mechanisms capable of satisfying latency, cost, and energy requirements in emerging edge-cloud applications through accurate prediction and stable policy execution.

Key-words — Deep Reinforcement Learning, Resource Allocation, Cloud Computing, Edge Computing, LSTM, Transformer, Multi-Agent Systems, DQN, Workload Forecasting, Alibaba Cloud Trace

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, κ. Εμμανουήλ Βαρβαρίγο, για την ευκαιρία που μου έδωσε και την εμπιστοσύνη που μου έδειξε στο να εκπονήσω τη διπλωματική μου εργασία στο Εργαστήριο Δικτύων Επικοινωνίας Υψηλής Ταχύτητας, καθώς και για την πολύτιμη καθοδήγηση που μου παρείχε, κατά τη διάρκεια αυτής της διπλωματικής, αλλά και πέραν αυτής.Θα ήθελα επιπλέον να ευχαριστήσω τον μεταδιδακτορικό ερευνητή Πολυζώη Σούμπλη για τη στενή συνεργασία μας, την ανεκτίμητη βοήθεια και τη συνεχή υποστήριξή του, δίχως των οποίων η εκπόνηση αυτής της διπλωματικής δεν θα οδηγούσε σε αυτό το αποτέλεσμα.

Τέλος, θέλω να ευχαριστήσω τους γονείς μου Χρήστο και Δήμητρα καθώς και τα αδέλφια μου, χωρίς αυτούς δεν θα μπορούσα να είχα καταφέρει όσα έχω πετύχει, όπως επίσης και τους φίλους μου, με τους οποίους περάσαμε αμέτρητες ώρες μελέτης, συμπαράστασης, διασκέδασης και ταξιδιών, στιγμές που θα μείνουν ανεξίτηλες στη μνήμη μου.

Γεώργιος Μπαρής, Ιούνιος 2025

Contents

List of Figures

Chapter 1

Εκτεταμένη Περίληψη στα Ελληνικά

1.1 Θεωρητικό Υπόβαθρο

Η ραγδαία ανάπτυξη cloud-native εφαρμογών και η αυξανόμενη πολυπλοκότητα των υπολογιστικών συστημάτων δημιουργούν νέες προκλήσεις στη διαχείριση πόρων [6]. Οι αρχιτεκτονικές edge–cloud επιτρέπουν την εκτέλεση εργασιών σε διαφορετικά επίπεδα, από κοντινούς κόμβους (near-edge) έως απομακρυσμένα data centers (cloud), απαιτώντας όμως έξυπνη πρόβλεψη και λήψη αποφάσεων σε πραγματικό χρόνο.

1.1.1 Αρχιτεκτονική Edge–Cloud

Το υπολογιστικό νέφος, σύμφωνα με τον ορισμό του NIST [48], προσφέρει ευέλικτη πρόσβαση σε παραμετροποιήσιμους υπολογιστικούς πόρους μέσω διαδικτύου. Παρά τις ισχυρές δυνατότητές του, η συγκεντρωτική φύση του εισάγει καθυστερήσεις και συμφόρηση, ιδιαίτερα σε εφαρμογές με αυστηρές απαιτήσεις απόκρισης [77]. Η αρχιτεκτονική edge computing προτείνει τη μεταφορά της επεξεργασίας πλησιέστερα στην πηγή των δεδομένων, μειώνοντας τη καθυστέρηση και αυξάνοντας την αποδοτικότητα. Ο συνδυασμός edge και cloud σε υβριδικά μοντέλα οδηγεί σε μια πολυεπίπεδη αρχιτεκτονική [14], η οποία περιλαμβάνει τις τερματικές συσκευές, τους edge κόμβους και το cloud. Οι τερματικές συσκευές, όπως κινητά τηλέφωνα, αισθητήρες και IoT μονάδες, επικεντρώνονται κυρίως στη συλλογή δεδομένων. Οι edge κόμβοι, όπως τοπικοί servers, routers και access points, επιτελούν άμεση επεξεργασία και προσωρινή αποθήκευση των δεδομένων κοντά στο σημείο παραγωγής τους. Το ανώτερο επίπεδο, δηλαδή το cloud, φιλοξενεί κεντρικά data centers με υψηλή υπολογιστική ισχύ και χρησιμοποιείται για σύνθετη ανάλυση, εκπαίδευση μοντέλων και μακροχρόνια αποθήκευση. Η εγγύτητα των edge κόμβων στον χρήστη επιτρέπει την υποστήριξη λειτουργιών σε πραγματικό χρόνο, ενώ το cloud συμπληρώνει την υποδομή παρέχοντας δυνατότητες μεγάλης κλίμακας. Η ολοκληρωμένη αξιοποίηση όλων των επιπέδων ενισχύει την απόδοση, προστατεύει την ιδιωτικότητα και καθιστά την αρχιτεκτονική edge–cloud κατάλληλη για την υλοποίηση έξυπνων, αποκεντρωμένων συστημάτων και την επέκταση του οικοσυστήματος του Internet of Things (IoT).

1.1.2 Κατανομή Πόρων (Resource Allocation)

Η κατανομή πόρων αφορά τη δυναμική και αποδοτική διανομή υπολογιστικών πόρων, όπως επεξεργαστική ισχύ, μνήμη και αποθηκευτικού χώρου, μεταξύ ανταγωνιστικών εφαρμογών, υπηρεσιών και χρηστών, με στόχο τη βελτιστοποίηση της απόδοσης και την τήρηση των απαιτήσεων ποιότητας υπηρεσίας (QoS) [3]. Σε σύγχρονα υπολογιστικά περιβάλλοντα, όπου οι απαιτήσεις μεταβάλλονται συνεχώς, απαιτούνται ευέλικτες και προσαρμοστικές στρατηγικές διαχείρισης. Η πολυπλοκότητα αυξάνεται ιδιαίτερα σε κατανεμημένα και ετερογενή συστήματα, όπου οι κόμβοι διαφέρουν ως προς τις δυνατότητες, την ενεργειακή διαθεσιμότητα και την τοπολογική εγγύτητα προς τις πηγές δεδομένων. Η αυξημένη παραγωγή δεδομένων από αισθητήρες και IoT συσκευές απαιτεί προσεκτική διαχείριση των υπολογιστικών φορτίων, της δικτυακής κυκλοφορίας και της ενεργειακής κατανάλωσης. Η αναποτελεσματική κατανομή μπορεί να οδηγήσει σε συμφόρηση, καθυστέρηση και σπατάλη πόρων. Σε υποδομές edge και cloud, η ανάγκη για έξυπνη εκχώρηση εργασιών ανάλογα με τη χρονική κρισιμότητα και τους ενεργειακούς περιορισμούς καθίσταται κρίσιμη. Η ενσωμάτωση ευέλικτων μηχανισμών επιτρέπει την άμεση ανταπόκριση σε μεταβολές φορτίου, διασφαλίζοντας αξιοπιστία, απόδοση και οικονομία κλίμακας. Η εξάπλωση τεχνολογιών όπως το cloud computing, το edge computing και το IoT εντείνει τις απαιτήσεις για επεκτάσιμες και ευέλικτες στρατηγικές κατανομής. Οι cloud-native εφαρμογές λειτουργούν σε δυναμικά περιβάλλοντα με απρόβλεπτα φορτία, ενώ οι υποδομές edge περιορίζονται από χαμηλούς πόρους, καθιστώντας απαραίτητη την ισορροπημένη κατανομή μεταξύ cloud, edge και τερματικών συσκευών. Η αποδοτική κατανομή πόρων συμβάλλει σε μείωση της καθυστέρησης, ενεργειακή αποδοτικότητα, αποφυγή συμφόρησης και μείωση κόστους. Το πρόβλημα βέλτιστης κατανομής υπό περιορισμούς θεωρείται υπολογιστικά δύσκολο (NP-complete) και παρουσιάζει ομοιότητες με το κλασικό πρόβλημα του σακιδίου (knapsack problem) [41], απαιτώντας την αξιοποίηση ευφυών και προσεγγιστικών μεθόδων για αποτελεσματική επίλυση.

1.1.3 Δίκτυα Μακράς-Βραχύχρονης Μνήμης (LSTM)

Τα δίκτυα LSTM (Long Short-Term Memory) αποτελούν μια εξειδικευμένη μορφή αναδρομικών νευρωνικών δικτύων (RNN), σχεδιασμένη για να υπερνικά τα προβλήματα που αντιμετωπίζουν τα συμβατικά RNNs κατά τη μάθηση μακροπρόθεσμων εξαρτήσεων σε ακολουθιακά δεδομένα. Τα κλασικά RNN συχνά υποφέρουν από το φαινόμενο της εξαφάνισης ή της έκρηξης των βαθμίδων (vanishing/exploding gradients), γεγονός που δυσχεραίνει τη σύνδεση γεγονότων που εμφανίζονται σε απομακρυσμένες χρονικές στιγμές. Οι Hochreiter και Schmid-

huber εισήγαγαν το 1997 τη δομή LSTM [31], εισάγοντας ένα σύστημα εσωτερικής μνήμης που μπορεί να διατηρεί πληροφορία σε βάθος χρόνου. Η βασική δομή ενός LSTM περιλαμβάνει μια κυψελίδα κατάστασης (cell state), η οποία λειτουργεί ως εσωτερική μνήμη και διατηρεί κρίσιμες πληροφορίες κατά μήκος της χρονικής ακολουθίας. Η διαχείριση αυτής της μνήμης επιτυγχάνεται μέσω τριών πυλών: της πύλης εισόδου, της πύλης λήθης και της πύλης εξόδου. Κάθε πύλη βασίζεται σε ένα επίπεδο ενεργοποίησης sigmoid, το οποίο ελέγχει ποιο ποσοστό πληροφορίας θα προστεθεί, διατηρηθεί ή απορριφθεί. Η πύλη εισόδου ρυθμίζει τη ροή νέων πληροφοριών, η πύλη λήθης αποφασίζει τι θα διαγραφεί από την προηγούμενη μνήμη, και η πύλη εξόδου καθορίζει την τελική κρυφή κατάσταση που μεταδίδεται στο επόμενο χρονικό βήμα.

Η εσωτερική αρχιτεκτονική ενός LSTM παρουσιάζεται στο Σχήμα 3.4.2, όπου αποτυπώνεται η ροή πληροφορίας μεταξύ των πυλών και της μνήμης. Η μαθηματική περιγραφή της λειτουργίας του LSTM είναι η εξής:

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\
\tilde{c}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
$$

όπου σ η συνάρτηση sigmoid, tanh η υπερβολική εφαπτομένη, και $\odot$ η στοιχειομετρική πολλαπλασιαστική πράξη.

Η αρχιτεκτονική των LSTM προσφέρει σημαντικά πλεονεκτήματα σε εφαρμογές όπου απαιτείται εκμάθηση χρονικών εξαρτήσεων, όπως στην αναγνώριση φωνής, την επεξεργασία φυσικής γλώσσας, τη μετάφραση και την πρόβλεψη χρονοσειρών. Στο πλαίσιο της υπολογιστικής νέφους, τα LSTM χρησιμοποιούνται για την πρόβλεψη χρήσης πόρων, επιτρέποντας πιο αποδοτική και προληπτική κατανομή σε δυναμικά περιβάλλοντα.

1.1.4 Μετασχηματιστές (Transformers)

Οι μετασχηματιστές (Transformers)[45] αποτελούν μια κατηγορία μοντέλων βαθιάς μάθησης που άλλαξε ριζικά τον τρόπο επεξεργασίας ακολουθιακών δεδομένων, εισάγοντας τον μηχανισμό αυτοπροσοχής (self-attention). Σε αντίθεση με τα RNN και τα CNN, οι μετασχηματιστές δεν βασίζονται σε διαδοχική επεξεργασία και επιτρέπουν πλήρη παραλληλία κατά την εκπαίδευση. Αυτό ενισχύει σημαντικά την υπολογιστική αποδοτικότητα και επιτρέπει την επεξεργασία μακροπρόθεσμων εξαρτήσεων χωρίς προβλήματα βαθμίδων. Η αρχιτεκτονική του Transformer βασίζεται σε μια δομή κωδικοποιητή–αποκωδικοποιητή, όπως απεικονίζεται στο Σχήμα 3.4.3. Κάθε επίπεδο περιλαμβάνει μηχανισμούς πολυκέφαλης αυτοπροσοχής (multi-head self-attention), προωθητικά νευρωνικά δίκτυα (feed-forward layers), καθώς και συνδέσεις υπολοίπου (residual connections) και κανονικοποίηση επιπέδου (layer normalization). Οι λέξεις/είσοδοι μετατρέπονται σε τριπλέτες από διανύσματα ερωτήματος (query), κλειδιού (key) και τιμής (value), τα οποία επεξεργάζονται μέσω του μηχανισμού scaled dot-product attention. Οι μετασχηματιστές προσφέρουν πλεονεκτήματα όπως ευελιξία εισόδου–εξόδου, καλύτερη γενίκευση σε μακροπρόθεσμες ακολουθίες και δυνατότητα προεκπαίδευσης σε τεράστια σύνολα δεδομένων με αυτοεπιβλεπόμενους στόχους. Η επιτυχία τους δεν περιορίζεται μόνο στην επεξεργασία φυσικής γλώσσας αλλά επεκτείνεται σε τομείς όπως η όραση υπολογιστών, η βιοπληροφορική και η πρόβλεψη χρονοσειρών, καθιστώντας τους μετασχηματιστές απαραίτητο εργαλείο στη σύγχρονη τεχνητή νοημοσύνη.

1.1.5 Ενισχυτική Μάθηση (Reinforcement Learning)

Η Ενισχυτική Μάθηση (Reinforcement Learning – RL) αποτελεί μια κατηγορία Μηχανικής Μάθησης (Machine Learning – ML). Το RL εστιάζει στην εκπαίδευση πρακτόρων οι οποίοι αλληλεπιδρούν με ένα δυναμικό περιβάλλον, λαμβάνοντας ανατροφοδότηση με τη μορφή επιβραβεύσεων ή ποινών, προκειμένου να μάθουν βέλτιστες συμπεριφορές [27]. Σε αντίθεση με τη Μάθηση με Επίβλεψη (Supervised Learning), το RL δεν απαιτεί δεδομένα με ετικέτες, ούτε ρητές οδηγίες για την επίτευξη ενός στόχου, αν και μπορεί να ενσωματώσει γνώση από ειδικούς ή το πεδίο εφαρμογής. Η ενισχυτική μάθηση συνιστά έναν από τους τρεις θεμελιώδεις άξονες της μηχανικής μάθησης, μαζί με τη Μάθηση με Επίβλεψη και τη Μάθηση χωρίς Επίβλεψη. Ενώ η πρώτη βασίζεται σε σύνολα δεδομένων με γνωστές εισόδους-εξόδους και η δεύτερη επιδιώκει την ανίχνευση κρυμμένων δομών ,το RL διακρίνεται από την ιδιαιτερότητά του να μαθαίνει μέσω αλληλεπίδρασης με το περιβάλλον. Ο πράκτορας λαμβάνει αποφάσεις, παρατηρεί τις συνέπειες των ενεργειών του και προσαρμόζει σταδιακά την πολιτική του με

σκοπό τη μεγιστοποίηση της αθροιστικής επιβράβευσης σε βάθος χρόνου. Η δυνατότητα λήψης αποφάσεων υπό αβεβαιότητα και η ικανότητα διαχείρισης καθυστερημένων επιβραβεύσεων καθιστούν το RL κατάλληλο για εφαρμογές όπως η ρομποτική, τα αυτόνομα οχήματα, τα συστήματα πόρων και τα ευφυή παιχνίδια. ΤΟ RL βασίζεται σε βασικά δομικά στοιχεία [42]: τον πράκτορα (agent), το περιβάλλον (environment), την κατάσταση (state), τη δράση (action) και την επιβράβευση (reward). Ο πράκτορας λαμβάνει ως είσοδο την τρέχουσα κατάσταση του περιβάλλοντος και επιλέγει μια ενέργεια με βάση την πολιτική του. Το περιβάλλον ανταποκρίνεται, επιστρέφοντας νέα κατάσταση και αντίστοιχη επιβράβευση, επιτρέποντας στον πράκτορα να ενημερώσει την πολιτική του. Η αλληλεπίδραση αυτή διαμορφώνεται μαθηματικά ως Μαρκοβιανή Διεργασία Αποφάσεων (Markov Decision Process – MDP). Σε ένα MDP, ορίζεται ένα σύνολο καταστάσεων, ένα σύνολο ενεργειών, μια συνάρτηση μετάβασης που περιγράφει την πιθανότητα μετάβασης σε νέα κατάσταση, μια συνάρτηση επιβράβευσης και ένας συντελεστής προεξόφλησης $\gamma \in [0, 1]$ που καθορίζει τη βαρύτητα μελλοντικών επιβραβεύσεων. Στόχος του πράκτορα είναι η εκμάθηση μιας βέλτιστης πολιτικής $\pi^*(a|s)$ που μεγιστοποιεί τη μαθηματική προσδοκία της συνολικής επιβράβευσης:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}$$

Ο αλγόριθμος **Q-learning** συνιστά θεμελιώδη τεχνική στο RL. Βασίζεται στη συνάρτηση Q, η οποία εκτιμά τη συνολική αναμενόμενη επιβράβευση για κάθε ζεύγος κατάστασης και ενέργειας. Η εκτίμηση πραγματοποιείται βάσει της εξίσωσης Bellman:

$$Q(x, a) = \mathbb{E}\left[Y + \gamma \max_{a'} Q(x', a') \mid x, a\right]$$

όπου x' είναι η επόμενη κατάσταση και a' η ενέργεια που ακολουθεί. Στην πράξη, η εκτίμηση υλοποιείται επαναληπτικά μέσω κανόνων ενημέρωσης:

$$Q(x, a) \leftarrow Q(x, a) + \alpha\left[Y + \gamma \max_{a'} Q(x', a') - Q(x, a)\right]$$

όπου α είναι ο ρυθμός μάθησης. Υπό κατάλληλες συνθήκες, όπως η σταδιακή μείωση του α, η επαρκής εξερεύνηση του χώρου και η χρήση προεξόφλησης $\gamma < 1$, η μέθοδος εγγυάται σύγκλιση στη βέλτιστη πολιτική.

Η Βαθιά Ενισχυτική Μάθηση (Deep Reinforcement Learning – DRL) αποτελεί την προέκταση του RL με χρήση βαθιών νευρωνικών δικτύων. Ο όρος «βαθιά» αναφέρεται στη χρήση πολλαπλών στρωμάτων τεχνητών νευρώνων για την προσέγγιση συναρτήσεων πολιτικής ή τιμής. Η πρώτη επιτυχής υλοποίηση DRL ήταν το Deep Q-Network (DQN) από την ομάδα DeepMind [49], το οποίο χρησιμοποίησε συνελικτικά δίκτυα για την εκπαίδευση πολιτικής σε περιβάλλοντα με συνεχή χώρο καταστάσεων. Για τη σταθεροποίηση της μάθησης εισήχθησαν δύο τεχνικές: η επανάληψη εμπειριών (experience replay) και το target network. Περαιτέρω βελτιώσεις περιλαμβάνουν το Double DQN [30], που μειώνει την υπερεκτίμηση των Q-τιμών μέσω διαχωρισμού της επιλογής και της εκτίμησης ενέργειας, το DDPG (Deep Deterministic Policy Gradient) [44] για συνεχή έλεγχο, και το TD3 (Twin Delayed DDPG) [25], το οποίο εισάγει καθυστερημένες ενημερώσεις και ζευγάρια συναρτήσεων Q για αυξημένη σταθερότητα. Η ενισχυτική μάθηση και ιδιαίτερα οι βαθιές επεκτάσεις της, προσφέρουν ισχυρά εργαλεία για την επίλυση προβλημάτων λήψης αποφάσεων σε δυναμικά, αβέβαια και κατανεμημένα περιβάλλοντα, καθιστώντας την ιδανική προσέγγιση για την κατανομή πόρων σε υποδομές τύπου edge–cloud, όπου η βέλτιστη διαχείριση σε πραγματικό χρόνο είναι κρίσιμη.

1.1.6 Πολυπρακτορικά Συστήματα (Multi-Agent Systems)

Τα πολυπρακτορικά συστήματα (Multi-Agent Systems - MAS) [9] αποτελούνται από πολλαπλές αυτόνομες οντότητες λήψης αποφάσεων—τους πράκτορες—οι οποίοι λειτουργούν σε ένα κοινό περιβάλλον, λαμβάνοντας τοπικές αποφάσεις βάσει μερικών παρατηρήσεων και αποκεντρωμένης πληροφορίας. Στο πλαίσιο του υπολογιστικού νέφους και της υποδομής αιχμής (edge-cloud), κάθε πράκτορας μπορεί να αναλάβει τον ρόλο ενός προγραμματιστή εργασιών ή διαχειριστή πόρων σε τοπικό επίπεδο. Η αποκεντρωμένη φύση των MAS προσφέρει σημαντικά πλεονεκτήματα έναντι των κεντρικοποιημένων προσεγγίσεων. Οι παραδοσιακοί αλγόριθμοι βελτιστοποίησης αντιμετωπίζουν εκθετική αύξηση της υπολογιστικής πολυπλοκότητας καθώς αυξάνεται ο αριθμός των οντοτήτων. Αντίθετα, το πολυπρακτορικό μοντέλο rollout βασίζεται σε τοπικούς υπολογισμούς και κλιμακώνεται γραμμικά ως προς τον αριθμό των πρακτόρων, καθιστώντας το ιδιαίτερα κατάλληλο για σύγχρονες υποδομές νέφους. Επιπλέον, τα MAS υποστηρίζουν φυσικά κατανεμημένη λειτουργία, με τους πράκτορες να

δρουν ασύγχρονα, παράλληλα και με ελάχιστη επικοινωνία. Αυτή η ευελιξία τα καθιστά κατάλληλα για περιβάλλοντα όπου η κεντρική διαχείριση είναι ανέφικτη λόγω καθυστερήσεων, βλαβών ή γεωγραφικής διασποράς. Ένα επιπρόσθετο πλεονέκτημα αποτελεί η δυνατότητα λειτουργίας υπό ποικίλα πρότυπα επικοινωνίας—από πλήρως ανεξάρτητους πράκτορες έως περιορισμένα συνεργατικά σχήματα—επιτρέποντας την προσαρμογή του συστήματος σε ένα ευρύ φάσμα πραγματικών σεναρίων. Τα πολυπρακτορικά συστήματα επεκτείνονται φυσικά και σε προβλήματα ενισχυτικής μάθησης (Reinforcement Learning), διευρύνοντας τις δυνατότητές τους σε δυναμικά και αβέβαια περιβάλλοντα που απαιτούν διαχείριση πόρων.

1.2 Βιβλιογραφική Έρευνα

Πρόβλεψη Πόρων Υπολογιστικού Νέφους

Κίνητρα για Πρόβλεψη σε Υπολογιστικά Συστήματα Νέφους

Η ταχεία εξάπλωση και ευρεία υιοθέτηση του υπολογιστικού νέφους σε επιχειρήσεις και οργανισμούς καθιστά κρίσιμη την ανάγκη για αποτελεσματική επεξεργασία και πρόβλεψη των χρονοσειρών που παράγονται από τέτοια συστήματα. Η διαχείριση πόρων αποτελεί κεντρική πρόκληση, απαιτώντας προσαρμοστικές και προβλεπτικές στρατηγικές για να ανταποκριθεί στις συνεχώς μεταβαλλόμενες απαιτήσεις των φορτίων εργασίας.[50]

Οι χρονοσειρές αναφέρονται σε ακολουθίες παρατηρήσεων οι οποίες συλλέγονται σε διαδοχικά χρονικά διαστήματα. Σε περιβάλλοντα υπολογιστικού νέφους, οι χρονοσειρές προέρχονται από συστήματα τηλεμετρίας τα οποία καταγράφουν μετρικές απόδοσης και κατανάλωσης πόρων, όπως χρήση CPU, μνήμης, δίσκου. Αυτά τα δεδομένα αποτυπώνουν τη δυναμική συμπεριφορά της υποδομής και είναι απαραίτητα για την κατανόηση της κατάστασης του συστήματος, την ανίχνευση ανωμαλιών και την πρόβλεψη μελλοντικών συνθηκών λειτουργίας. Ο φόρτος εργασίας στο υπολογιστικό νέφος είναι πολυδιάστατος, μη σταθερός και συχνά παρουσιάζει αιφνίδιες μεταβολές. Πολλές υπάρχουσες μέθοδοι πρόβλεψης αποτυγχάνουν να ανταποκριθούν σε αυτή τη μεταβλητότητα, οδηγώντας είτε σε υπέρ-εκχώρηση είτε σε ανεπαρκή εκχώρηση πόρων. Η υπέρ-εκχώρηση οδηγεί σε μειωμένη χρήση της υποδομής και αυξημένο λειτουργικό κόστος, ενώ η ανεπαρκής εκχώρηση προκαλεί μειωμένη απόδοση και πιθανή παραβίαση των Συμφωνιών Επιπέδου Υπηρεσιών (SLA) [16]. Για παράδειγμα, όταν παρατηρείται ξαφνική αύξηση αιτημάτων χρηστών, το σύστημα μπορεί να υπερφορτωθεί και να εμφανίσει καθυστερήσεις ή αποτυχίες υπηρεσιών. Αντίστροφα, σε περιόδους χαμηλής ζήτησης, οι πόροι μένουν αχρησιμοποίητοι, προκαλώντας άσκοπη ενεργειακή κατανάλωση και οικονομική σπατάλη [16]. Αυτά τα φαινόμενα καθιστούν σαφές ότι απαιτείται ακριβής και έγκαιρη πρόβλεψη για την αποτελεσματική διαχείριση πόρων.

Πρόβλεψη για Κατανομή Πόρων

Ο βασικός στόχος της πρόβλεψης σε περιβάλλοντα υπολογιστικού νέφους είναι η ακριβής εκτίμηση μελλοντικών προτύπων κατανάλωσης πόρων, ώστε να είναι δυνατή η προληπτική και αποδοτική κατανομή. Η πρόβλεψη λειτουργεί ως κρίσιμο προκαταρκτικό στάδιο για εξελιγμένες τεχνικές διαχείρισης, όπως αυτόματη κλιμάκωση (autoscaling), εξισορρόπηση φορτίου (load balancing) και ενισχυτική μάθηση για ενορχήστρωση πόρων. Ως μεταβλητή στόχος, η χρήση της επεξεργαστικής ισχύς είναι η συνηθέστερη επιλογή, καθώς αποτελεί άμεσο δείκτη της πίεσης στο σύστημα και συσχετίζεται άμεσα με την απόδοση των εφαρμογών και τις αποφάσεις κλιμάκωσης. Η πρόβλεψή της σε κατάλληλη χρονική ανάλυση είναι θεμελιώδης για τη διατήρηση της ποιότητας υπηρεσίας και τη μείωση του λειτουργικού κόστους.[33][72]

Κλασικές Μέθοδοι Πρόβλεψης

Η πρόβλεψη χρονοσειρών σε περιβάλλοντα νέφους έχει αρχικά προσεγγιστεί μέσω κλασικών στατιστικών μεθόδων όπως τα AutoRegressive (AR), Moving Average (MA), AutoRegressive Integrated Moving Average (ARIMA), και Vector AutoRegression (VAR) [51]. Αυτές οι μέθοδοι προϋποθέτουν στασιμότητα και γραμμικές σχέσεις μεταξύ των δεδομένων, προσφέροντας καλή ακρίβεια σε προβλέψιμα και χαμηλού θορύβου περιβάλλοντα. Πρόσθετα στατιστικά εργαλεία, όπως οι μέσες τιμές, οι τυπικές αποκλίσεις, η συνάρτηση κατανομής (CDF) και ο συντελεστής μεταβλητότητας (CoV) [20], χρησιμοποιούνται για τον χαρακτηρισμό της διακύμανσης των μετρικών. Για βραχυπρόθεσμη πρόβλεψη, συχνά εφαρμόζονται τα μοντέλα SMA και ARIMA, με το τελευταίο να αποδίδει ικανοποιητικά όταν τα δεδομένα παρουσιάζουν ομαλότητα. Ωστόσο, σε περιπτώσεις μη γραμμικότητας ή έντονου θορύβου, η ακρίβεια αυτών των μοντέλων μειώνεται δραστικά. Η στατιστική προσέγ-

γιση πλεονεκτεί λόγω της διαφάνειας και του χαμηλού υπολογιστικού κόστους, ωστόσο δεν επαρκεί για την αποτύπωση πολύπλοκων δυναμικών, χαρακτηριστικά που κυριαρχούν στα σύγχρονα cloud-native συστήματα.

Μοντέλα Μηχανικής και Βαθιάς Μάθησης

Για την αντιμετώπιση των περιορισμών των κλασικών μεθόδων, έχουν προταθεί τεχνικές μηχανικής μάθησης (ML) και βαθιάς μάθησης (DL) [26]. Τα ML μοντέλα, όπως η γραμμική παλινδρόμηση, τα SVM, τα Random Forests και οι k-κοντινότεροι γείτονες (k-NN), προσφέρουν μεγαλύτερη ικανότητα γενίκευσης μέσω μάθησης από δεδομένα. Η χρήση νευρωνικών δικτύων, σε συνδυασμό με τεχνικές όπως το παράθυρο εισόδου (windowing), έχει αποδειχθεί αποτελεσματική στην πρόβλεψη πόρων εικονικών μηχανών(SVM) [35]. Παρόλο που τα SVM παρουσιάζουν καλές επιδόσεις σε συχνές διακυμάνσεις φορτίου, η απόδοσή τους υποβαθμίζεται σε μεγάλα datasets [52].

Τα μοντέλα LSTM υπερέχουν στην αποτύπωση χρονικών εξαρτήσεων και σειριακών μοτίβων, επιλύοντας το πρόβλημα εξαφάνισης βαθμίδων που χαρακτηρίζει τα συμβατικά RNN [33]. Πολλαπλές μελέτες έχουν καταδείξει την υπεροχή των LSTM έναντι των παραδοσιακών στατιστικών προσεγγίσεων στην πρόβλεψη φορτίου[39]. Οι μετασχηματιστές προσφέρουν ακόμη μεγαλύτερη ακρίβεια, ιδίως σε περιπτώσεις σύνθετων ή πολυπαραγοντικών χρονικών εξαρτήσεων[4]. Ο μηχανισμός αυτοπροσοχής επιτρέπει στο μοντέλο να εστιάζει επιλεκτικά σε σχετικές χρονικές στιγμές, αποδίδοντας βελτιωμένα αποτελέσματα σε μεταβλητά ή εκρηκτικά πρότυπα κατανάλωσης [54]. Αν και τα DL μοντέλα είναι πιο ακριβή και ανθεκτικά στη μη γραμμικότητα, απαιτούν μεγάλους όγκους δεδομένων, ισχυρούς υπολογιστικούς πόρους και προσεκτική παραμετροποίηση. Επιπλέον, η ερμηνευσιμότητά τους παραμένει πρόκληση για την παραγωγική τους εφαρμογή σε ευαίσθητα συστήματα.

Μηχανισμοί Κατανομής Πόρων

Η κατανομή πόρων (Resource Allocation – RA) αποτελεί θεμελιώδες στοιχείο στα υπολογιστικά περιβάλλοντα νέφους, αφορώντας τη διανομή περιορισμένων υπολογιστικών, αποθηκευτικών και δικτυακών πόρων σε ανταγωνιστικές διεργασίες ή χρήστες. Η ύπαρξη αποδοτικών μηχανισμών RA είναι κρίσιμη για τη βελτιστοποίηση της συνολικής απόδοσης, τη μεγιστοποίηση της χρήσης των πόρων, τη μείωση του κόστους και τη συμμόρφωση με απαιτήσεις ποιότητας υπηρεσίας [22]. Οι μηχανισμοί RA διακρίνονται σε δύο βασικές κατηγορίες: στατικούς και δυναμικούς.

Στατικοί Μηχανισμοί Κατανομής Πόρων

Οι στατικές στρατηγικές κατανομής πόρων στηρίζονται σε κανόνες που καθορίζονται εκ των προτέρων και παραμένουν σταθεροί κατά την εκτέλεση των διεργασιών. Αυτοί οι μηχανισμοί είναι κατάλληλοι για σταθερά ή προβλέψιμα φορτία. Τυπικές τεχνικές περιλαμβάνουν γραμμικό και μικτό ακέραιο προγραμματισμό (MILP, MINLP), ενώ σε πιο σύνθετα περιβάλλοντα εφαρμόζονται στοχαστικά μοντέλα για να ενσωματωθεί η αβεβαιότητα στη ζήτηση ή τις τιμές [15].. Οι λύσεις υπολογίζονται εκτός χρόνου με χρήση εργαλείων όπως CPLEX ή Gurobi και εφαρμόζονται σε εφαρμογές όπως αρχική τοποθέτηση εικονικών μηχανών ή admission control [75]. Παρά τα πλεονεκτήματα σε απλότητα και προβλεψιμότητα, οι στατικοί μηχανισμοί δεν προσαρμόζονται σε απότομες μεταβολές, γεγονός που οδηγεί σε υποεκμετάλλευση ή υπερφόρτωση των πόρων [76].

Δυναμικοί Μηχανισμοί Κατανομής Πόρων

Οι δυναμικές στρατηγικές κατανομής πόρων (DRA) είναι απαραίτητες για περιβάλλοντα με έντονες διακυμάνσεις. Αυτοί οι μηχανισμοί προσαρμόζουν τη χρήση CPU, μνήμης και άλλων πόρων με βάση πραγματικά δεδομένα ή προβλέψεις. **Αυτόματη κλιμάκωση (Auto-scaling)**: Περιλαμβάνει κατακόρυφη (εντός VM) και οριζόντια (προσθήκη/αφαίρεση VM ή containers) κλιμάκωση. Η πρώτη έχει μικρή καθυστέρηση, ενώ η δεύτερη προσφέρει μεγαλύτερη ευελιξία. **Προβλεπτική διαχείριση πόρων**: Βασίζεται σε ιστορικά δεδομένα και χρησιμοποιεί γραμμική παλινδρόμηση, φίλτρα Kalman ή multiplexing για πρόβλεψη μελλοντικής ζήτησης [19]. **Βελτιστοποίηση πολλαπλών στόχων**: Εφαρμόζεται ως πρόβλημα πολλαπλής αντικειμενικής βελτιστοποίησης, συνδυάζοντας επιδόσεις, κόστος και ενέργεια. Έχουν προταθεί αλγόριθμοι όπως coral-reefs optimization με θεωρία παιγνίων για δυναμική ανακατανομή πόρων [24]. **Συστήματα βασισμένα σε κανόνες**: Λειτουργούν με όρια ενεργοποίησης (π.χ. CPU $> 80\%$) και είναι απλά αλλά λιγότερο ακριβή [66]. **Συστήματα ανάδρασης (feedback control)**: Εμπνευσμένα από θεωρία ελέγχου, χρησιμοποιούν PID controllers για συνεχή ρύθμιση. Παρέχουν σταθερότητα και δυνατότητα αυτόνομης απόκρισης

[19]. **Ενεργειακά ευαίσθητες στρατηγικές**: Επιχειρούν ελαχιστοποίηση της κατανάλωσης ενέργειας μέσω συγχώνευσης φόρτου και απενεργοποίησης αδρανών κόμβων (π.χ. το σύστημα pMapper) . [69]. Παρά τα οφέλη τους, οι DRA μηχανισμοί αντιμετωπίζουν προκλήσεις όπως η ακρίβεια πρόβλεψης, το κόστος μετεγκατάστασης, η καθυστέρηση απόκρισης και η ανάγκη εξισορρόπησης αντικρουόμενων στόχων [36].

Ο Ρόλος της Ενισχυτικής Μάθησης (RL)

Οι παραδοσιακές μέθοδοι εκχώρησης εργασιών στο νέφος και στο edge, απαιτούν ισχυρές υποθέσεις και αδυνατούν να προσαρμοστούν σε δυναμικά περιβάλλοντα. Η Ενισχυτική Μάθηση (RL) προσφέρει ένα εναλλακτικό πλαίσιο, όπου οι πράκτορες μαθαίνουν βέλτιστες στρατηγικές μέσω δοκιμής και σφάλματος, χωρίς γνώση του υποκείμενου μοντέλου. Για την αντιμετώπιση της εκθετικής πολυπλοκότητας σε μεγάλες κλίμακες, η Βαθιά Ενισχυτική Μάθηση (DRL) αξιοποιεί νευρωνικά δίκτυα για την εκμάθηση πολιτικών σε χώρους υψηλών διαστάσεων. Το Deep Q-Network (DQN) έχει αποδείξει την αποτελεσματικότητά του σε περιβάλλοντα edge–cloud [49]. Η επέκταση σε Πολυπρακτορική DRL (MADQN) επιτρέπει την αποκεντρωμένη λήψη αποφάσεων από πολλαπλούς πράκτορες, προσφέροντας καλύτερη κλιμάκωση, ανθεκτικότητα και παράλληλη λειτουργία. Η προσέγγιση αυτή είναι ιδανική για σύγχρονες κατανεμημένες υποδομές, όπου ο κεντρικός συγχρονισμός είναι πρακτικά ασύμφορος.

1.3 Διατύπωση Προβλήματος και Μοντελοποίηση Συστήματος

Η παρούσα διπλωματική εργασία επικεντρώνεται στον σχεδιασμό ενός ευφυούς συστήματος κατανομής πόρων ο οποίος λειτουργεί σε σύγχρονα πολυεπίπεδα περιβάλλοντα edge–cloud υποδομών. Το βασικό πρόβλημα που αντιμετωπίζεται είναι η αποτελεσματική και αποδοτική διαχείριση εργασιών (jobs) με ποικίλα χαρακτηριστικά όπως ευαισθησία σε καθυστερήσεις, διαφορετικά επίπεδα προτεραιότητας και απαιτήσεις πόρων.

Μοντέλο Συστήματος

Η υποδομή που μελετάται αποτελείται από τρία είδη διαθέσιμων πόρων:

- **Near-Edge** (T_0): Το πιο κοντινό επίπεδο προς τον τελικό χρήστη, με χαμηλή καθυστέρηση (latency), αλλά υψηλό λειτουργικό κόστος και περιορισμένη χωρητικότητα.
- **Far-Edge** (T_1): Μεσαίο επίπεδο με μέτρια καθυστέρηση και κόστος, και μέτρια διαθεσιμότητα πόρων.
- **Cloud** (T_2): Το πιο απομακρυσμένο επίπεδο, με υψηλή καθυστέρηση λόγω απόστασης, αλλά σχεδόν απεριόριστους πόρους και το χαμηλότερο κόστος ανά μονάδα CPU.

Κάθε επίπεδο T_i περιγράφεται ως:

$$T_i = (M_i, Cap_i, L_i, Cost_i, P_i)$$

όπου M_i το σύνολο των μηχανημάτων, Cap_i η CPU χωρητικότητα ανά μηχάνημα, L_i η δικτυακή καθυστέρηση, $Cost_i$ το κόστος λειτουργίας ανά CPU-ώρα και P_i η κατανάλωση ισχύος ανά πυρήνα σε Watt.

Κατάσταση Πόρων και Χρήση

Σε κάθε χρονική στιγμή τ, το σύστημα παρακολουθεί τη χρήση και διαθεσιμότητα CPU πόρων ανά επίπεδο:

$$Usage_i(\tau) = \sum_{m \in M_i} \sum_{j \in \text{ActiveJobs}_m(\tau)} \text{cpu\_demand}_j,$$

$$Avail_i(\tau) = TotalCap_i - Usage_i(\tau),$$

$$Util_i(\tau) = \frac{Usage_i(\tau)}{TotalCap_i}$$

με την προϋπόθεση ότι η χρήση διατηρείται κάτω από ένα μέγιστο όριο Max_Util για αποφυγή κορεσμού και διασφάλιση σταθερής απόδοσης.

Περιγραφή Εργασιών

Κάθε εργασία j περιγράφεται ως:

$$j = (id, start\_time, end\_time, cpu\_demand, duration, priority)$$

όπου τα πεδία αναφέρονται σε αναγνωριστικό, χρόνο άφιξης, αναμενόμενο χρόνο ολοκλήρωσης, ζήτηση για CPU, εκτιμώμενη διάρκεια εκτέλεσης και επίπεδο προτεραιότητας. Οι προτεραιότητες είναι ακέραιες τιμές όπου μικρότερες τιμές σημαίνουν υψηλότερη προτεραιότητα και άρα χαμηλότερη ανοχή σε καθυστέρηση.

Πολυ-παραγοντική Διατύπωση Προβλήματος

Το πρόβλημα κατανομής πόρων ενσωματώνει πολλαπλούς αντικειμενικούς στόχους:

- **Μείωση κόστους και ενέργειας:** Προκειμένου να επιτευχθεί βιωσιμότητα και οικονομία, λαμβάνονται υπόψη το κόστος ανά CPU ανα ώρα και η κατανάλωση ενέργειας, με το κόστος να υπολογίζεται ως

  $$Cost_j^i = cpu\_demand_j \times \frac{duration_j}{3600} \times Cost_i,$$

  και η ενέργεια ως

  $$Energy_j^i = P_i \times cpu\_demand_j \times \frac{duration_j}{3600}.$$

- **Λαμβάνοντας υπόψη καθυστερήσεις (latency):** Η καθυστέρηση κάθε εργασίας καθορίζεται κυρίως από τη θέση της στο δίκτυο και τους χρόνους αναμονής, με το βασικό μοντέλο

  $$Latency_j^i = L_i.$$

- **Τήρηση Συμφωνιών Επιπέδου Υπηρεσίας (SLA):** Για κάθε επίπεδο προτεραιότητας p, υπάρχει μέγιστο επιτρεπτό όριο καθυστέρησης SLA_p που πρέπει να ικανοποιείται (π.χ. 5 δευτερόλεπτα για την υψηλότερη προτεραιότητα). Παραβιάσεις SLA επιφέρουν ποινές, ενισχύοντας τη σημασία της άμεσης εκτέλεσης των κρίσιμων εργασιών.

Δυναμική Άφιξη και Ομαδοποίηση Εργασιών

Οι εργασίες φτάνουν δυναμικά και ομαδοποιούνται σε χρονικά παράθυρα διάρκειας w για αποτελεσματικό προγραμματισμό σε παρτίδες. Οι εργασίες εντός κάθε παραθύρου ταξινομούνται κατά προτεραιότητα, εξασφαλίζοντας ότι θα ξεκινήσουν να εκτελούνται πρώτα οι πιο κρίσιμες.

Κατηγοριοποιούνται σε κλάσεις:

- **Κλάση I (Κρίσιμες):** Προτεραιότητα 1–2, απαιτούν ελάχιστη καθυστέρηση.
- **Κλάση II (Ευέλικτες):** Προτεραιότητα 3–4, ανεκτικές σε μέτριες καθυστερήσεις.
- **Κλάση III (Best-effort):** Προτεραιότητα 5, κατάλληλες για cloud εκτέλεση καθώς έχουν τεράστια ανοχή σε καθυστέρηση.

Συνάρτηση Ανταμοιβής (Reward Function)

Η απόφαση τοποθέτησης εργασίας σε cluster i αξιολογείται μέσω σύνθετης συνάρτησης ανταμοιβής:

$$R_j^i = w_1 R_{base} + w_2 R_{\text{placement}}(j,i) - w_3 R_{cost}(j,i) - w_4 R_{latency}(j,i) - w_5 R_{energy}(j,i) + w_6 R_{\text{SLA}}(j),$$

όπου κάθε όρος αναπαριστά διαφορετική διάσταση:

- R_{base}: Βασική ανταμοιβή για επιτυχή κατανομή ή ποινή για αποτυχία.
- $R_{\text{placement}}$: Προσαρμοσμένη ανταμοιβή/ποινή που ευνοεί τοποθετήσεις σύμφωνα με την προτεραιότητα και την τρέχουσα χρήση, αποτρέποντας π.χ. υπερφόρτωση edge επιπέδων.

- Ποινές που σχετίζονται με κόστος, καθυστέρηση και ενέργεια.
- Ποινή SLA που εφαρμόζεται όταν ξεπερνιέται το όριο ανοχής σε καθυστέρηση για την κάθε κατηγορία, δίνοντας μεγαλύτερη ποινή όταν αυτό συμβαίνει σε εργασίες με μεγάλη προτεραιότητα.

Η παραμετροποίηση βαρών w_i επιτρέπει ευελιξία στην έμφαση σε συγκεκριμένους στόχους, όπως αυστηρή τήρηση SLA ή μείωση κόστους.

Μηχανισμός Μάθησης

Η κατανομή πόρων αντιμετωπίζεται ως πρόβλημα λήψης αποφάσεων σε δυναμικό περιβάλλον, με τη μέθοδο Q-learning ως βασικό μηχανισμό μάθησης. Κάθε πράκτορας μαθαίνει να εκτιμά την αναμενόμενη μελλοντική ανταμοιβή $Q(s, a)$ για κάθε κατάσταση s και ενέργεια a, οδηγώντας σε βέλτιστη πολιτική.

Η βελτιστοποίηση της συνολικής ανταμοιβής ισορροπεί μεταξύ μείωσης κόστους, σεβασμού προτεραιοτήτων, συμμόρφωσης με SLA και διαχείρισης καθυστερήσεων.

Αρχιτεκτονική Πολυ-Πρακτόρων (Multi-Agent)

Η υλοποίηση είναι αποκεντρωμένη, με πολλαπλούς πράκτορες (gateways) που διαχειρίζονται διακριτά υποσύνολα εργασιών. Κάθε πράκτορας παρατηρεί το τοπικό περιβάλλον, λαμβάνει αποφάσεις και ακολουθεί κοινή πολιτική.

Αυτή η διατύπωση και το μοντέλο αποτελούν τη βάση για την ανάπτυξη ενός ευφυούς, προσαρμοστικού μηχανισμού προγραμματισμού που ανταποκρίνεται σε σύνθετες απαιτήσεις και περιορισμούς των σύγχρονων edge–cloud υποδομών, ενσωματώνοντας τόσο οικονομικά όσο και τεχνικά κριτήρια απόδοσης.

1.4 Πειράματα

1.4.1 Σύνολα Δεδομένων

Καθαρισμός και Προεπεξεργασία Δεδομένων

Το σύνολο δεδομένων **Alibaba Cluster Trace 2017** περιλαμβάνει δεδομένα τηλεμετρίας σε υποδομή υπολογιστικού νέφους μεγάλης κλίμακας. Αν και το αρχικό υλικό είναι εξαιρετικά πλούσιο σε πληροφορία, παρουσιάζει σημαντικές προκλήσεις για άμεση χρήση σε μοντελοποίηση, λόγω θορύβου, απωλειών και ετερογενών χαρακτηριστικών. Ακολουθώντας τη μεθοδολογία των Lackinger et al. [40], υλοποιήθηκε μια πολυφασική διαδικασία καθαρισμού και κανονικοποίησης, με στόχο την εξαγωγή χρονικά ευθυγραμμισμένων, αριθμητικά συνεπών και σημασιολογικά εμπλουτισμένων συνόλων δεδομένων, κατάλληλων τόσο για πρόβλεψη όσο και για ενισχυτική μάθηση.

Η αρχική δομή του αρχείου περιλάμβανε αναγνωριστικά εργασιών και μηχανών, χρονικές σημάνσεις, μέγιστη και μέση κατανάλωση CPU, καθώς και τιμές κατανάλωσης μνήμης, πολλές από τις οποίες απουσίαζαν. Έτσι, αποκλείστηκαν πλήρως πεδία με μη αριθμητικές ή ελλιπείς τιμές, όπως το task ID, τα πεδία κατάστασης (status), και η κατανάλωση μνήμης. Ακολούθως, εφαρμόστηκε μετατροπή των χρονικών στιγμών σε σχετικές χρονικές αναφορές και κανονικοποίηση όλων των αριθμητικών πεδίων μέσω MinMax scaling στο διάστημα $[0, 1]$, για σταθεροποίηση της εκπαίδευσης των μοντέλων.

Τέλος, προστέθηκε ετικέτα προτεραιότητας για κάθε εργασία (από 1 έως 5), βάσει SLA κατηγοριοποίησης, ώστε να καταστεί δυνατή η ποιοτική διαστρωμάτωση των εργασιών ανάλογα με τις απαιτήσεις καθυστέρησης και η δυναμική ενσωμάτωση στην ανταμοιβή του πράκτορα. Το καθαρισμένο dataset αξιοποιείται τόσο ως ακολουθία εισόδων για πρόβλεψη πόρων, όσο και στο περιβάλλον ενισχυτικής μάθησης.

Table 1.1: Απόσπασμα του αρχικού ακατέργαστου συνόλου δεδομένων (Alibaba trace)

start_ts	end_ts	job id	task id	machine id	max cpu	avg cpu	max mem	avg mem
41562	41618	120	686	299	1.50	0.29	NaN	NaN
41561	41619	120	686	1279	0.89	0.28	NaN	NaN
41562	41617	120	686	828	0.94	0.29	NaN	NaN

Table 1.2: Απόσπασμα του καθαρισμένου συνόλου δεδομένων με εκχωρημένες προτεραιότητες

Start Timestamp	End Timestamp	Job ID	Machine ID	Max CPU	Avg CPU	Priority
2017-01-01 00:00:00	2017-01-01 00:00:01	10528874	518	1.01	1.01	1
2017-01-01 00:00:01	2017-01-01 00:00:55	30629148	352	1.01	0.98	3
2017-01-01 00:00:01	2017-01-01 00:00:56	30629079	258	1.00	0.97	3
2017-01-01 00:00:01	2017-01-01 00:00:55	30629105	278	1.02	0.99	3
2017-01-01 00:00:02	2017-01-01 00:00:59	30629093	897	1.01	0.96	3

1.4.2 Μοντέλα

Μοντέλα Πρόβλεψης

Η φάση πρόβλεψης της παρούσας μελέτης επικεντρώνεται στην εκτίμηση της μελλοντικής ζήτησης υπολογιστικών πόρων, και συγκεκριμένα της χρήσης CPU, για παρτίδες εργασιών (batch jobs), με βάση ιστορικά τηλεμετρικά δεδομένα από περιβάλλον μεγάλης κλίμακας. Η ικανότητα πρόβλεψης αποτελεί κρίσιμο συστατικό ενός ευφυούς μηχανισμού διαχείρισης πόρων, καθώς καθιστά δυνατή τη λήψη αποφάσεων που δεν περιορίζονται αποκλειστικά στην τρέχουσα κατάσταση του συστήματος, αλλά ενσωματώνουν και την αναμενόμενη μελλοντική ζήτηση.

Από το σύνολο δεδομένων των 8 εκατομμυρίων εγγραφών επιλέχθηκε ένα αντιπροσωπευτικό υποσύνολο 500.000 εγγραφών για σκοπούς εκπαίδευσης, ώστε να διασφαλιστεί τόσο η υπολογιστική αποδοτικότητα όσο και η στατιστική ποικιλία.

Η φάση αυτή εξυπηρετεί δύο βασικούς στόχους: αφενός, την παραγωγή ακριβών βραχυπρόθεσμων προβλέψεων της ζήτησης CPU, οι οποίες αξιοποιούνται από το σύστημα RL για τη βέλτιστη κατανομή των εργασιών, και αφετέρου, τη σύγκριση δύο σύγχρονων αρχιτεκτονικών νευρωνικών δικτύων — των **LSTM** και των **Transformer**. Η ανάπτυξη των μοντέλων έγινε με χρήση των βιβλιοθηκών TensorFlow και Keras. Η προετοιμασία των δεδομένων και η δημιουργία των batches πραγματοποιήθηκε μέσω μηχανισμού sequence generator, επιτρέποντας τη διαχείριση μεγάλων όγκων δεδομένων χωρίς υπερφόρτωση της μνήμης. . Και τα δύο μοντέλα εκπαιδεύτηκαν υπό κοινό πειραματικό πρωτόκολλο, εμπνευσμένο από τη μεθοδολογία του Lackinger et al., ώστε να εξασφαλιστεί η συγκρισιμότητα των αποτελεσμάτων. Χρησιμοποιήθηκε ο αλγόριθμος Adam με σταθερό ρυθμό μάθησης, συνάρτηση απώλειας MSE, και ποσοστό επικύρωσης 20%. Η εκπαίδευση έγινε στο Google Colab με χρήση επιτάχυνσης GPU (T4). Ο Πίνακας 1.3 παρουσιάζει τις βασικές παραμέτρους εκπαίδευσης για κάθε μοντέλο.

Table 1.3: Παράμετροι Εκπαίδευσης για LSTM και Transformer

Παράμετρος	LSTM	Transformer
Μήκος εισόδου	60	60
Ορίζοντας πρόβλεψης	1 βήμα	1 βήμα
Μέγεθος δέσμης	32	32
Βελτιστοποιητής	Adam	Adam
Ρυθμός μάθησης	0.001	0.001
Συνάρτηση απώλειας	MSE	MSE
Εποχές	13	36
Υπομονή για early stopping	5	5
Ποσοστό επικύρωσης	20%	20%
Υποδομή	Google Colab (GPU)	Google Colab (GPU)

Περιβάλλον Κατανομής Πόρων και Εκπαίδευσης Συστήματος Βαθιάς Ενισχυτικής Μάθησης

Για την αξιολόγηση προσαρμοστικών στρατηγικών τοποθέτησης εργασιών σε υποδομές τύπου edge–cloud, σχεδιάστηκε και υλοποιήθηκε ένα προσαρμοσμένο περιβάλλον ενισχυτικής μάθησης, βασισμένο στο πρότυπο `gymnasium`. Το περιβάλλον προσομοιώνει ένα σενάριο μονού πράκτορα, ο οποίος καλείται να εκχωρήσει δυναμικά κάθε εργασία σε μία από τρεις διαθέσιμες βαθμίδες υποδομής: το near-edge, το far-edge ή το κεντρικό cloud.

Η αρχιτεκτονική του περιβάλλοντος απεικονίζεται στο Σχήμα 6.1.1, το οποίο παρουσιάζει την ιεραρχική δομή που υποστηρίζει την τοποθέτηση των εργασιών βάσει επιπέδου επεξεργασίας και γεωγραφικής εγγύτητας. Κάθε εισερχόμενη εργασία περιγράφεται μέσω ενός συνόλου χαρακτηριστικών, που περιλαμβάνουν τη ζητούμενη ισχύ CPU, τη διάρκεια εκτέλεσης και την ευαισθησία καθυστέρησης. Ο πράκτορας παρατηρεί ένα συνεχή διανυσματικό χώρο έξι διαστάσεων, ο οποίος περιλαμβάνει πληροφορίες τόσο για την ίδια την εργασία όσο και για την τρέχουσα κατάσταση των διαθέσιμων υποδομών. Οι περιορισμοί του συστήματος, όπως η διαθεσιμότητα πόρων, οι ενεργειακοί περιορισμοί και τα όρια καθυστέρησης, ενσωματώνονται στο περιβάλλον και επηρεάζουν άμεσα την απόδοση και τη δομή της επιβράβευσης. Για την ρεαλιστική προσομοίωση των ροών εργασίας και τη διατήρηση της χρονικής συνέχειας, εφαρμόστηκε σχήμα προγραμματισμού βάσει παραθύρων. Αντί να αντιμετωπίζονται οι εργασίες μεμονωμένα, ομαδοποιούνται χρονικά με βάση κοινά timestamps. Συγκεκριμένα, συγκροτούνται παράθυρα εισερχόμενων εργασιών με n διαφορετικά στιγμιότυπα έναρξης. Εντός κάθε παραθύρου, οι εργασίες ταξινομούνται βάσει προτεραιότητας και εκχωρούνται κατά φθίνουσα σειρά. Το μέγεθος παραθύρου ορίζεται μέσω του υπερπαραμέτρου n, με τιμή $n = 3$ κατά την εκπαίδευση και $n = 10$ κατά την αξιολόγηση, ώστε να ελεγχθεί η γενίκευση της πολιτικής σε πιο σύνθετα και εκρηκτικά μοτίβα φορτίου.

Το περιβάλλον αξιοποιεί το ίδιο καθαρισμένο σύνολο δεδομένων από το Alibaba Cluster Trace 2017, προσαρμοσμένο για ενισχυτική μάθηση. Η κωδικοποίηση της προτεραιότητας των εργασιών αποτελεί αναπόσπαστο στοιχείο της επιβράβευσης, καθώς αποτυπώνει απαιτήσεις ποιότητας υπηρεσίας. Ο Πίνακας 1.4 παρουσιάζει τις πέντε κατηγορίες προτεραιότητας σε σχέση με την απαιτούμενη καθυστέρηση και τις εφαρμογές στις οποίες αντιστοιχούν.

Table 1.4: Κατηγορίες Προτεραιότητας Εργασιών με Βάση την Καθυστέρηση

Επίπεδο Προτεραιότητας	**Ανοχή Καθυστέρησης**	**Ενδεικτικές Εφαρμογές**
1 – Πολύ Χαμηλή	< 10 ms	Έλεγχος σε πραγματικό χρόνο, Εικονική/Επαυξημένη Πραγματικότητα
2 – Χαμηλή	10–50 ms	Διαδικτυακά παιχνίδια, τηλεδιάσκεψη
3 – Μέτρια	50–200 ms	Υπηρεσίες ιστού
4 – Υψηλή	200–500 ms	Συγχρονισμός δεδομένων, περιοδική παρακολούθηση
5 – Χαμηλής Προτεραιότητας	> 500 ms	Εφεδρικές εργασίες, μαζική επεξεργασία

Η πολιτική τοποθέτησης των εργασιών διαμορφώνεται βάσει της κατηγορίας προτεραιότητας. Εργασίες με πολύ χαμηλή ανοχή καθυστέρησης πρέπει να τοποθετούνται στο near-edge. Εκείνες με μέτρια ανοχή μπορούν να ανατεθούν στο far-edge ή στο cloud. Τέλος, οι εργασίες χαμηλής προτεραιότητας τοποθετούνται ιδανικά στο cloud, προκειμένου να διατηρηθούν οι υποδομές χαμηλής καθυστέρησης διαθέσιμες για κρίσιμες εφαρμογές. Η συνάρτηση επιβράβευσης τιμωρεί την τοποθέτηση υψηλής προτεραιότητας εργασιών σε ακατάλληλες βαθμίδες και ενθαρρύνει την ευθυγράμμιση μεταξύ απαιτήσεων καθυστέρησης και διαθέσιμων πόρων.

Η εκπαίδευση πραγματοποιείται με τη χρήση του πλαισίου Ray RLlib και συγκεκριμένα του αλγορίθμου Standard DQN. Το περιβάλλον καταχωρείται μέσω της διεπαφής `register_env`. Το πειραματικό πρωτόκολλο περιλαμβάνει 500.000 εργασίες, με πλήρη κάλυψη όλων των επιπέδων προτεραιότητας. Η υποδομή που προσομοιώνεται αποτελείται από μία μονάδα near-edge με 16 πυρήνες CPU, τέσσερις μονάδες far-edge επίσης με 16 πυρήνες η καθεμία, και πολλαπλές μονάδες cloud με 64 πυρήνες, οι οποίες προσεγγίζουν θεωρητικά το άπειρο. Η μικρή αυτή υποδομή επιλέγεται για λόγους ταχείας εκπαίδευσης και είναι ανάλογη και του μεγέθους των δεδομένων εκπαίδευσης, ενώ σε φάση αξιολόγησης γίνεται αύξηση της για τον έλεγχο της γενίκευσης της πολιτικής σε ρεαλιστικά σενάρια.

Συνάρτηση Ανταμοιβής και Μοντελοποίηση Αποδοτικότητας Πόρων. Ο αλγόριθμος μάθησης καθοδηγείται από μια σύνθετη συνάρτηση ανταμοιβής, η οποία λαμβάνει υπόψη τα εξής:

- **Λειτουργικό Κόστος:** Ανάλογο της χρέωσης ανά CPU-hour για κάθε επίπεδο:
  - Near-edge: \$0.10
  - Far-edge: \$0.05

  - Cloud: \$0.01

- **Ποινή Καθυστέρησης:** Κάθε job φέρει προτεραιότητα και συνεπώς σχετική απαίτηση καθυστέρησης. Η βασική καθυστέρηση ανά επίπεδο είναι:
  - Near-edge: 1 ms
  - Far-edge: 20 ms
  - Cloud: 100 ms
- **Κατανάλωση Ενέργειας:** Χρησιμοποιείται για να προωθείται η εκτέλεση σε πιο αποδοτικά επίπεδα:
  - Near-edge: 40 W
  - Far-edge: 70 W
  - Cloud: 200 W
- **Αξιοποίηση CPU:** Αν η χρήση CPU παραμένει κάτω από 80%, παρέχεται bonus. Αν η χρήση ξεπερνά τη διαθέσιμη χωρητικότητα, εφαρμόζεται ποινή υπερφόρτωσης.
- **Ανταμοιβή Τοποθέτησης (Placement Reward):** Τοποθετήσεις σε κατάλληλα επίπεδα ανάλογα με την προτεραιότητα του job επιβραβεύονται, ώστε να προσεγγίζεται η συμμόρφωση SLA χωρίς σκληρούς κανόνες.

Παραμετροποίηση: Όλες οι σταθμίσεις της συνάρτησης ανταμοιβής ελέγχονται μέσω της παρακάτω ρυθμιζόμενης δομής:

```
"reward_penalty_config": {
  "base_reward": 50,
  "cost_weight": 2,
  "latency_weight": 2,
  "placement_reward": 400,
  "placement_penalty": 100,
  "utilization_bonus": 300,
  "over_utilization_penalty": 200,
  "energy_weight": 2
}
```

Αυτή η σχεδίαση προσφέρει ευελιξία για την ισορροπία μεταξύ κόστους, καθυστέρησης, ενεργειακής απόδοσης και αξιοποίησης πόρων κατά την εκπαίδευση και αξιολόγηση του αλγορίθμου ενίσχυσης.

Προσέγγιση Πολλών Πρακτόρων

Το προηγούμενο περιβάλλον μονού πράκτορα επεκτύνεται σε μια εκδοχή πολλών πρακτόρων, με σκοπό την αξιολόγηση της κλιμάκωσης, της γενίκευσης και του συντονισμού σε κατανεμημένα σενάρια λήψης αποφάσεων. Η βασική ιδέα έγκειται στη μοντελοποίηση πολλαπλών πυλών (gateways) ως ανεξάρτητων πρακτόρων, οι οποίοι λειτουργούν ταυτόχρονα πάνω σε κοινή υποδομή και μοιράζονται τους διαθέσιμους πόρους του συστήματος. Κάθε πύλη διαθέτει δική της ουρά εισερχόμενων εργασιών και λαμβάνει αποφάσεις εκχώρησης με βάση τοπική παρατήρηση, χωρίς άμεση επικοινωνία με τους υπόλοιπους πράκτορες. Το σημαντικό χαρακτηριστικό αυτής της προσέγγισης είναι ότι όλοι οι πράκτορες επαναχρησιμοποιούν την ίδια πολιτική που έχει μάθει ο αρχικός DQN πράκτορας κατά την εκπαίδευση στο περιβάλλον ενός πράκτορα. Με αυτόν τον τρόπο, αποφεύγεται η ανάγκη εκ νέου εκπαίδευσης σε επίπεδο πολλών πρακτόρων και μειώνεται σημαντικά το υπολογιστικό κόστος. Η κατανομή των εργασιών στους πράκτορες γίνεται με στρατηγική κυκλικής εναλλαγής (round-robin), ώστε να διασφαλίζεται ισότιμη κατανομή φορτίου μεταξύ των πυλών. Καθώς οι πράκτορες λειτουργούν πάνω σε κοινή φυσική υποδομή, η κατανομή από έναν πράκτορα επηρεάζει τους διαθέσιμους πόρους για τους υπόλοιπους, γεγονός που δημιουργεί δυναμική αλληλεπίδραση και έμμεσο ανταγωνισμό. Το περιβάλλον παρακολουθεί ανεξάρτητα τη λήξη εργασιών κάθε πράκτορα και συλλέγει ξεχωριστά στατιστικά απόδοσης, όπως συνολική ανταμοιβή, ποσοστό επιτυχίας και κατανομή ενεργειών. Η χρήση κοινής πολιτικής από όλους τους πράκτορες επιτρέπει την αξιολόγηση της ικανότητας γενίκευσης της στρατηγικής, καθώς και του βαθμού προσαρμοστικότητας σε πολύπλοκα, μη-συνεργατικά

περιβάλλοντα. Παράλληλα, εξετάζεται η ευρωστία της πολιτικής όταν αυτή εφαρμόζεται σε διαφορετικές πύλες με ετερογενή πρότυπα φόρτου, καθώς και ο τρόπος με τον οποίο οι πράκτορες ανταγωνίζονται ή συνεργάζονται εμμέσως μέσω της κοινής χρήσης πόρων. Το τελικό σύστημα επιτρέπει την ταυτόχρονη εκτέλεση χιλιάδων εργασιών από πολλαπλές ανεξάρτητες πύλες δικτύου, εξομοιώνοντας ένα ρεαλιστικό σενάριο edge-cloud υποδομής με κατανεμημένους ενορχηστρωτές. Η προσέγγιση αυτή ανοίγει τον δρόμο για μελλοντική μελέτη κατανομής αρμοδιοτήτων, συνεργατικής μάθησης και μηχανισμών διαμεσολάβησης μεταξύ πρακτόρων σε μεγάλης κλίμακας έξυπνα συστήματα.

1.4.3 Πειράματα

Πείραμα 1: Πειραματική Σύγκριση LSTM και Transformer στην Πρόβλεψη Φορτίου

Προκειμένου να αξιολογηθεί η αποδοτικότητα προηγμένων μοντέλων βαθιάς μάθησης για την πρόβλεψη βραχυπρόθεσμου φόρτου εργασιών σε cloud υποδομές, διεξήχθη συγκριτικό πείραμα μεταξύ δύο σύγχρονων αρχιτεκτονικών: των επαναλαμβανόμενων δικτύων τύπου Μακράς Βραχύχρονης Μνήμης και των Transformer. Και τα δύο μοντέλα εκπαιδεύτηκαν σε ιστορικά δεδομένα τηλεμετρίας από το καθαρισμένο και κανονικοποιημένο Alibaba Cluster Trace dataset, με στόχο την πρόβλεψη της χρήσης επεξεργαστικής ισχύς και της διάρκειας εκτέλεσης εργασιών σε μελλοντικά χρονικά παράθυρα. Η εκπαίδευση πραγματοποιήθηκε με ενοποιημένες ρυθμίσεις: σταθερό μήκος εισόδου 60 βημάτων, κοινό loss function (MSE), ίδιο batch size και early stopping. Η αξιολόγηση έγινε μέσω των μετρικών RMSE, MAE και R^2, ενώ πραγματοποιήθηκαν δέκα ανεξάρτητες δοκιμές για κάθε μοντέλο.

Table 1.5: Συνοπτική Απόδοση LSTM και Transformer σε Δοκιμές Πρόβλεψης

Μετρική	**Μοντέλο**	**Μέση Τιμή**	**Τυπ. Απόκλ.**	**Καλύτερη**	**Χειρότερη**
RMSE	LSTM	0.0850	0.0020	0.0823	0.0889
RMSE	Transformer	0.0820	0.0020	0.0795	0.0856
MAE	LSTM	0.0650	0.0012	0.0635	0.0673
MAE	Transformer	0.0630	0.0011	0.0615	0.0649
R^2	LSTM	0.8190	0.0025	0.8234	0.8129
R^2	Transformer	0.8500	0.0008	0.8513	0.8485

Το Σχήμα 1.4.1 απεικονίζει τη σύγκριση των μοντέλων, παρουσιάζοντας την εξέλιξη των μετρικών σε κάθε επανάληψη (άνω σειρά), καθώς και τη συνολική μέση απόδοση και σταθερότητα (κάτω σειρά).

Συμπεράσματα. Το Transformer υπερείχε σε όλες τις μετρικές, εμφανίζοντας σταθερότερη και πιο ακριβή συμπεριφορά. Η αρχιτεκτονική προσοχής που χρησιμοποιεί επέτρεψε την ανίχνευση πολύπλοκων χρονικών σχέσεων, καθιστώντας τον ιδανικό για πρόβλεψη φόρτου σε δυναμικά περιβάλλοντα cloud. Αν και οι βελτιώσεις είναι οριακές (3–4%), η πρακτική τους σημασία σε περιβάλλοντα μεγάλης κλίμακας είναι σημαντική, οδηγώντας σε βελτιωμένη αξιοποίηση πόρων και μείωση κόστους. Το LSTM εξακολουθεί να αποτελεί αξιόπιστη επιλογή, ωστόσο το Transformer παρουσιάζει ανώτερη συνέπεια και απόδοση, ιδίως όταν απαιτείται ακρίβεια και σταθερότητα στην πρόβλεψη για αυτοματοποιημένο scaling ή κατανομή εργασιών σε edge–cloud υποδομές.

Πείραμα 2: Αξιολόγηση Πολυπρακτορικού DQN για Κατανομή Πόρων

Το δεύτερο πείραμα αξιολόγησε την απόδοση πολυπρακτορικού συστήματος DQN για κατανομή 50,000 εργασιών σε αρχιτεκτονική τριών στρωμάτων (near edge, far edge, cloud)), με τρεις ανεξάρτητους πράκτορες (gateways) που λαμβάνουν αποφάσεις χωρίς επικοινωνία. Το μοντέλο χρησιμοποίησε κοινή πολιτική ενισχυτικής μάθησης, εκπαιδευμένη σε μονοπρακτορικό περιβάλλον. Το σύστημα πέτυχε ποσοστό επιτυχίας 100%, χωρίς αποτυχίες ή υπερδέσμευση πόρων. Εργασίες υψηλής προτεραιότητας κατανεμήθηκαν κυρίως στο near και far-edge, ενώ εργασίες χαμηλής προτεραιότητας τοποθετήθηκαν ευκαιριακά στο edge, εφόσον υπήρχε διαθέσιμη χωρητικότητα. Η συμπεριφορά αυτή αντανακλά την ικανότητα του DQN να ισορροπεί ανάμεσα στην άμεση διαθεσιμότητα και τις απαιτήσεις ποιότητας υπηρεσίας. Η στρατηγική κατανομής ανέδειξε προτίμηση για το edge **2.7**× μεταξύ της Προτεραιότητας 1 (39.8%) και της Προτεραιότητας 3 (14.6%), όπως φαίνεται στο Διάγραμμα 1.4.2(a). Ενδιαφέρον προκαλεί η τοποθέτηση ορισμένων κρίσιμων εργασιών στο cloud, καθώς και αντίστροφα η ανάθεση εργασιών

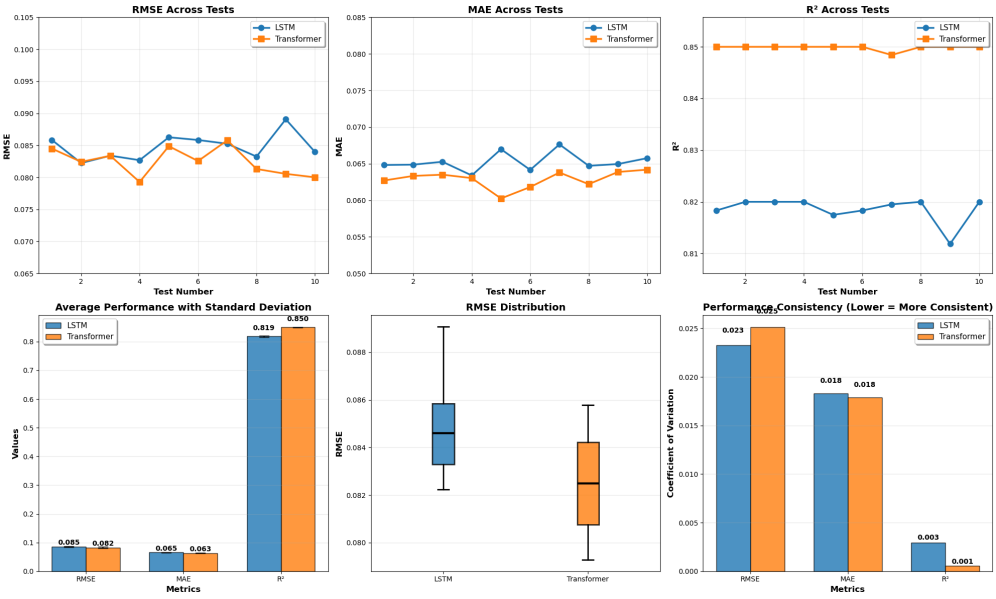

Figure 1.4.1: Σύγκριση LSTM και Transformer σε δέκα δοκιμές. Άνω σειρά: τιμές RMSE, MAE, R^2 ανά επανάληψη. Κάτω σειρά: μέση απόδοση με δείκτες διακύμανσης και ανάλυση σταθερότητας.

χαμηλής προτεραιότητας στο near-edge. Αυτή η συμπεριφορά δεν αποτελεί σφάλμα, αλλά προσαρμοστική ευφυΐα: το σύστημα αξιοποιεί διαθέσιμους πόρους αποδοτικά, προσαρμόζοντας την κατανομή σε πραγματικό χρόνο με βάση τη διαθεσιμότητα.

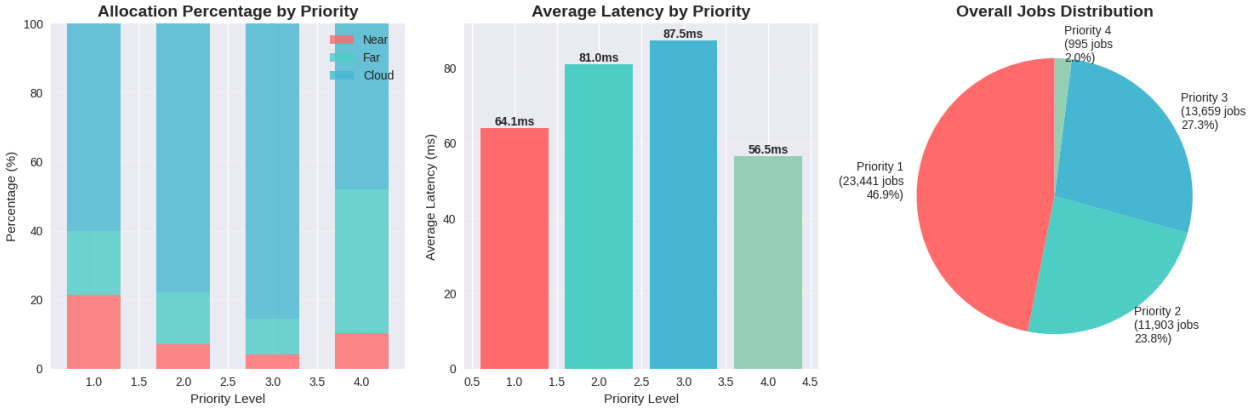

Figure 1.4.2: Ανάλυση κατανομής πόρων και καθυστέρησης ανά προτεραιότητα: (a) κατανομή ανά υποδομή, (b) καθυστέρηση ανά επίπεδο προτεραιότητας, (c) συνολική κατανομή εργασιών.

Ανάλυση Αξιοποίησης Πόρων. Η Εικόνα 1.4.3 απεικονίζει τη χρονική εξέλιξη της χρήσης CPU στα τρία επίπεδα της υποδομής (Near Edge, Far Edge και Cloud) κατά την εκτέλεση του πολυπρακτορικού DQN. Παρατηρείται ότι το near-edge διατηρεί σταθερά υψηλή αξιοποίηση (άνω του 70%), λειτουργώντας ως κύριος φορέας επεξεργασίας. Το far-edge παρουσιάζει παρόμοια εικόνα, ενώ το cloud ενεργοποιείται μόνο παροδικά, κυρίως όταν τα περιφερειακά επίπεδα φτάνουν σε κορεσμό. Αυτή η συμπεριφορά αντανακλά στρατηγική εκμετάλλευση των διαθέσιμων πόρων, όπου η πολιτική μάθησης προτιμά τοπική επεξεργασία για χαμηλό latency, ενώ διατηρεί το cloud ως buffer σε περιόδους αιχμής.

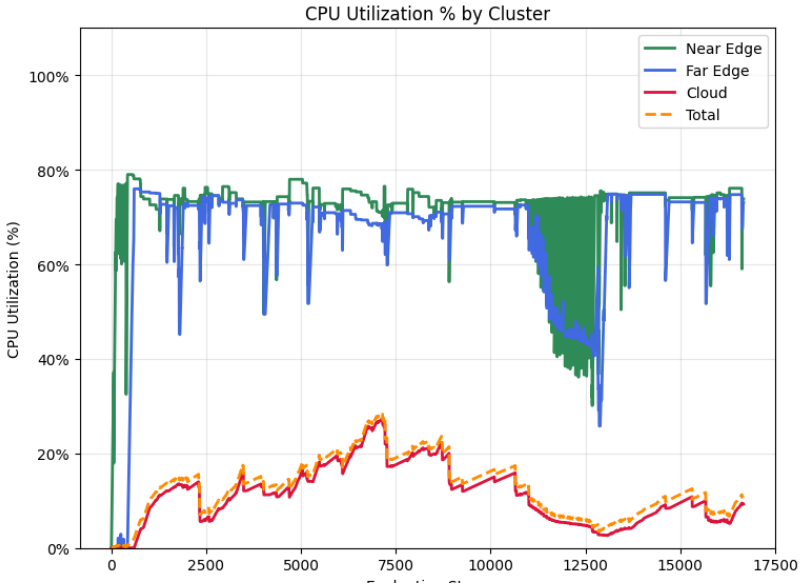

Figure 1.4.3: Χρονική εξέλιξη της χρήσης CPU στα τρία επίπεδα (Near, Far, Cloud). Η αξιοποίηση στο near-edge παραμένει σταθερή, ενώ το cloud χρησιμοποιείται κυρίως ως buffer.

Αξιοσημείωτη είναι επίσης η **προσαρμοστική συμπεριφορά** του συστήματος: παρά την απουσία επικοινωνίας, οι πράκτορες απέδωσαν ισοδύναμα, με ελάχιστη διακύμανση απόδοσης, υποδεικνύοντας επιτυχή γενίκευση της κοινής πολιτικής.Η ενεργειακή απόδοση ανήλθε σε 1,246,625 μονάδες ανταμοιβής ανά kWh, αποδεικνύοντας ότι η έξυπνη κατανομή μπορεί να ικανοποιήσει ταυτόχρονα στόχους απόδοσης και βιωσιμότητας. Παράλληλα, το συνολικό κόστος επεξεργασίας για 50,000 εργασίες διαμορφώθηκε μόλις στα $22.61 (0.0005$/εργασία), επιβεβαιώνοντας την εξαιρετική οικονομική αποδοτικότητα του συστήματος.

Πείραμα 3: Σύγκριση Μονοπρακτορικού και Πολυπρακτορικού DQN

Το Πείραμα 3 αξιολογεί την απόδοση του πολυπρακτορικού DQN συστήματος σε σχέση με τη μονοπρακτορική του εκδοχή, σε κοινό σύνολο δεδομένων 50,000 εργασιών και κοινή υποδομή. Το μονοπρακτορικό σύστημα λαμβάνει αποφάσεις σε κεντρικό επίπεδο, ενώ το πολυπρακτορικό βασίζεται σε τρεις ανεξάρτητους agents χωρίς μεταξύ τους επικοινωνία.

Και οι δύο προσεγγίσεις πέτυχαν **100% επιτυχία στην εκχώρηση** χωρίς αποτυχίες ή υπερκατανομές. Η κύρια διαφορά εντοπίζεται στην αποδοτικότητα: το πολυπρακτορικό μοντέλο απαιτεί **67% λιγότερα βήματα επεξεργασίας**,ως αναμένετο, και προσφέρει **2% υψηλότερη μέση ανταμοιβή**.

Table 1.6: Συνοπτική Σύγκριση Απόδοσης Μονοπρακτορικού και Πολυπρακτορικού Συστήματος

Μετρική	**Single-Agent**	**Multi-Agent**	**Διαφορά**
Βήματα Επεξεργασίας	50,000	16,667	-67%
Μέση Ανταμοιβή/Εργασία	109.73	111.96	+2.0%
Κόστος/Εργασία	$0.0005	$0.0005	0%
Κατανάλωση Ενέργειας/Εργασία	0.0001 kWh	0.0001 kWh	0%

Η κατανομή με βάση την προτεραιότητας και το ποσοστό επιτυχίας ήταν πανομοιότυπη, επιβεβαιώνοντας ότι και οι δύο προσεγγίσεις διαχειρίζονται εξίσου αποτελεσματικά κρίσιμες εργασίες:

Table 1.7: Κατανομή Προτεραιοτήτων και Επιτυχία

Προτεραιότητα	**Πλήθος Εργασιών**	**Edge Alloc. (SA)**	**Edge Alloc. (MA)**	**Επιτυχία**
1 (Κρίσιμη)	23,439	40.3%	40.3%	100%
2 (Υψηλή)	11,905	22.1%	22.1%	100%
3 (Μέση)	13,659	15.9%	15.9%	100%
4 (Χαμηλή)	995	51.1%	50.9%	100%
5 (Best-effort)	2	50.0%	50.0%	100%

Στην αξιοποίηση CPU, οι διαφορές είναι αμελητέες:

Table 1.8: Αξιοποίηση Πόρων

Υποδομή	**Single-Agent**	**Multi-Agent**	**Διαφορά**
Near Edge (Μέσος Όρος)	72.0% ± 6.7%	72.2% ± 7.0%	+0.2pp
Far Edge (Μέσος Όρος)	66.7% ± 13.9%	66.8% ± 14.0%	+0.1pp
Cloud (Μέσος Όρος)	11.3% ± 6.1%	11.3% ± 6.1%	0pp

Τέλος, σε οικονομικούς και περιβαλλοντικούς δείκτες, και τα δύο μοντέλα πέτυχαν ταυτόσημο συνολικό κόστος και κατανάλωση, με το πολυπρακτορικό να υπερτερεί ελαφρώς σε απόδοση ως προς κόστος και ενέργεια:

Table 1.9: Αποδοτικότητα Κόστους και Ενέργειας

Δείκτης	**Single-Agent**	**Multi-Agent**
Ανταμοιβή ανά Δολάριο	242,816	247,648
Ανταμοιβή ανά kWh	1,269,074	1,294,781

Συμπερασματικά, το πολυπρακτορικό DQN παρέχει ίδιας ποιότητας κατανομή με σημαντικά καλύτερη επεκτασιμότητα και απόδοση μέσω παραλληλισμού, καθιστώντας το κατάλληλο για δυναμικά περιβάλλοντα edge–cloud.

Πείραμα 4: Απόδοση Πολυπρακτορικού DQN με Συνθετικά Φορτία Transformer

Το Πείραμα 4 αξιολογεί τη λειτουργία του πολυπρακτορικού DQN υπό φορτία εργασίας που παράχθηκαν με τον αλγόριθμο πρόβλεψης Transformer. Ο στόχος είναι η εκτίμηση της ευρωστίας του συστήματος και της προσαρμοστικότητας του σε δυναμικά, στατιστικά διαφοροποιημένα φορτία. Το Transformer εκπαιδεύτηκε σε 500,000 αρχικές εργασίες και παρήγαγε ένα νέο σύνολο 42,156 εργασιών, με μέση τιμή σφάλματος $R^2 = 0.850$. Οι προβλέψεις του οδήγησαν σε 21% αύξηση της μέσης απαίτησης CPU (2.24 έναντι 1.85) και 5% μείωση στη διάρκεια εκτέλεσης κατά μέσο όρο, όπως φανερώνουν και τα διαγράμματα 1.4.4, προσφέροντας πιο ρεαλιστικές και ποικίλες ροές εργασιών.

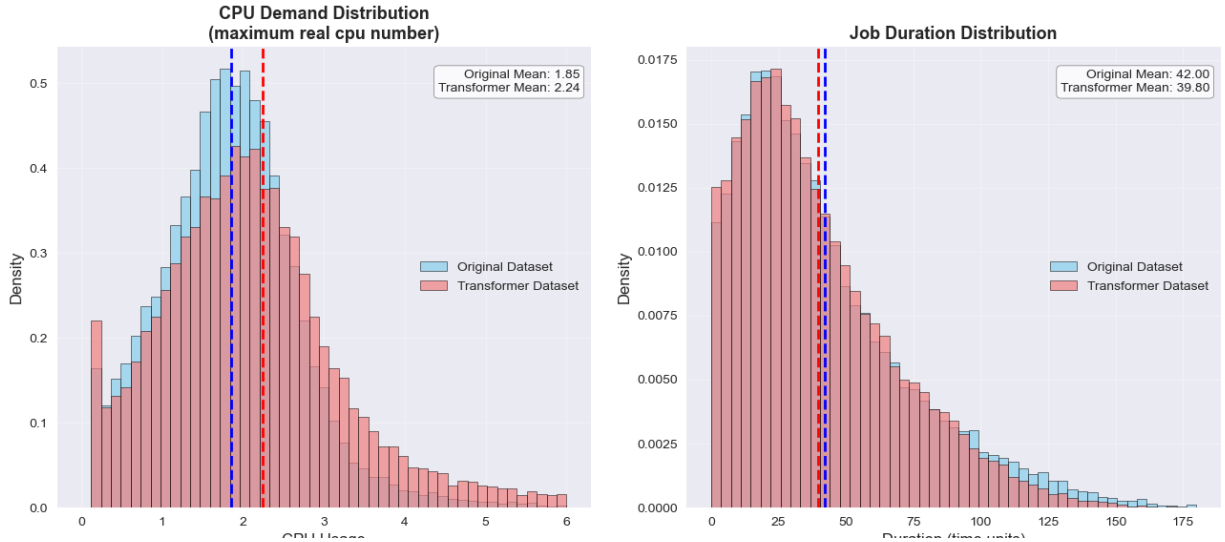

Figure 1.4.4: Κατανομή απαιτήσεων CPU (αριστερά) και διάρκειας (δεξιά) για το συνθετικό φορτίο Transformer σε σχέση με το αρχικό.

Η κατανομή προτεραιοτήτων επίσης διατηρήθηκε με μικρές αποκλίσεις: οι κρίσιμες εργασίες (Priority 1) μειώθηκαν στο 43.2%, ενώ οι Priority 2 αυξήθηκαν στο 28.9%.

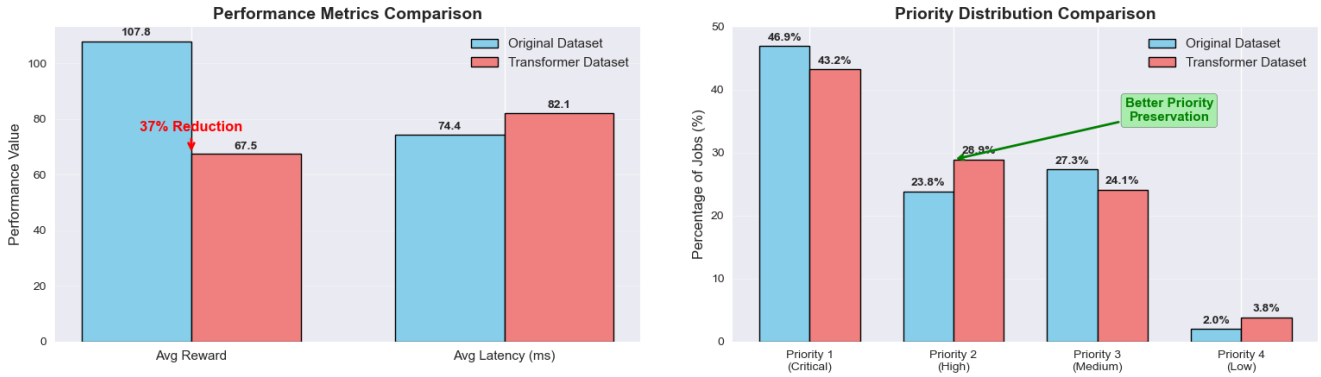

Figure 1.4.5: Ανάλυση ανταμοιβής και καθυστέρησης (αριστερά) και πιστότητα προτεραιοτήτων (δεξιά) για το φορτίο Transformer.

Παρά την αυξημένη πολυπλοκότητα, η μέση ανταμοιβή μειώθηκε μόνο κατά 37% (67.45 έναντι 107.78) και η καθυστέρηση αυξήθηκε κατά 10%, επιβεβαιώνοντας την ανθεκτικότητα του συστήματος σε αντίξοες συνθήκες. Η αξιοποίηση πόρων παρέμεινε υψηλή και ισορροπημένη,όπως φαίνεται στο σχήμα 1.4.6:

Τέλος, η συχνότητα αλλαγών πολιτικής ανά 100 βήματα μειώθηκε σημαντικά (από 290–300 σε 210–240), ενισχύοντας τη σταθερότητα, όπως φαίνεται στο σχήμα 1.4.7:

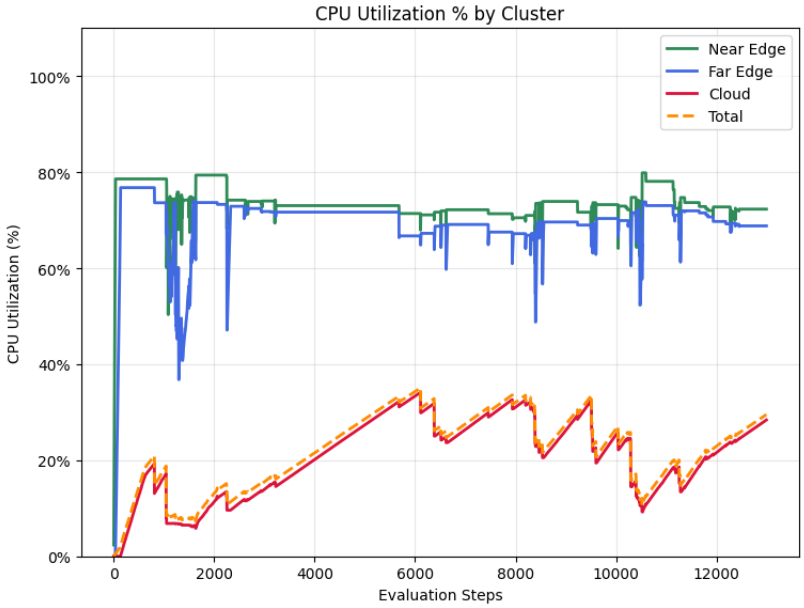

Figure 1.4.6: Αξιοποίηση CPU ανά επίπεδο υποδομής κατά την εκτέλεση φορτίου Transformer.

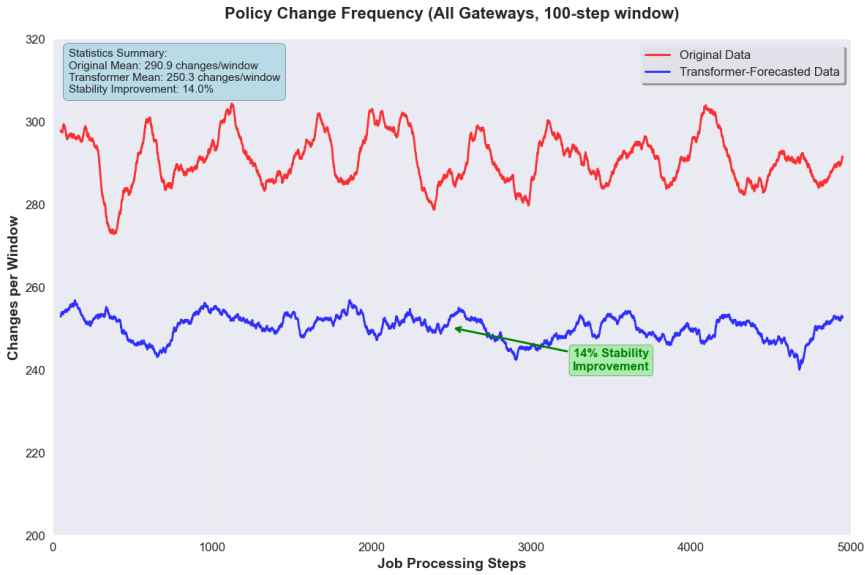

Figure 1.4.7: Συγκριτική συχνότητα αλλαγών πολιτικής για αρχικό και συνθετικό φορτίο Transformer.

Συμπερασματικά, το συνθετικό φορτίο Transformer επιτρέπει απαιτητικότερη αξιολόγηση, αποκαλύπτοντας αυξημένη σταθερότητα και προσαρμοστικότητα του πολυπρακτορικού DQN. Η μείωση στην απόδοση αντανακλούν την αυξημένη πολυπλοκότητα του φορτίου και όχι αδυναμία του αλγορίθμου, ο οποίος αποδείχθηκε ευέλικτος και ανθεκτικός.

1.5 Συμπεράσματα

Συμπεράσματα και Προοπτικές Εξέλιξης

Η παρούσα εργασία προσέφερε μια ολοκληρωμένη μελέτη συστημάτων ευφυούς κατανομής πόρων σε υποδομές edge–cloud, αξιοποιώντας τεχνολογίες βαθιά ενισχυτική μάθηση σε πολυπρακτορικά περιβάλλοντα καθώς και

μοντέλα πρόβλεψης βασισμένα σε LSTM και Transformer αρχιτεκτονικές.

Τα αποτελέσματα του πρώτου πειράματος κατέδειξαν την υπεροχή του Transformer σε σχέση με το LSTM όσον αφορά την ακρίβεια πρόβλεψης (με $R^2 = 0.850$ έναντι 0.819), τη σταθερότητα αποτελεσμάτων και τη χαμηλότερη ευαισθησία σε διαφορετικές υποομάδες δεδομένων. Η αρχιτεκτονική προσοχής (attention mechanism) του Transformer επέτρεψε την αποδοτικότερη κατανόηση μακροπρόθεσμων χρονικών εξαρτήσεων στα δεδομένα τηλεμετρίας, ενισχύοντας τη χρήση του ως βασικό συστατικό σε προληπτικούς αλγόριθμους αυτο-κλιμάκωσης. Το πολυπρακτορικό σύστημα DQN που μελετήθηκε στη συνέχεια αποδείχθηκε εξαιρετικά ικανό στην κατανομή εργασιών σε διαφορετικά επίπεδα υποδομής (near-edge, far-edge, cloud), με πλήρη επιτυχία κατανομής (100%). Οι πράκτορες υιοθέτησαν στρατηγικές διαφοροποίησης με βάση την προτεραιότητα εργασιών, ευνοώντας την τοποθέτηση εργασιών υψηλής προτεραιότητας σε κόμβους με χαμηλό latency, ενώ επέδειξαν και ευφυή προσαρμογή σε περιόδους χαμηλού φορτίου όπου εκμεταλλεύτηκαν διαθέσιμους edge πόρους για εργασίες χαμηλότερης προτεραιότητας. Επιπλέον, το σύστημα παρουσίασε δυνατότητα προσαρμογής των στόχων βελτιστοποίησης μέσω αλλαγής των βαρών ανταμοιβής (για κόστος, ενέργεια, latency), επιτρέποντας την υιοθέτηση στρατηγικών ανάλογα με τις προτεραιότητες εφαρμογής (π.χ. βιωσιμότητα, κόστος, απόκριση). Η τέταρτη δοκιμή ανέδειξε τη γενίκευση και σταθερότητα του συστήματος όταν εφαρμόζεται σε συνθετικά workloads που παρήγαγε το Transformer. Παρά την αύξηση της πολυπλοκότητας (π.χ. 21% υψηλότερη απαίτηση σε CPU), το σύστημα διατήρησε πλήρη αξιοπιστία, ενώ η σταθερότητα της πολιτικής βελτιώθηκε κατά 20–25%. Αυτή η ανθεκτικότητα ενισχύει τη χρήση συνθετικών δεδομένων ως αξιόπιστη μέθοδο ελέγχου και προσομοίωσης σε πραγματικές συνθήκες.

Προτεινόμενες Μελλοντικές Επεκτάσεις

Οι μελλοντικές κατευθύνσεις περιλαμβάνουν:

- Την ενσωμάτωση συνεργατικής πολυπρακτορικής μάθησης (MARL) με δυνατότητες συντονισμού και επικοινωνίας μεταξύ πρακτόρων.
- Την υλοποίηση αλγορίθμων ανθεκτικών σε βλάβες, επιθέσεις ή αβεβαιότητα, μέσω μηχανισμών ανίχνευσης ανωμαλιών ή ενσωμάτωσης γνώσης ρίσκου.
- Την ενσωμάτωση περιβαλλοντικών δεικτών στη διαδικασία κατανομής (π.χ. εκπομπές άνθρακα, χρήση ανανεώσιμων πηγών), προάγοντας την ανάπτυξη βιώσιμων και ευφυών υποδομών.

Συνολικά, η παρούσα εργασία προετοιμάζει το έδαφος για ευφυή και περιβαλλοντικά ευαίσθητα συστήματα edge–cloud, ικανά να προσαρμόζονται σε δυναμικές συνθήκες και να παρέχουν αποτελεσματική και δίκαιη διαχείριση υπολογιστικών πόρων.

Chapter 2

About this thesis

Contents

2.1 Motivation and Problem Statement

With a variety of workloads spread over heterogeneous infrastructures spanning near-edge, far-edge, and centralized cloud tiers, modern cloud-native systems function under ever-increasing complexity. These workloads differ remarkable in computational demand, execution duration, latency sensitivity, and service priority. As a result, resource allocation in such environments is no longer a matter of static provisioning but requires dynamic, context-aware decision-making.

Traditional strategies—based on static rules or reactive autoscaling—fail to capture the intricate interplay between job urgency, resource availability, and quality-of-service (QoS) constraints. Misplacing high-priority jobs often leads to SLA violations, while over-allocating cloud resources incurs unnecessary operational and energy costs. There is a growing need for intelligent scheduling frameworks capable of learning and adapting resource allocation policies that balance these trade-offs in real time.

This thesis addresses the problem of computational job placement in multi-tier cloud infrastructures using reinforcement learning (RL). The primary objective is to train agents that can dynamically allocate resources by jointly optimizing for latency, energy efficiency, execution cost, and CPU utilization, while also respecting job priorities. The system is modeled as a Markov Decision Process and deployed in a custom simulation environment that incorporates real-world job trace data and system constraints.

To reflect emerging architectural trends, we extend this framework into a multi-agent setting, where multiple gateways act as independent agents operating over shared resources. Each agent handles its job queue and reuses a shared policy trained in the single-agent configuration. This distributed approach allows us to evaluate system scalability, inter-agent fairness, and policy generalization under decentralized orchestration.

Furthermore, the framework is enhanced with short-term workload forecasting. By predicting future CPU demand from past telemetry, the system can anticipate load bursts and adapt job placement decisions accordingly. This forecasting module, implemented using a Transformer model, serves as an input and is fully decoupled from the reinforcement learning process.

2.2 Short Description of the Thesis

This thesis presents an integrated framework for intelligent, scalable, and QoS-aware job placement in edge–cloud infrastructures using deep reinforcement learning. The main contributions are summarized as follows:

- Development of a custom Gymnasium-compatible simulation environment that models job placement decisions across near-edge, far-edge, and cloud clusters, incorporating resource constraints, latency tiers, priority levels, and energy–cost dynamics.
- Design of a multi-objective reward function that captures trade-offs between operational cost, SLA compliance, CPU utilization, energy consumption, and overload avoidance, providing informative feedback to the learning agent.
- Training of a DQN-based scheduler using realistic job traces. The evaluation includes priority-aware job ordering and varying load intensities.
- Extension of the RL framework to a multi-agent setting, where multiple independent agents operate sequentially on shared infrastructure. The evaluation explores scalability and policy robustness in decentralized orchestration scenarios.
- Comparative analysis of state-of-the-art time series forecasting models, specifically LSTM and Transformer architectures, to determine the most suitable model for workload prediction based on real telemetry.
- Integration of a forecasting module using Transformer-based predictors to enrich agent observations with short-term CPU usage forecasts, enhancing proactive decision-making under non-stationary workload patterns.

The final system demonstrates intelligent, adaptive behavior in job placement decisions, achieving high resource utilization and low operational cost. It also maintains strong compliance with job priority constraints and QoS guarantees in both centralized and distributed configurations.

Chapter 3

Background

Contents

3.1 Cloud Computing

3.1.1 Definition

The US National Institute of Standards and Technology (NIST) has defined cloud computing in its widely referenced technical document. According to NIST:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [48]

3.1.2 Characteristics

[28] Cloud computing operates on a service-oriented model, where users gain access to various computing resources and services without the need to manage the underlying infrastructure. This abstraction allows users to focus on leveraging the services provided, rather than dealing with the complexities of hardware and software management.

- **Service-Oriented Architecture:** Cloud computing operates on a service-oriented model, where users gain access to various computing resources and services without the need to manage the underlying infrastructure. This abstraction allows users to focus on leveraging the services provided, rather than dealing with the complexities of hardware and software management.
- **Loose Coupling:** A critical feature of cloud computing is its loose coupling, which allows flexibility and scalability. Users can access and interact with cloud services independently of the specific resources being used, providing a dynamic and adaptable environment. This flexibility enables organizations to quickly adjust to changing demands without being limited by rigid infrastructure constraints.
- **Strong Fault Tolerance:** Cloud systems are designed with robust fault tolerance mechanisms, ensuring that they can handle failures gracefully and continue operating without many disruptions. This resilience is essential for maintaining high service availability and reliability, especially for businesses that require continuous access to their data and applications.
- **Innovative Business Model:** The cloud computing business model is primarily supported by large technology companies focused on maximizing return on investment (ROI) and gaining a competitive advantage. This approach contrasts with grid computing, which is typically funded by government and academic institutions. Cloud computing's commercial viability highlights its ability to meet market demands effectively while delivering value to businesses and consumers.
- **User-Friendly Experience:** Cloud services are designed to be intuitive and user-friendly, enabling individuals with minimal technical expertise to navigate and utilize the available resources. This ease of use broadens accessibility, allowing a wider audience to benefit from cloud technologies, including those who may not have a strong IT background.
- **Pay-Per-Use Pricing Model:** A standout feature of cloud computing is its pay-per-use pricing structure, where users are billed based on the resources they consume, much like utilities such as electricity or water. This pricing model enables organizations to optimize costs by paying only for the resources they use, making cloud computing a cost-effective and efficient solution.
- **Scalability:** Cloud computing provides exceptional scalability, enabling users to scale resources up or down in response to fluctuating demands. Whether an organization needs additional resources during peak usage times or to scale back during quieter periods, cloud services offer seamless adaptability, ensuring that performance remains optimal at all times.
- **Virtualization Technology:** Virtualization is at the core of cloud computing, allowing multiple virtual instances to run on a single physical server. This technology enhances resource utilization and provides flexibility in workload management. By using virtualization, cloud providers can offer a diverse range of services while maintaining efficiency and reducing operational costs.

- **High Reliability:** Cloud platforms are engineered for high reliability, ensuring users can access their applications and data whenever needed. This reliability is achieved through redundant systems, data replication, and a strong infrastructure backbone, all of which contribute to a dependable and uninterrupted service experience.
- **Multi-Tenancy:** Cloud computing uses a multi-tenant architecture, where multiple users share the same physical infrastructure while keeping their data and applications isolated. This setup optimizes resource utilization and enhances security by ensuring that each tenant's data remains private and protected from unauthorized access.

3.1.3 Evolution

Cloud computing, by nature, is difficult and has strict boundaries, making it challenging to pinpoint its origin. The term "cloud" became commonly used in the mid-2000s, but the concept has much deeper roots. In the 1950s, large-scale mainframe computers emerged, and users accessed these powerful systems through terminal computers that lacked computing capabilities. The earliest forms of cloud computing were closely tied to supercomputing and High-Performance Computing (HPC), and their primary applications were in scientific, financial, and academic fields.

The 1990s marked an important shift, as researchers developed large-scale computing systems that could be shared among multiple users. Telecommunication companies began to use these infrastructures to offer Virtual Private Network (VPN) services that provided lower costs and maintained good Quality of Service (QoS). This era also saw the rise of time-sharing systems, prompting the development of optimization algorithms to improve user efficiency.

In the late 2000s, the first cloud platforms were introduced, offering developers tools that simplified the management and sharing of centralized computing resources. Over time, cloud computing became a mainstream technology, evolving rapidly to meet the demands of businesses and consumers. Many of the world's leading technology companies have begun to build their cloud infrastructures, offering an expanding range of services. This shift brought about a wave of new products and services, with many widely used applications transitioning to cloud-based models.[7]

Today, cloud computing is no longer limited to researchers, developers, or those requiring high-performance computing. Nearly everyone uses cloud-based services, often without even realizing it. Tasks such as file storage, synchronization across multiple devices, and backup creation are now common functions for everyday users of personal computers, smartphones, and tablets.

3.1.4 Service Models

Cloud computing enables the provision of computational services over the Internet, as defined by the National Institute of Standards and Technology (NIST) [48]. Traditional cloud computing is categorized into three primary service models:

Infrastructure as a Service (IaaS). IaaS delivers virtualized computing resources over the internet, managed by cloud providers. These resources include storage, servers, and networking infrastructure, which are provided to clients through virtual machines. Examples of IaaS offerings include Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). IaaS offers a cost-effective way to run workloads without the need for organizations to invest in, manage, or maintain physical infrastructure. However, the responsibility for operational tasks such as resource provisioning and managing application code remains with the developers.

Software as a Service (SaaS). SaaS provides users with direct access to cloud-hosted applications, such as Microsoft 365 and Zoom. By utilizing SaaS solutions, users can leverage applications without having to manage the underlying infrastructure or operational complexities. This model simplifies usage but introduces certain trade-offs, such as reduced control over application customization and functionality.

Platform as a Service (PaaS). PaaS offers developers tools and platforms to build, deploy, and manage applications within cloud-based environments. Examples of PaaS offerings include IBM Cloud Foundry and Microsoft Azure App Service. PaaS strikes a balance between IaaS and SaaS by abstracting some of

the complexities of infrastructure management while still requiring developers to manage aspects such as configuration and certain operational tasks.

3.1.5 Advantages

Cloud computing offers a wide range of advantages that have transformed how organizations operate, scale, and innovate [6]. One of its primary benefits is **cost effectiveness**. By replacing capital expenses with operational expenses through a pay-as-you-go model, organizations, particularly small and medium-sized enterprises (SMEs), can avoid the upfront investments traditionally required for infrastructure, software, and IT personnel. This shift allows them to allocate resources more strategically toward growth and innovation.

Another key strength is **scalability and adaptability**. Cloud platforms allow businesses to dynamically scale computing resources based on real-time demand. This capability proves especially valuable during peak usage periods, such as promotional campaigns or seasonal traffic spikes, and allows organizations to reduce resources during off-peak times, optimizing both performance and cost.

In addition, **global accessibility** enhances workforce flexibility by enabling users to access data and applications from virtually any location with internet connectivity. This supports the rise of remote and hybrid work models and facilitates seamless collaboration across distributed teams. **Enhanced agility** further contributes to organizational competitiveness. Cloud services drastically reduce the time required to deploy applications and services, empowering businesses to bring new products to market faster and adapt more quickly to shifting conditions.

Improved collaboration is also supported through built-in cloud tools such as real-time editing, shared workspaces, and integrated communication features. These tools streamline workflows and reduce barriers to communication among geographically dispersed teams.

Cloud computing also relieves organizations of routine IT maintenance. **Automated maintenance and updates** are handled by service providers, who manage software patches and infrastructure upgrades, ensuring systems remain secure and current without manual intervention. Furthermore, **robust disaster recovery and backup capabilities** are built into most cloud platforms. These systems provide automated data protection against hardware failures, cyber incidents, or natural disasters, allowing businesses to minimize downtime and maintain continuity.

Advanced security measures are another major benefit. Major providers invest in cutting-edge technologies like encryption, multi-factor authentication, and continuous vulnerability monitoring, in addition to adhering to rigorous compliance standards.

A further advantage is **access to cutting-edge technologies**. Cloud platforms allow organizations to harness artificial intelligence (AI), machine learning (ML), and big data analytics without the need for specialized hardware or software infrastructure. This democratization of advanced tools helps drive smarter decision-making and operational efficiency.

Cloud computing also contributes to **environmental sustainability**. By consolidating workloads in centralized, energy-efficient data centers, cloud platforms help reduce carbon emissions compared to the environmental footprint of individual enterprise data centers. Finally, the cloud **fosters innovation** by lowering the barriers to entry for experimentation and rapid prototyping. Startups and smaller firms, in particular, can leverage scalable infrastructure and advanced services to develop, test, and iterate on new ideas with minimal risk.

3.1.6 Challenges

Despite its many benefits, cloud computing faces several challenges that continue to act as barriers to broader adoption [6]. These challenges span technical, regulatory, economic, and organizational domains.

One of the most critical issues is the **dynamic and unpredictable nature of workloads**. Cloud-native applications often experience rapidly fluctuating demand, which requires systems to scale resources in real time. Static resource allocation strategies struggle to meet this variability, frequently resulting in either underutilized infrastructure or performance bottlenecks.

Security and privacy remain top concerns for IT decision-makers. Cloud environments introduce uncertainty at multiple levels, including network, application, host, and data security. Furthermore, ensuring compliance with privacy regulations—such as GDPR or HIPAA—is particularly challenging when sensitive data may be processed across multiple jurisdictions. The ambiguity around whether cloud service providers can meet these regulatory requirements raises the risk of accidental violations.

Another fundamental challenge is **connectivity and open access**. The effectiveness of cloud computing is largely dependent on the availability of reliable, high-speed internet. Like electricity in previous industrial revolutions, global internet access has the potential to unlock innovation and fuel economic growth. However, inconsistent infrastructure across regions limits the cloud's potential as a universally accessible resource.

Reliability is also a major concern, especially for modern enterprise applications that require continuous availability. Downtime, whether due to system failure or external factors, can have serious consequences. Organizations must develop comprehensive contingency plans and include reliability clauses in their service-level agreements (SLAs). High reliability often comes at increased cost, and this trade-off must be carefully assessed.

Closely related is the issue of **interoperability**. Enterprises increasingly demand seamless integration between private and public cloud environments. However, legacy systems and fast-changing business needs often complicate this goal. Although Software-as-a-Service (SaaS) solutions offer rapid deployment and cost advantages, integrating them with existing infrastructure across different platforms can be challenging. Establishing robust interoperability standards is essential for ensuring data consistency and operational integrity.

The **economic value** of cloud computing is another consideration. While cloud platforms reduce capital expenditures and offer a flexible pay-as-you-go model, hidden costs—such as those associated with disaster recovery, ongoing maintenance, and support—can erode expected savings. Additionally, service consolidation and switching providers can introduce transition costs, which must be evaluated carefully to determine true return on investment.

Organizational transformation is often necessary when adopting cloud services. As seen in previous waves of IT evolution, new technologies demand new skill sets. Cloud computing reshapes traditional IT roles, requiring professionals to focus on service integration, vendor management, and risk mitigation. Organizations must ensure that their workforce can adapt to these changes and distinguish between hype and the practical applications of cloud technologies.

Finally, **geopolitical challenges** complicate the global scalability of cloud systems. Because data can be stored or processed across international borders, compliance with regional regulations becomes complex. For example, data sovereignty laws in some countries prohibit the use of foreign-based servers. Political dynamics—such as concerns over the USA Patriot Act or ongoing debates about net neutrality—can influence the legality and accessibility of cloud infrastructure. Cloud providers have begun addressing these concerns through innovations like virtual private clouds and regional data centers, but overarching political and regulatory uncertainties remain a barrier to universal adoption.

3.1.7 Edge Computing

Edge computing represents a transformative approach to processing and managing data in modern networks by bringing computation and storage closer to the source of data generation. Unlike traditional centralized cloud computing, which relies on distant data centers, edge computing leverages distributed infrastructure positioned at the edge of the network. This architecture addresses the growing demands for low-latency, real-time processing in applications such as autonomous vehicles, drones, virtual reality, and IoT systems. Zha et al. [77] proposed the following definitions for Edge computing:

> "Edge computing is a new computing model that unifies resources that are close to the user in geographical distance or network distance to provide computing, storage, and network for application services."

Hybrid Edge-Cloud Architecture

The hybrid edge-cloud architecture combines the scalability of cloud computing with the efficiency of edge computing. As shown in Figure 3.1.1, the system is structured in three primary layers: the cloud computing layer, the boundary layer, and terminal devices.[14]

Terminal Layer The terminal layer comprises various devices connected to the edge network, including mobile devices and IoT devices such as sensors, smartphones, smart vehicles, and cameras. These devices serve dual roles as both data consumers and providers. In this layer, only the sensing capabilities of the devices are considered, rather than their computational power, to minimize service latency. Consequently, an extensive number of devices in the terminal layer gather raw data and transmit it to higher layers for storage and processing.

Boundary Layer The boundary layer, also referred to as the edge layer, serves as the core of the three-tier architecture. Situated at the edge of the network, it comprises a distributed network of edge nodes such as base stations, access points, routers, switches, and gateways. This layer enables terminal devices to connect seamlessly while managing the storage and processing of the data they upload. It also communicates with the cloud by transmitting processed data for further analysis or storage. Thanks to its proximity to end users, the edge layer is well-suited for real-time data analysis and intelligent processing, offering improved efficiency and security compared to relying exclusively on cloud computing. However, this layer is constrained by its limited resources compared to the theoretically unlimited capacity of the cloud.

Cloud Layer The cloud layer remains the central hub for data processing within the cloud-edge computing paradigm. This layer comprises high-performance servers and storage systems, providing robust computational and storage capabilities. It is particularly suited for applications requiring extensive data analysis, such as predictive maintenance and decision-making. The cloud layer serves as a permanent repository for data transmitted from the edge layer while also performing advanced analytics and processing tasks. [13]

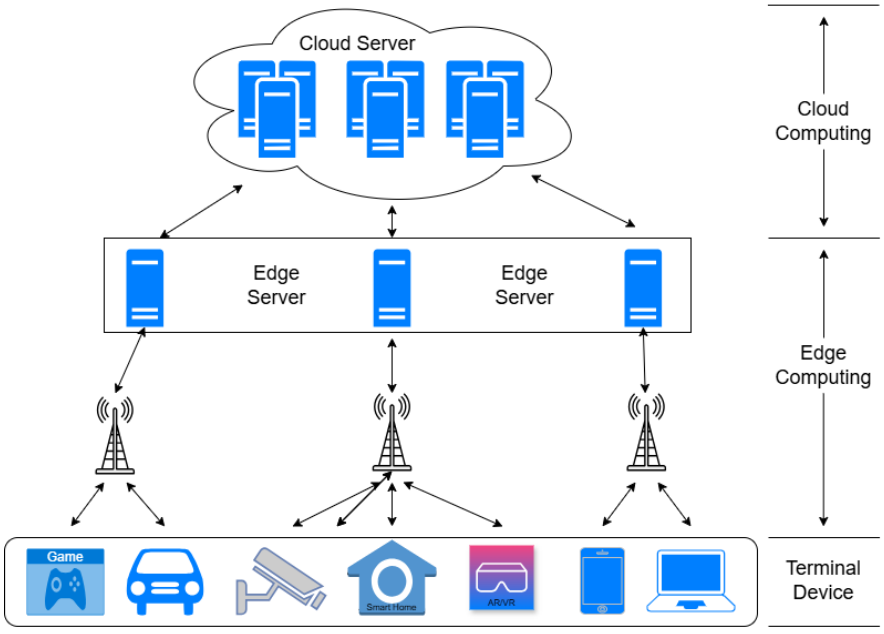

Figure 3.1.1: Edge-Cloud Computing Architecture.

Advantages of Edge Computing

Edge computing offers numerous advantages that are driving its adoption across various industries [73]. One of the most prominent benefits is **reduced latency**. By processing data closer to the source, such as IoT devices, edge computing reduces the time required for information to travel to and from centralized cloud servers. This is particularly critical for time-sensitive applications like real-time monitoring and control

systems, where even slight delays can lead to operational inefficiencies or pose safety risks.

Another advantage is **improved bandwidth efficiency**. Local data processing reduces the volume of data transmitted to the cloud, thereby alleviating network congestion and optimizing bandwidth usage. This is especially important in settings where bandwidth is constrained or data transmission costs are high.

Enhanced data privacy and security is also a key strength of edge computing. By processing sensitive data locally rather than transmitting it across networks, organizations can better protect personal or proprietary information. This approach not only helps in complying with data protection regulations but also mitigates the risk of data breaches.

In terms of operational robustness, edge computing increases **system reliability**. Local processing enables devices to function independently of the cloud, which is particularly beneficial when network connectivity is unstable or interrupted. As a result, critical applications can continue operating without disruption.

The architecture also promotes **scalability**. As the number of IoT devices grows, more edge nodes can be added to the system without burdening the centralized infrastructure. This flexibility supports the seamless expansion of distributed computing environments.

Energy efficiency is another notable advantage, particularly for battery-powered IoT devices. By offloading processing tasks to nearby edge nodes, these devices conserve energy, which is essential for deployments in remote or inaccessible areas.

Edge computing also supports **real-time data processing**, making it ideal for use cases such as autonomous vehicles, industrial automation, and smart city infrastructure. These applications require immediate responsiveness to changing conditions, which can only be achieved through localized processing.

Furthermore, the technology provides **support for diverse applications** across sectors like healthcare, manufacturing, transportation, and urban development. Its versatility enhances its attractiveness for organizations integrating IoT technologies into their operations.

From an economic perspective, edge computing contributes to **cost reduction** by decreasing the need for high-volume data transmission to the cloud and minimizing latency-related inefficiencies. This results in lower bandwidth expenses and greater overall system efficiency.

Lastly, edge computing improves **quality of service** by ensuring that performance metrics such as latency, bandwidth, and reliability are aligned with the specific demands of IoT applications. This is crucial for systems that require high availability and low response times to function effectively.

3.1.8 Latency Sensitivity Across Application Domains

Different categories of applications impose varying latency requirements, which play a critical role in resource placement and scheduling in cloud–edge systems. Understanding these latency profiles is essential for designing intelligent scheduling algorithms that prioritize QoS.

Table 3.1 summarizes typical application domains and their corresponding latency tolerance, based on a synthesis of latency-aware scheduling literature.

Table 3.1: Latency-Aware Classification of Application Domains

Priority Level	**Application Examples**	**Latency Tolerance**
Ultra-Low (P1)	VR/AR, real-time control systems	< 10 ms
Low (P2)	Online gaming, video conferencing	10–50 ms
Moderate (P3)	Web services, transactional queries	50–200 ms
High (P4)	Data syncing, telemetry monitoring	200–500 ms
Best-Effort (P5)	Backups, batch analytics, reporting	> 500 ms

These latency categories align with common deployment tiers in edge–cloud systems. For instance, ultra-low latency applications are typically executed at the near-edge to meet strict real-time constraints, while

best-effort tasks are deferred to central cloud clusters for cost efficiency [64, 67, 17].

3.2 Cloud-Native Applications

In recent years, cloud-native applications (CNAs) have emerged as a cornerstone of contemporary software development, fundamentally transforming the design and deployment of software in cloud environments. These applications are purpose-built to fully harness the benefits of cloud computing, characterized by their inherent scalability and ability to function seamlessly on elastic platforms. Cloud-native applications rely on automation and self-service mechanisms, enabling swift deployment and adaptability to fluctuating workloads. By embracing core principles like resilience, agility, and efficient resource utilization, CNAs are uniquely positioned to address the challenges of today's dynamic and rapidly evolving digital ecosystem.

3.2.1 From Monolithic to Cloud-Native Applications

The evolution from monolithic applications to CNA[32] represents a important shift in software architecture and deployment strategies. Monolithic applications are traditionally built as a single, unified unit, where all components—such as the user interface, application logic, and data access layer—are tightly integrated and interdependent. Monolithic applications rely on a centralized database, which creates a strict structure. While this approach simplifies initial implementation and deployment, it poses significant challenges in scalability, maintainability, and fault tolerance. Any change to one part of the system often requires the redeployment of the entire application, leading to potential downtime and increased operational complexity.

In contrast, cloud-native applications adopt a microservices-based architecture, as depicted in Figure 3.2.1, where each functionality is developed as an independent service. Each microservice operates autonomously, has its database, and communicates with other services through well-defined APIs. This distributed approach offers several advantages:

- **Scalability:** Individual microservices can scale horizontally based on demand without impacting other components.
- **Resilience:** Faults in one microservice are isolated, ensuring the overall system remains operational.
- **Flexibility:** Updates or modifications can be made to specific services without requiring a full system redeployment.

The diagram in Figure 3.2.1 highlights the structural differences between monolithic and cloud-native architectures. In monolithic systems, the tight coupling of components creates a single point of failure, whereas in cloud-native systems, the decoupling of microservices eliminates this risk and enhances system reliability.

This transition facilitates faster development cycles, improved fault tolerance, and optimized resource utilization. Moreover, the ability of cloud-native architectures to dynamically adapt to workload changes makes them a preferred choice in modern software engineering.

Characteristics of Microservices

The microservices architecture is defined by several foundational principles that guide its design and operational philosophy.

Independent Development and Deployment Each microservice can be built, tested, deployed, and scaled autonomously. This isolation empowers development teams to iterate rapidly and update services without introducing regressions or dependencies in other parts of the system.

Single Responsibility Principle Each microservice is responsible for a specific business capability. This modularization promotes cleaner codebases, simplifies debugging, and facilitates targeted testing strategies.

Containerization Microservices are typically encapsulated in lightweight containers that include the code, runtime, and dependencies. This packaging ensures consistent behavior across development, testing, and production environments and simplifies deployment pipelines.

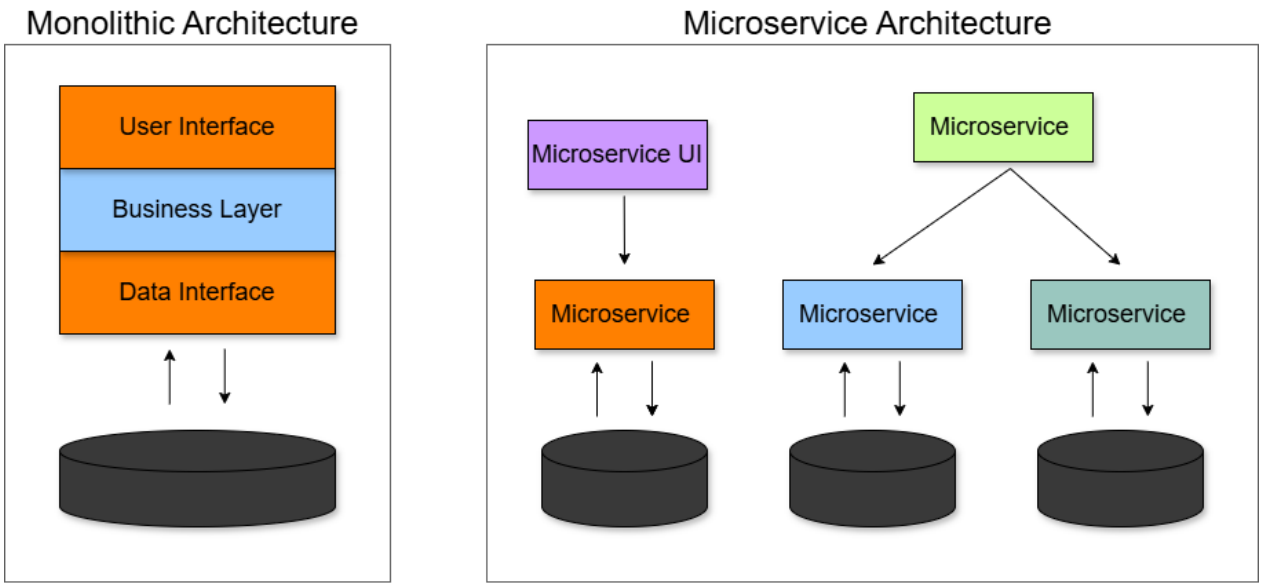

Figure 3.2.1: Structural differences between monolithic and cloud-native architectures. In monolithic systems, components are tightly coupled, relying on a centralized database. Cloud-native systems utilize a microservices-based architecture with independent services and databases.

Inter-Service Communication Communication between microservices is managed through lightweight protocols such as HTTP/HTTPS, gRPC, or WebSockets. This approach decouples services and allows for more flexible orchestration and integration.

Minimal Inter-Service Dependency Microservices are designed to be loosely coupled, interacting only with designated services when necessary. This design principle reduces the likelihood of cascading failures and increases the resilience and maintainability of the overall system.

Advantages of Microservices Architecture

The transition to a microservices-based architecture introduces several critical advantages[32]. First, it enables **autonomous lifecycle management**, as each microservice operates independently, allowing teams to update, deploy, or restart individual components without affecting the entire system. This autonomy supports real-time updates and continuous integration practices. Second, microservices offer **scalability and resource optimization**. Unlike monolithic applications that require scaling the entire system, microservices allow selective scaling based on demand. High-traffic services can be allocated additional resources independently, thereby improving cost-efficiency and system performance. Moreover microservices promote **fault isolation and resilience**. Because services are decoupled, a failure in one component is contained and does not necessarily disrupt the overall application. This design enhances system reliability and fault tolerance.

3.2.2 Network Slicing in Cloud-Native Architectures

Network slicing[59], while not exclusively tied to cloud-native architectures, is a fundamental enabler of these paradigms, particularly in the context of 5G networks. As shown in Figure 3.2.2, cloud-native architectures, such as Kubernetes clusters, provide a flexible framework for deploying and managing applications across distributed infrastructures. Similarly, network slicing facilitates the creation of multiple virtual networks on shared physical infrastructure. Each virtual network (or slice) is logically isolated, allowing for independent configuration and allocation of resources. This capability ensures optimization for specific applications, user groups, or service components such as microservices.

Key technologies, including Software-Defined Networking (SDN) and Network Functions Virtualization

(NFV), empower this flexibility by allowing for scalable, dynamic allocation of network resources. On the computational side, containerization offers similar benefits by tailoring resource allocation to each microservice.

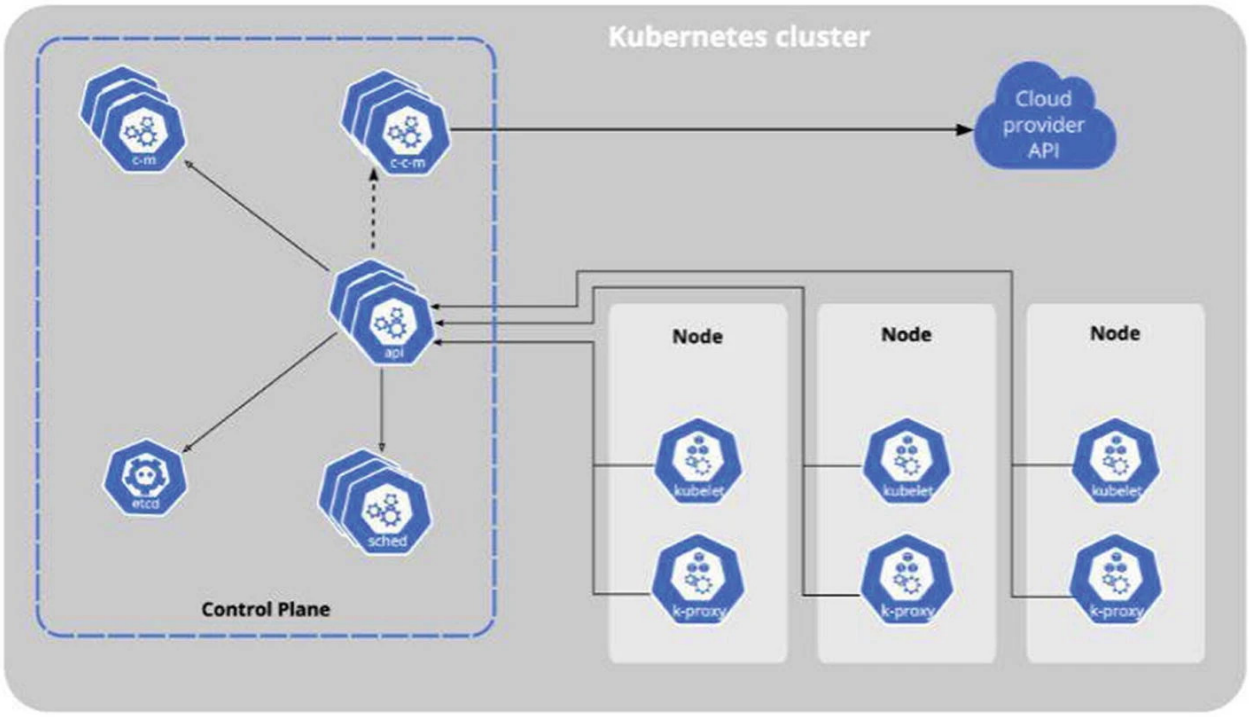

Figure 3.2.2: Kubernetes cluster architecture illustrating the control plane and worker nodes.[71]

3.2.3 Container Orchestration

The components illustrated in Figure 3.2.2 form the backbone of Kubernetes [58], one of the most widely adopted container orchestration systems. Containers, unlike traditional virtual machines, share the host's kernel, making them lightweight, portable, and quick to deploy. These features make containers an ideal choice for hosting microservices, especially in scenarios where applications need to scale dynamically or operate in distributed environments.

Microservices encapsulate their code and dependencies within container images, which can be illustrated as container instances. To handle high workloads or reduce latency, orchestrators like Kubernetes allow for deploying multiple container instances across a distributed infrastructure. The orchestration process demonstrated in Figure 3.2.2 ensures that containers operate smoothly while meeting performance and resource demands.

3.2.4 Advantages of Orchestrators

Orchestrators provide critical capabilities that simplify and strengthen the management of containerized applications [58]. One of the most remarkable advantages is **automated management**. In large-scale environments with thousands of containers, manual administration becomes unfeasible. Kubernetes addresses this challenge through its centralized Control Plane, which automates essential tasks such as container scheduling, scaling, load balancing, and health monitoring across distributed nodes.

Another key benefit is **resilience**. Orchestrators maintain high availability by detecting container failures and automatically restarting or rescheduling them to ensure continuous operation. They also dynamically scale resources based on real-time workload demands, improving system efficiency and responsiveness.

Finally, orchestrators contribute to enhanced **security**. By reducing manual intervention in deployment and scaling processes, they minimize the risk of human error and enforce consistent security policies across environments. Kubernetes, in particular, offers fine-grained access controls, network policies, and secret management features that further strengthen the platform's security posture.

3.3 Internet of Things (IoT)

With the continuous evolution of information technology, the Internet of Things (IoT) has emerged as a critical component of modern life. IoT enables interconnected devices and sensors to gather and exchange vast amounts of data via sophisticated communication networks comprising millions of nodes. This interconnected ecosystem allows IoT applications to deliver highly precise and granular network services, enhancing user experiences.[47]

As IoT adoption grows, the volume of data generated by sensors and devices demands efficient processing to extract meaningful insights and provide intelligent services for users and businesses alike. Traditional cloud computing models require this data to be transmitted to centralized servers for processing, with results subsequently relayed back to the devices. This centralized approach places substantial demands on network infrastructure, often leading to increased data transmission costs and resource constraints.

The limitations of conventional cloud architectures become even more apparent with the rise of time-sensitive IoT applications, such as smart transportation systems, smart power grids, and smart cities. These applications require rapid response times, where delays could compromise safety or operational efficiency. However, centralized cloud systems are hindered by long transmission paths, bandwidth limitations, and network congestion, resulting in latency levels that are unsuitable for real-time IoT requirements.

Moreover, the majority of IoT devices, including smart sensors, are constrained by limited energy resources. Prolonging their operational lifespan necessitates effective task scheduling, which allocates computational loads to devices with greater power reserves and processing capacity. By prioritizing localized data processing closer to end-users, both transmission delays and energy consumption can be minimized. Additionally, network traffic plays a important role in determining data transmission speed, with high congestion levels further exacerbating delays and power costs. Addressing these concerns is pivotal to ensuring the scalability and effectiveness of IoT systems across diverse applications.[73]

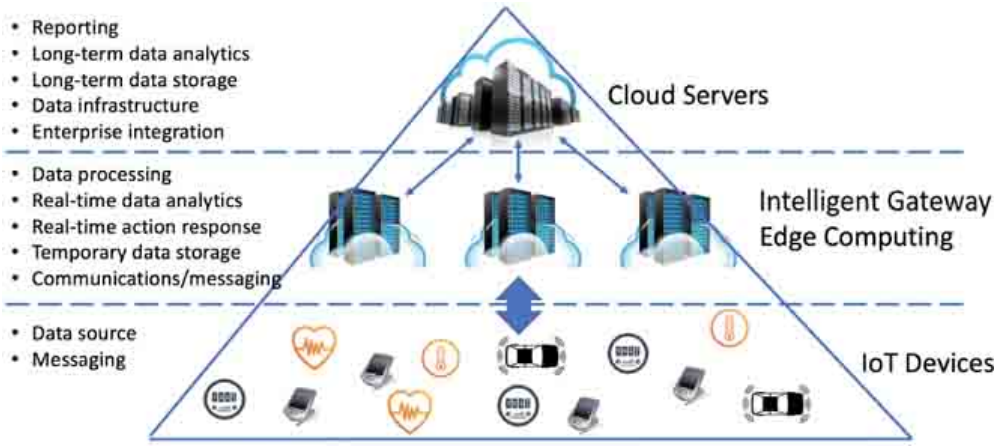

Figure 3.3.1: Illustrates the three-layer architecture of edge computing-based IoT, which consists of three layers: IoT devices, Edge Computing, and Cloud Computing. All IoT devices are end users for edge computing. In this architecture, IoT can benefit from both edge computing and cloud computing, because of the characteristics of the two structures (i.e., high computational capacity and large storage) [73]

3.3.1 Components of IoT

IoT networks are typically composed of the following key components [47]:

Sensors/Devices: Sensors serve as the fundamental building blocks of IoT, responsible for collecting and generating data that is essential for network functionality. These sensors support diverse data types and act as interactive interfaces for end-users, enabling seamless interaction with the IoT ecosystem. Furthermore, they facilitate communication across the network, ensuring that application-specific requirements are met while maintaining efficient data transmission and device interoperability.

IoT Gateways: IoT gateways act as intermediaries that link sensors with core cloud networks, facilitating smooth data flow. In addition to their role as communication bridges, they perform data pre-processing to reduce redundancy and optimize data for transmission. By aggregating sensor data and forwarding it to cloud servers for advanced processing, IoT gateways enhance system efficiency. Once the cloud servers analyze and process the data, gateways relay the results back to users, ensuring a seamless and responsive IoT environment.

Cloud/Core Network: The cloud infrastructure is central to processing, storing, and analyzing IoT data. Cloud servers handle complex computations and ensure that computing resources are readily available to support a variety of applications. This infrastructure enables large-scale data processing and real-time analytics, making it possible for IoT applications to function efficiently. Once data has been processed, the results are transmitted back to users, completing the IoT data loop and enabling real-time decision-making [73].

3.3.2 Communication Models

IoT relies on several communication models to enable interaction among devices, gateways, and cloud systems [73].

Machine-to-Machine Communication: Machine-to-machine (M2M) communication facilitates direct device-to-device interaction without requiring intermediary hardware. Devices connect over various networks, such as Bluetooth or IP-based systems, allowing seamless data exchange. This model is commonly used in smart home and automation applications, enabling efficient device coordination. However, it is often constrained by protocol compatibility, which can limit interoperability between different IoT devices.

Machine-to-Cloud Communication: In this model, IoT devices transmit data to cloud servers for storage and processing. The architecture relies on conventional network infrastructures, such as Wi-Fi, enabling devices to send vast amounts of data to centralized cloud environments. However, this model is bandwidth-sensitive, meaning that network congestion and latency can affect performance. To address these challenges, enhancements in network architecture and resource allocation are critical for optimizing communication efficiency and reducing potential bottlenecks .

Machine-to-Gateway Communication: The machine-to-gateway communication model employs application-layer gateways as intermediaries to enhance security and facilitate data translation between devices and cloud services. By offloading computational tasks to gateways, this model reduces power consumption for IoT devices, extending their operational lifespan. A common implementation of this model is found in personal health devices, where mobile phones often act as gateways, collecting and transmitting data to cloud-based health monitoring systems. This approach ensures real-time data processing while maintaining energy efficiency.

3.3.3 Impact of IoT

The adoption of IoT is driving changes across industries and daily life, reshaping the way humans interact with technology and data [73]. IoT has revolutionized transportation, healthcare, and urban living, enabling innovations such as smart cities and intelligent energy grids. In smart cities, interconnected sensors optimize traffic flow, reduce energy consumption, and improve public safety, leading to more efficient and sustainable urban environments. Similarly, in healthcare, wearable IoT devices allow for real-time health monitoring, early disease detection, and personalized treatment plans, significantly enhancing patient care and medical outcomes.

The raise of IoT has led to an exponential increase in connected devices. By 2020, the number of interconnected IoT devices was projected to reach 75 billion, far surpassing the global human population [73]. This rapid expansion has created an enormous influx of data, necessitating advancements in network infrastructure, cloud computing, and data analytics to manage and process information efficiently.

IoT is poised to become one of the most dominant sources of data, fueling innovations in big data analytics, artificial intelligence, and intelligent systems. The vast amounts of data generated by IoT devices are being leveraged to enhance predictive analytics, automation, and decision-making across multiple domains. In industries such as manufacturing and logistics, IoT-driven predictive maintenance helps reduce downtime and operational costs by detecting potential failures before they occur. Additionally, IoT-driven AI applications, such as autonomous vehicles and smart home assistants, are transforming everyday experiences by making interactions with technology more seamless and intelligent.

As IoT continues to expand, its impact will extend beyond connectivity, shaping the future of digital transformation and technological evolution. The convergence of IoT with other emerging technologies will further drive efficiencies, enhance security, and create new opportunities for businesses and consumers alike.

3.4 Deep Learning

Artificial Intelligence (AI) [37] is the field of Computer Science dedicated to developing systems that can perform tasks requiring human-like intelligence. AI encompasses a broad spectrum of approaches, including rule-based systems, statistical methods, and machine learning techniques. At its core, AI focuses on creating intelligent agents—systems that can perceive their environment and take actions to maximize their chances of achieving specific objectives.

A major branch of AI is **Machine Learning (ML)** [55], which enables machines to learn from data and improve their performance over time without explicit programming. Researchers have been advancing Machine Learning techniques since the 1950s, leading to breakthroughs in automation, pattern recognition, and predictive modeling. Over the past few decades, the field has evolved rapidly, enabling machines to solve increasingly complex problems.

Building on these advancements, **Deep Learning (DL)** [65] has emerged as a specialized subfield of Machine Learning. As illustrated in Figure 3.4.1, Deep Learning distinguishes itself through the use of deep artificial neural networks, where "deep" refers to the presence of multiple hidden layers within the network. Unlike traditional Machine Learning, which often relies on handcrafted features, Deep Learning enables models to automatically extract hierarchical representations from raw data, enhancing their ability to recognize patterns and generalize across tasks.

Deep Learning has revolutionized AI by achieving state-of-the-art performance in areas such as computer vision, natural language processing, and reinforcement learning. The combination of large-scale datasets, enhanced computational power, and improved optimization techniques has made Deep Learning a dominant approach in modern AI research and applications.

Because of improvements in training methods, the availability of enormous datasets, and advances in computing power, Deep Learning has become increasingly popular. Using specialized hardware like GPUs and TPUs, modern neural networks can be trained effectively, leading to the creation of extremely complex models that can outperform humans in specific tasks. Additionally, by enabling models to learn straight from raw inputs, end-to-end learning does away with the necessity for human feature selection. Architectures like Deep Q-Networks (DQN) [49], AlexNet [38] for image classification, and Seq2Seq [62] for machine translation have all benefited greatly from this idea.

Deep Learning's capacity to use distributed representations—where each input is represented by several features, each of which may correlate to numerous inputs—is one of its main advantages. By doing this, deep models may effectively generalize even in high-dimensional regions, overcoming the curse of dimensionality. Furthermore, by avoiding overfitting and speeding up training convergence, regularization strategies like Dropout [61] and Batch Normalization [34] have improved the effectiveness and resilience of deep networks.

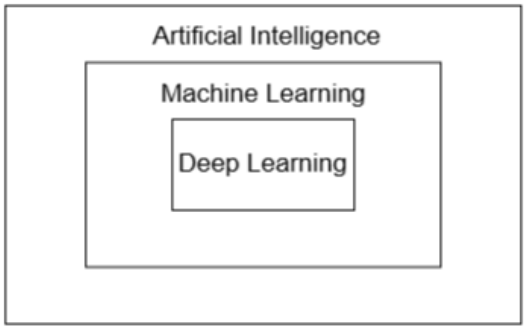

Figure 3.4.1: A hierarchical representation of Artificial Intelligence, Machine Learning, and Deep Learning.

3.4.1 Deep Learning Architecture

Deep Learning [65] contrasts with shallow learning, where conventional algorithms such as linear regression, logistic regression, support vector machines (SVMs), decision trees, and boosting operate with a single transformation between input and output layers. These methods often require manual preprocessing and feature selection before training. In contrast, Deep Learning employs multiple hidden layers between the input and output layers, allowing for a more expressive feature transformation.

A Deep Neural Network (DNN) consists of the following key components:

- **Input Layer:** Receives raw input data.
- **Hidden Layers:** Process and transform the data through weighted connections.
- **Output Layer:** Produces predictions or classifications based on learned representations.

Each unit (or **neuron**) in a layer computes a weighted sum of the outputs from the previous layer. This sum is then passed through an **activation function** to introduce non-linearity into the model. Common activation functions include the **sigmoid function**, which maps input values to a range between 0 and 1, the **tanh function**, which is similar to the sigmoid but ranges from -1 to 1, and the **Rectified Linear Unit (ReLU)**, which is the most widely used activation function in modern deep networks due to its ability to mitigate vanishing gradient issues.

Once data flows from input to output in a process known as **forward propagation**, the network evaluates its performance using a **loss function**. The **error derivatives** are then computed and propagated backward through the network using **backpropagation**. This optimization process updates the weights of the network iteratively, allowing it to learn from data effectively.

3.4.2 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a foundational class of neural network architectures designed to handle sequential data, where the order and temporal dependencies of the input are essential. Unlike traditional feedforward neural networks, which process each input independently, RNNs incorporate cyclic connections that allow the network to maintain a hidden state that captures contextual information from previous time steps. This inherent recurrence provides RNNs with a form of short-term memory, enabling them to model patterns over sequences of variable length. The idea was first formalized by Elman [74], whose work demonstrated the potential of recurrent connections for learning temporal structures in data.

The core mechanism of an RNN revolves around the recursive update of its hidden state. At each time step t, the network receives an input vector x_t and updates its hidden state h_t based on both the current input and the previous hidden state h_{t-1}. This recurrence enables the network to carry forward information across

the sequence. In its simplest form, the hidden state is updated via the equation.

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h),$$

where W_{xh} and W_{hh} are learnable weight matrices, b_h is a bias term, and ϕ is a nonlinear activation function such as tanh or ReLU. Depending on the task, an output y_t may also be generated at each time step using

$$y_t = \psi(W_{hy}h_t + b_y),$$

where W_{hy} and b_y are the output weights and biases, respectively, and ψ is typically a softmax or identity function.

This design allows RNNs to handle a wide range of tasks, including sequence classification, sequence-to-sequence learning, and time-series prediction. The parameter sharing across time steps makes RNNs parameter-efficient and well-suited for modeling sequences of arbitrary length. Furthermore, their flexibility in handling variable input and output lengths makes them ideal for tasks such as language modeling, speech recognition, and real-time sensor analysis.

However, despite their conceptual appeal, standard or "vanilla" RNNs are known to suffer from significant training challenges. One of the most well-documented issues is the vanishing or exploding gradient problem, which arises during backpropagation through time (BPTT) [74]. As the gradients are propagated backward over many time steps, they can either decay exponentially to zero or grow uncontrollably, making learning either ineffective or unstable. As a consequence, vanilla RNNs often fail to capture long-term dependencies, especially in tasks where distant inputs must influence future outputs.

In summary, Recurrent Neural Networks offer a powerful framework for modeling sequential patterns in data. Although constrained by issues related to training stability and memory limitations, they serve as the conceptual foundation for more advanced temporal modeling techniques such as LSTMs and Transformer-based architectures.

3.4.3 Long Short-Term Memory Networks (LSTM)

Long Short-Term Memory (LSTM) networks are a specialized type of RNN designed to overcome the difficulties that traditional RNNs face when learning long-range dependencies in sequential data. Standard RNNs typically struggle with vanishing or exploding gradients during training, which hinders their ability to connect events that occur far apart in a sequence. LSTMs address this problem by introducing a more sophisticated memory structure that allows information to persist across long time intervals. This architecture was first introduced by Hochreiter and Schmidhuber [31], and it remains one of the most influential contributions in the field of deep learning for sequence modeling.

At the heart of an LSTM network is the cell state, a vector that functions as an internal memory, capable of carrying information across multiple time steps. This cell state is manipulated by three learnable gates: the input gate, forget gate, and output gate. Each gate consists of a sigmoid-activated layer, which decides the extent to which information should be added, retained, or outputted at each step in the sequence. The input gate determines how much of the new incoming information should be incorporated into the cell state, while the forget gate decides which parts of the previous memory should be discarded. The output gate controls how much of the updated cell state is passed on to the next hidden state and the output of the current time step.

The internal architecture of a standard LSTM cell is illustrated in Figure 3.4.2, which highlights the flow of data through the gates and memory cell. The diagram also shows the neural operations and pointwise computations that regulate information flow, as well as the role of nonlinear activation functions such as the sigmoid and hyperbolic tangent.

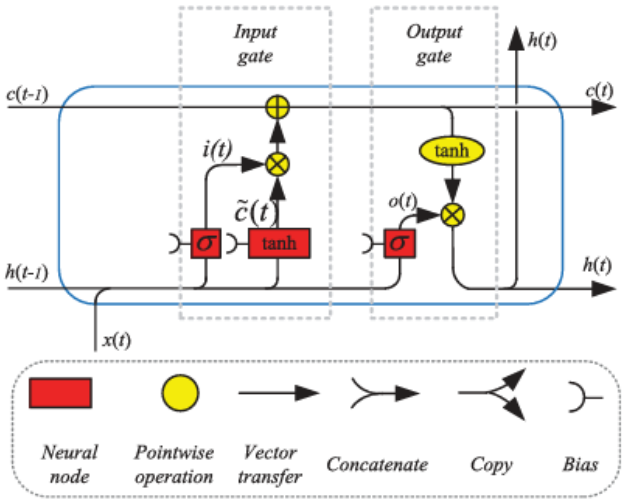

Figure 3.4.2: Standard LSTM cell architecture showing the internal gating mechanisms and information flow. [74]

The update process of an LSTM is governed by a sequence of mathematical operations. At each time step, the model receives an input vector and a hidden state from the previous time step. The forget gate evaluates what proportion of the previous cell memory should be retained. Simultaneously, the input gate evaluates how much of the current input, processed through a candidate activation, should be added to the memory. These two results are combined to produce the updated cell state. Finally, the output gate determines the next hidden state by applying a nonlinear transformation to the updated cell state and scaling it accordingly. This controlled memory update mechanism allows LSTMs to learn temporal relationships effectively, even over long sequences.

Formally, for a given input vector x_t, previous hidden state h_{t-1}, and previous cell state c_{t-1}, the operations can be described by the following equations:

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\
\tilde{c}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
$$

where σ is the sigmoid activation function, tanh is the hyperbolic tangent function, and $\odot$ denotes element-wise multiplication.

LSTM networks offer several advantages that have led to their widespread adoption in sequence modeling tasks. Their ability to maintain and update long-term memory makes them particularly well-suited for applications such as speech recognition, language modeling, machine translation, and time-series forecasting. In the context of cloud computing, LSTMs have proven useful for resource usage prediction, enabling more efficient and anticipatory resource allocation in dynamic environments. Their architectural flexibility also allows for variants such as bidirectional LSTMs and deep stacked LSTMs, which further improve performance in tasks requiring nuanced sequence understanding.

3.4.4 Transformers

Transformers [45] are a class of deep learning models that have revolutionized the processing of sequential and structured data by introducing the self-attention mechanism, which allows the model to learn contextual relationships between input elements regardless of their position. Unlike traditional models such as RNNs

or Convolutional Neural Networks (CNNs), Transformers avoid sequential dependencies during training, enabling substantial gains in computational efficiency through parallelization. At the heart of the Transformer architecture lies the encoder-decoder structure, as illustrated in Figure 3.4.3. Each encoder and decoder layer is composed of a multi-head self-attention mechanism, followed by position-wise feed-forward networks, with residual connections and layer normalization applied throughout. The self-attention module allows the model to weigh the influence of different parts of the input sequence dynamically, making it particularly adept at modeling long-range dependencies. In the self-attention mechanism, each token is represented by three vectors: a query (Q), a key (K), and a value (V). The attention score between tokens is computed using scaled dot products between queries and keys, followed by a softmax function that normalizes these scores into weights. These weights are used to combine the value vectors, producing a context-sensitive representation for each token. Transformers offer many advantages over previous architectures. They enable full parallelism during training, support flexible input-output configurations, and exhibit superior ability to capture long-term dependencies without suffering from gradient vanishing or exploding issues. This architectural strength has facilitated their rapid adoption across domains beyond natural language processing, including computer vision, speech processing, and bioinformatics. Modern Transformer-based models often undergo pre-training on massive unlabeled corpora using self-supervised objectives such as masked language modeling, and are later fine-tuned for downstream tasks like translation, classification, or summarization. Variants such as Vision Transformers (ViTs), Sparse Transformers, and Mixture-of-Experts models have further extended the capabilities of the original framework.

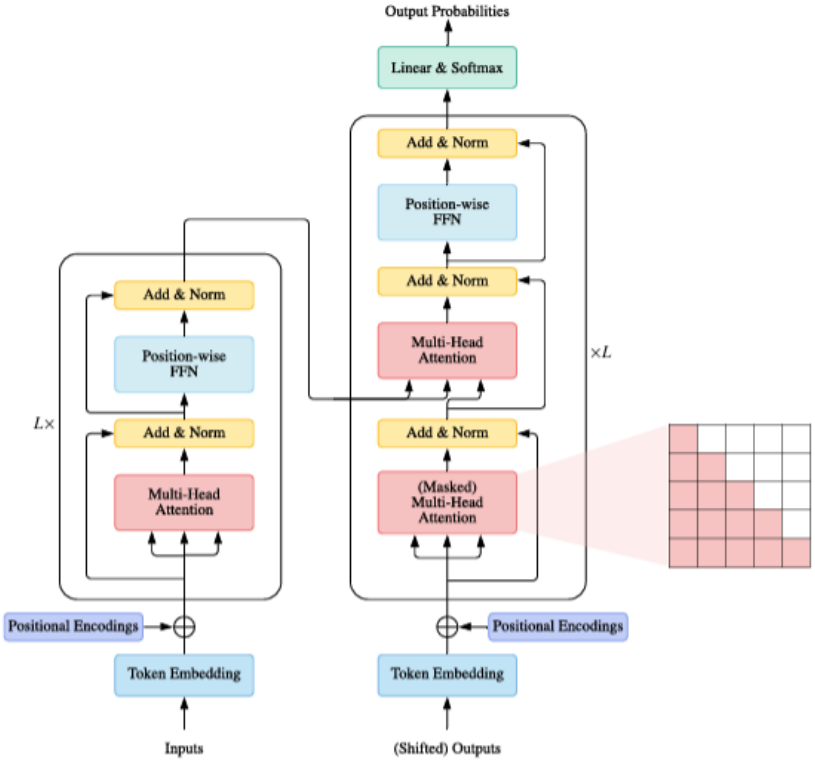

Figure 3.4.3: Architecture of the original Transformer model showing the encoder-decoder structure with multi-head attention and feed-forward layers. Adapted from Lin et al. [45].

3.5 Reinforcement Learning

Reinforcement Learning (RL) focuses on enabling agents to learn optimal behavior through interaction with a dynamic environment, guided by feedback in the form of rewards or penalties [27]. Unlike supervised learning, RL does not require labeled data or explicit instructions for achieving a task but can integrate

domain-specific knowledge or expert input when available. This versatility makes RL a powerful framework for solving a wide range of sequential decision-making problems.

3.5.1 Reinforcement Learning as a Distinct Machine Learning Category

RL is widely acknowledged as one of the three foundational approaches to Machine Learning, standing alongside Supervised Learning and Unsupervised Learning. Each of these approaches serves distinct purposes in the realm of data-driven intelligence, and RL sets itself apart by its reliance on interactive learning through trial and error in dynamic environments.

In Supervised Learning (SL), the learning algorithm is trained on a dataset comprising labeled examples, where each input is paired with a corresponding output that represents the correct answer. SL aims to infer a mapping function that generalizes accurately from the training data to unseen inputs. This approach underpins many core applications in ML, such as image recognition, natural language processing, and predictive analytics. By leveraging explicitly labeled data, SL excels in scenarios where clear supervision is feasible and sufficient data annotations exist to train high-performance models. [29]

In contrast, Unsupervised Learning (UL) operates on unlabeled data to discover inherent patterns, relationships, or latent structures within the dataset. UL is particularly valuable in exploratory analysis and applications where labeled data is scarce or unavailable. Common tasks under UL include clustering, where data points are grouped based on similarity, and dimensionality reduction, which seeks to simplify data representation while preserving essential features. Examples of UL techniques include principal component analysis (PCA) and k-means clustering, among others. [10]

Reinforcement Learning distinguishes itself from both SL and UL by its unique learning paradigm. In RL, learning does not depend on predefined labels or hidden structures within static datasets. Instead, RL centers around an agent that interacts dynamically with an environment. The agent takes action, observes the consequences in the form of new environmental states, and receives scalar rewards or penalties as feedback. This feedback mechanism enables the agent to iteratively improve its decision-making policy, aiming to maximize long-term cumulative rewards over time. Unlike SL, where the correct outcomes are explicitly provided for each input, or UL, where no feedback mechanism exists, RL offers a framework for learning optimal strategies through self-guided exploration and exploitation of the environment.[63]

This interaction-centric approach makes RL particularly well-suited for solving complex, sequential decision-making problems in which explicit supervision is either impractical or unavailable. Moreover, RL accommodates scenarios with delayed rewards, allowing the agent to optimize behaviors that yield benefits in the long run, even if immediate feedback is sparse or contradictory. By focusing on actions and their outcomes, RL addresses challenges in diverse fields, including robotics, autonomous systems, game playing, and resource allocation in dynamic systems.

3.5.2 Fundamental Elements of Reinforcement Learning

Reinforcement Learning (RL) comprises several key components that define the interaction between the agent and its environment [42]:

- **Agent**: The decision-making entity that learns and acts based on feedback from the environment.
- **Environment**: The system with which the agent interacts, providing rewards or punishments in response to actions.
- **State (s_t)**: A representation of the environment at time step t, capturing all relevant information for decision-making.
- **Action (a_t)**: A choice available to the agent at time t that influences the environment's state.
- **Reward (r_t)**: A scalar feedback signal indicating the immediate benefit or cost of an action taken by the agent.

3.5.3 Markov Decision Processes

Reinforcement Learning problems are commonly modeled as **Markov Decision Processes (MDPs)** [42], which provide a mathematical framework for sequential decision-making under uncertainty. An MDP consists of:

- A set of **states** S, representing all possible configurations of the environment.
- A set of **actions** A available to the agent.
- **Transition dynamics** $T(s_t, a_t, s_{t+1})$, denoting the probability of transitioning to state s_{t+1} given current state s_t and action a_t.
- A **reward function** $R(s_t, a_t, s_{t+1})$, quantifying the immediate reward for each transition.
- A **discount factor** $\gamma \in [0, 1]$, which balances immediate and future rewards.

The agent's goal is to learn an optimal **policy** $\pi(a_t|s_t)$, which maps states to actions to maximize the expected cumulative reward:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}$$

Where T denotes the episode length. The optimal policy π^* satisfies:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[R_t].$$

3.5.4 The Reinforcement Learning Cycle

The RL process is an iterative learning cycle [42]:

1. **Observation**: The agent perceives the current state.
2. **Action Selection**: The agent selects an action based on its policy, balancing exploration and exploitation.
3. **Environment Feedback**: The environment transitions to a new state and provides a reward.
4. **Policy Update**: The agent refines its policy using algorithms such as Q-learning or policy gradient methods.

This cycle continues until convergence or a terminal state is reached.

A key objective is to maximize the expected cumulative reward:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Where γ discounts future rewards.

Traditional RL techniques may struggle with scalability due to large state and action spaces. Deep Reinforcement Learning (DRL) techniques, such as Deep Q-Networks (DQN), leverage neural networks to approximate value functions and policies, enabling efficient learning in complex environments.

Figure 3.5.1 illustrates this cyclical interaction process between the agent and the environment.

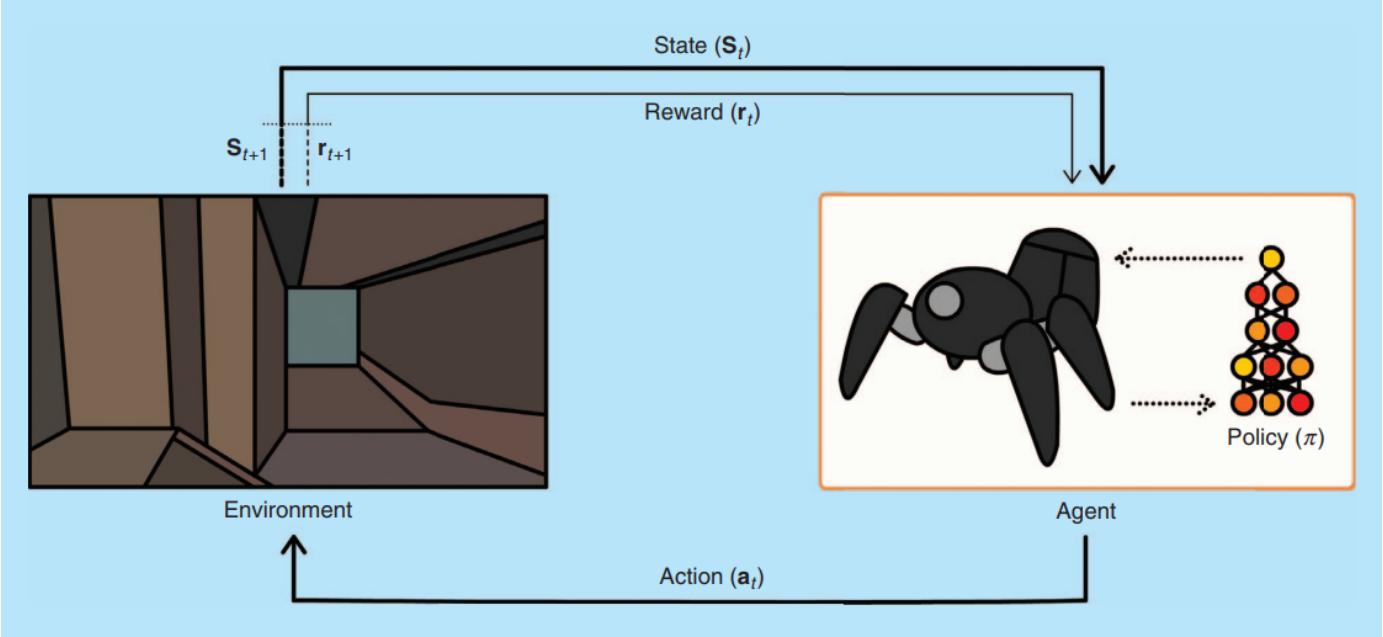

Figure 3.5.1: The agent-environment interaction cycle in RL: observation, action selection, feedback, and policy update [5].

3.5.5 Q-learning

Q-learning is a foundational algorithm in reinforcement learning, renowned for its versatility in sequential decision-making tasks. Originally introduced as an incremental method for infinite-horizon Markov Decision Processes, it has since been applied across disciplines including statistics, artificial intelligence, and control theory [18].

The Q-function

The core of Q-learning is the action-value function, or the Bellman Optimality Equation, denoted as $Q(x, a)$. It estimates the expected cumulative reward of taking action a in state x and following the current policy thereafter [18]:

$$Q(x, a) = \mathbb{E}\Big[Y + \gamma \max_{a'} Q(x', a') \mid x, a\Big],$$

Where:

- Y represents the immediate reward obtained after executing action a.
- $\gamma \in [0, 1]$ is the discount factor, balancing immediate versus future rewards; lower values prioritize short-term gains, while higher values emphasize long-term planning.
- x' denotes the next state reached after taking action a in state x.
- a' is the next action considered in the subsequent state x'.

This formulation is recursive and aligns with the Bellman Optimality Equation, which provides the theoretical foundation for Q-learning.

Iterative Computation of the Q-Function

The Q-function can be approximated using different iterative methods [18]:

- **Value Iteration**: This method directly implements the Bellman equation by repeatedly updating the Q-function:

$$Q^{k+1}(x, a) = \mathbb{E}\Big[Y + \gamma \max_{a'} Q^k(x', a') \mid x, a\Big].$$

As $k \to \infty$, Q^k converges to Q^* when $\gamma < 1$.

- **Temporal-Difference Learning (TD)**: In practice, Q-learning uses a temporal-difference update rule to refine $Q(x, a)$ incrementally based on observed transitions:

$$Q(x, a) \leftarrow Q(x, a) + \alpha \Big[Y + \gamma \max_{a'} Q(x', a') - Q(x, a) \Big],$$

Where $\alpha \in (0, 1)$ is the learning rate controlling the step size of each update.

Convergence and Stability

Q-learning is guaranteed to converge to Q^* under certain conditions:

1. The learning rate α diminishes over time but not too quickly.
2. Every state-action pair is visited infinitely often, ensuring sufficient exploration.
3. The discount factor $\gamma < 1$ ensures that the cumulative rewards remain finite.

Under these conditions, Q-learning provides a robust and widely applicable solution for sequential decision problems.

3.5.6 Deep Reinforcement Learning

Deep reinforcement learning (DRL) is an advanced integration of artificial neural networks and RL. The term "deep" highlights the presence of multiple neural network layers that mimic the structure and functioning of the human brain. Implementing DRL typically demands extensive interactive datasets and substantial computational resources. However, the rapid advancements in computing power in recent years have paved the way for numerous successful applications of DRL.

The first successful attempt to combine reinforcement learning with deep learning is credited to the deep Q-network (DQN) developed by Google DeepMind [49]. This approach integrated Q-learning with a convolutional neural network to approximate the optimal Q-function in a continuous state space.

One notable enhancement to DQN is the double deep Q-network (Double DQN) [30], proposed by Hasselt et al. (2016), which incorporates double Q-functions to reduce value overestimation. This technique involves learning two Q-functions to decouple action selection from action evaluation. Another variant, the deep deterministic policy gradient (DDPG) algorithm [44], is designed for continuous control tasks. It learns both a Q-function and a deterministic policy simultaneously. However, DDPG is prone to overestimation of Q-values, which can lead to unstable policies. To address this issue, Fujimoto et al. (2018) introduced twin delayed DDPG (TD3), which incorporates techniques such as bounded double Q-functions and delayed policy updates to improve stability [25].

3.5.7 Multi-Agent Systems

A multi-agent system (MAS) consists of several autonomous decision-making entities—agents—that function within a common environment, each making localized judgments based on incomplete observations, personal policies, or decentralized information. Each agent may function as a gateway, scheduler, or orchestrator in the context of edge-cloud computing, controlling a portion of resources or responding to task requests concurrently with other agents.

MAS's decentralized architecture offers a number of benefits over centralized methods. Combined explosion is a problem with traditional control frameworks that demand joint optimization across all system components. As the number of entities rises, the computing cost grows exponentially. By using scalable local calculations, the multi-agent rollout paradigm, as described in [9], lessens this burden. Because each agent separately calculates its own policy or value estimations, the system as a whole may grow linearly in size as the number of agents increases. In contemporary edge-cloud architectures, where scalability and real-time responsiveness are crucial, this feature is very important.

Furthermore, MAS designs are naturally adaptable and suitable for deployment in a distributed setting. Agents frequently need little communication and can work in parallel and asynchronously. This makes them perfect for applications like autonomous robots, widely dispersed computer systems, and large-scale sensor networks where centralized management is unfeasible because of latency, fault tolerance, or infrastructure limitations. The freedom of communication is another important benefit. From completely isolated agents to systems with selected, restricted coordination, MAS algorithms may be adjusted to function in a variety of communication regimes. From bandwidth-constrained edge nodes to high-speed data centers, this flexibility enables a broad variety of deployment scenarios.

3.6 Resource Allocation

Resource allocation is the process of efficiently distributing computing resources, such as CPU, memory, bandwidth, power, and storage, among competing applications, services, or users to ensure optimal system performance while maintaining QoS requirements [3]. This process is crucial in modern computing paradigms, where computing demands are continuously evolving, requiring adaptive and efficient resource management strategies.

Achieving these objectives forms the foundation for the sustainable and intelligent operation of modern distributed systems. The problem of optimal resource allocation under constraints of cost, latency, and energy consumption shares similarities with the classic knapsack problem and ultimately belongs to the class of NP-complete problems [41], which necessitates the application of intelligent and approximate methods for its effective solution.

In distributed and heterogeneous environments, where computing nodes vary in capacity, power availability, and proximity to data sources, resource allocation becomes particularly challenging. With the increasing number of interconnected devices generating vast amounts of data, managing computational workloads, data transfers, and energy consumption effectively is essential to prevent network congestion, latency issues, and inefficient power usage. Optimized resource allocation mechanisms are required to balance workloads, regulate network bandwidth, and distribute storage efficiently, ensuring that computational resources are utilized effectively while meeting performance and QoS requirements [73, 56]. By integrating adaptive allocation strategies, modern computing infrastructures can dynamically respond to fluctuating demands, ensuring efficient, reliable, and cost-effective operations across diverse applications.

3.6.1 Necessity of Resource Allocation

The rapid expansion of cloud computing, edge computing, and IoT has significantly increased the complexity of managing resources efficiently. Cloud-native applications operate in highly dynamic, multi-tenant environments, where workloads fluctuate unpredictably. Similarly, edge computing introduces constraints on computation and power availability, making it essential to efficiently balance workloads between cloud servers, edge nodes, and end devices [60].

The key objectives of resource allocation include:

- **Scalability and elasticity**: Ensuring seamless adaptation to workload fluctuations.
- **Low-latency processing**: Optimizing task distribution for real-time applications.
- **Energy efficiency**: Allocating power-constrained resources in IoT and edge computing.
- **Avoiding network congestion**: Regulating bandwidth usage to maintain seamless connectivity.
- **Cost optimization**: Dynamically provisioning cloud resources to minimize expenses.

Chapter 4

Related Work

Contents

4.1 Forecasting in Cloud Resource Management

4.1.1 Motivation for Forecasting in Cloud Systems

Cloud computing has experienced rapid growth in recent years and is now widely adopted across enterprises and organizations. One of the major challenges in this domain is the effective processing and accurate prediction of time-series data generated by cloud systems [50]. Resource provisioning is critical to this challenge, and it must be both adaptive and predictive to match dynamic workload demands.

Time series data refers to sequences of observations collected at successive, evenly spaced points in time. In cloud computing environments, such data typically arises from telemetry systems that monitor and log various performance and usage metrics over time. These include, but are not limited to, CPU utilization, memory consumption, disk I/O rates, network traffic, and task scheduling patterns. Each of these metrics forms a univariate or multivariate time series that captures the dynamic behavior of cloud infrastructure and workloads. This temporal data is crucial for understanding system behavior, detecting anomalies, and forecasting future states. Effectively analyzing time series data is essential for intelligent resource provisioning, performance optimization, and meeting service-level objectives in cloud-native systems.

Cloud workloads are inherently high-dimensional, nonstationary, and often exhibit sudden spikes or drops. Existing forecasting methods frequently struggle to cope with this variability, leading to inefficiencies such as over-provisioning or under-provisioning of resources. Over-provisioning results in underutilized infrastructure and increased operational costs, while under-provisioning may cause degraded application performance and violations of Service Level Agreements (SLAs) [16]. For instance, when a large number of user requests arrive concurrently, the system may experience a sudden workload spike that exceeds the available capacity, causing service delays or failures. Conversely, during periods of low demand, resources may remain idle, leading to unnecessary energy consumption and financial waste [16]. These opposing scenarios highlight the importance of precise and responsive workload prediction mechanisms in modern cloud environments.

4.1.2 Forecasting for Resource Allocation

The fundamental goal of predictive forecasting in cloud computing environments is to accurately anticipate future resource consumption patterns, enabling proactive allocation decisions that optimize system performance and efficiency. This predictive capability serves as a critical foundation for numerous cloud management strategies, where forecasting functions as an essential preprocessing component for sophisticated control mechanisms, including autoscaling systems, load balancing algorithms, and reinforcement learning-based resource orchestration.[33][72]

Target Variables and Granularity Considerations

Because it is a crucial indicator of workload pressure and system stress, CPU utilization stands out among the other resource metrics available for prediction as the variable that is most commonly targeted in the literature. Because CPU utilization has a direct relationship to application performance and may be used to assist scaling decisions, it was chosen as the main prediction objective.

4.1.3 Classical Forecasting Approaches

Time series forecasting in cloud computing has historically been addressed using classical statistical techniques. These methods include AutoRegressive (AR), Moving Average (MA), AutoRegressive Integrated Moving Average (ARIMA), Vector AutoRegression (VAR), and their variations [51]. These models assume stationarity and linear relationships in the data, which makes them well-suited for predictable, low-noise environments. In their survey, Derdus et al. [20] presented a detailed review of statistical techniques for characterizing cloud workloads. Basic statistical descriptors such as mean, standard deviation, percentiles, cumulative distribution function (CDF), and coefficient of variation (CoV) are commonly used to explore variability in resource consumption. For short-term forecasting, Simple Moving Averages (SMA) and ARIMA models are typically applied. ARIMA performs effectively when the input time series is smooth and normally distributed [12], but its accuracy deteriorates in the presence of noise or nonlinear patterns, common

characteristics of cloud telemetry data. Correlation-based analysis, using metrics like the Pearson Correlation Coefficient (PCC) and Spearman Rank Correlation Coefficient (SRCC) [51], is also employed to study relationships among resource types over time. Additionally, the Auto-Correlation Function (ACF) is widely used to detect short-term patterns and daily cycles in usage behavior.

A strength of statistical forecasting methods lies in their transparency and low computational cost. Automated model order selection and parameter estimation techniques make these models simple to implement. However, they cannot capture nonlinear interactions or adapt to complex workload dynamics, which limits their scalability in modern, cloud-native systems.

4.1.4 Machine Learning and Deep Learning Models

To overcome the limitations of classical methods, researchers have increasingly applied ML and DL models for cloud workload prediction [26]. ML techniques such as Linear Regression, Support Vector Machines (SVM), Random Forests (RF), and k-Nearest Neighbors (k-NN) offer improved generalization by learning patterns directly from data.

Gao et al. [26] identified several ML-based methods frequently used for virtual machine (VM) resource prediction. Neural Networks (NN), when combined with techniques like windowing or regression, have demonstrated improved accuracy over linear models [35]. Nikravesh et al. [53] showed that SVM outperforms NN and Linear Regression under conditions of frequent workload fluctuation, although its performance declines with large-scale datasets.

Lazy learning strategies such as Local Learning and Nearest Neighbors simplify complex time series modeling by constructing localized models for each input, using dynamically selected neighborhoods [11]. These approaches are effective in short-horizon predictions but suffer from scalability and sensitivity to noise.

On the deep learning front, architectures LSTM, DNN, Autoencoders, and Transformers have demonstrated strong performance in workload prediction tasks [57]. LSTM networks excel particularly in capturing temporal dependencies and sequential patterns that are inherent in resource usage time series data. These recurrent neural network architectures are specifically designed to address the vanishing gradient problem, enabling them to learn long-term dependencies in sequential data while maintaining the ability to capture short-term fluctuations.[39] The effectiveness of LSTM models in cloud workload prediction has been demonstrated across multiple studies, with researchers showing major improvements in prediction accuracy when compared to conventional statistical approaches.[39]

Transformer Models have emerged as a powerful alternative to LSTM networks, showing superior performance in modeling long-range dependencies and complex temporal relationships present in cloud workload data. These attention-based architectures leverage self-attention mechanisms to capture relationships between different time points in the input sequence, enabling more effective modeling of complex temporal patterns. The attention mechanism inherent in Transformer architectures allows these models to focus on relevant historical information when making predictions, leading to more accurate forecasting of dynamic and bursty cloud workloads.[4] This capability is particularly valuable in cloud environments where workload patterns can exhibit sudden spikes and unprecedented changes in user request patterns over time.[54]

While DL models achieve high accuracy and handle nonlinearity well, they also require large volumes of labeled training data, computational resources, and careful tuning of hyperparameters. Their interpretability also remains a challenge in practical deployments.

4.2 Resource Allocation Mechanisms

Resource allocation (RA) is a fundamental aspect of cloud and edge computing systems, involving the distribution of limited computational, storage, and network resources to competing tasks or users. An efficient RA mechanism is crucial for optimizing system performance, maximizing resource utilization, minimizing operational costs, and ensuring that user requirements and QoS constraints are satisfied [22].

Resource allocation mechanisms can broadly be categorized into two classes: **static** and **dynamic**. Static mechanisms operate with pre-defined decisions made offline or before task execution, often relying on op-

timization techniques or historical workload profiles. While simple and computationally efficient, they are limited in their responsiveness to real-time system changes. Conversely, dynamic mechanisms adapt decisions at runtime based on current system state, workload fluctuations, or predictive models. These methods enable more flexible and resilient scheduling policies, often leveraging reinforcement learning, heuristics, or control theory.

The following sections review both categories, highlighting key techniques, applications, and limitations, with particular emphasis on their relevance to cloud-native application scenarios.

4.2.1 Workload Variability in Resource Allocation

Cloud workloads exhibit a wide spectrum of temporal patterns, which impact the effectiveness of resource allocation strategies. As illustrated in Figure 4.2.1, these patterns can be categorized into five classes [23]:

- **Static:** Constant resource usage over time, typical of low-interaction services such as internal dashboards or background monitoring agents.
- **Periodic:** Predictable cycles of demand, such as shopping platforms during weekends or traffic sensors during peak hours.
- **Single-Peak (Once-in-a-Lifetime):** Generally stable workloads interrupted by one-time bursts—commonly seen in payroll, billing, or software deployment.
- **Unpredictable:** Highly variable usage patterns with no clear trend. Often linked to user-driven activity, anomaly detection systems, or emergency dispatch platforms.
- **Continuously Changing:** Gradual but persistent changes in resource demands. Examples include social media applications and mobile app stores where user engagement evolves.

Figure 4.2.1: Representative patterns of cloud application workloads. Adapted from [23].

Understanding these patterns is essential when designing adaptive resource management systems. Static allocation schemes may perform well under stable or periodic conditions but fail in environments characterized by bursty or nonstationary behavior. As such, dynamic and learning-based methods are increasingly employed to handle the complexity introduced by real-world workload variability.

4.2.2 Static Resource Allocation Mechanisms

Static resource allocation mechanisms rely on pre-determined rules and fixed provisioning strategies established before task execution. Unlike dynamic schemes, they do not adjust resource assignments in response to runtime variations. These approaches are typically suitable for predictable workloads or long-term capacity planning and are commonly used in scenarios where decision overhead must be minimized.

One of the primary methodologies in static algorithms is the use of mathematical programming. Techniques such as integer programming (IP), mixed-integer linear programming (MILP), and mixed-integer nonlinear programming (MINLP) are widely used to model resource management problems that involve multiple constraints and optimization objectives. For instance, in multi-cloud environments, 0–1 integer programming models are applied to determine optimal placements of services or applications across cloud data centers. These models consider static parameters—such as CPU and memory capacities, latency constraints, and monetary cost—to minimize deployment cost or maximize overall system utilization [76]

In addition to deterministic formulations, some static algorithms incorporate stochastic programming to manage demand and pricing uncertainties. These models assume that the probability distributions of resource demand or price are known in advance. A notable example is two-stage stochastic integer programming, which plans both reservation and on-demand resource allocation under uncertain but estimable demand profiles. The objective in such models is typically to minimize the expected total cost over a planning horizon while ensuring SLA compliance [76].

The primary advantage of static allocation lies in its simplicity and predictability. By reserving resources in advance, cloud providers can ensure that applications have guaranteed access to the necessary resources, minimizing the risk of resource contention or performance degradation. This can be particularly beneficial for legacy systems or batch processing tasks with well-understood requirements.[70]

However, the fundamental limitation of static approaches lies in their inability to adapt to real-time system changes, such as sudden workload bursts, hardware failures, or shifting user demands. In such cases, the pre-computed allocations may lead to underutilization of resources or SLA violations. As noted by Zhang et al. [76], static scheduling is often insufficient in highly dynamic or heterogeneous cloud environments, necessitating integration with dynamic or online scheduling algorithms.

4.2.3 Dynamic Resource Allocation Mechanisms

Dynamic resource allocation (DRA) mechanisms are essential for managing the volatile workloads typical of modern cloud computing environments. These mechanisms dynamically adjust resource assignments—such as CPU, memory, and storage—based on real-time system state or predictive demand forecasts. Their primary goals are to optimize application performance, minimize resource waste, and ensure compliance with SLAs, while also reducing energy and operational costs.

Predictive Resource Management. Predictive techniques leverage historical and real-time telemetry data to forecast future workload demand. Common models include linear regression, polynomial regression, statistical multiplexing, and Kalman filters. Dawoud et al. [19] proposed a controller-based system that utilizes linear prediction to proactively scale VMs, thereby avoiding SLA violations and reducing energy consumption. Proactive scaling improves responsiveness by anticipating rather than reacting to demand surges.

Optimization-Driven Approaches. These techniques treat resource allocation as a multi-objective optimization problem, balancing conflicting goals such as performance, energy efficiency, and cost. Ficco et al. [24] introduced a hybrid method combining the coral-reefs optimization algorithm and game theory to dynamically reallocate resources in a cloud federation, targeting both SLA satisfaction and energy reduction.

Rule-Based and Heuristic Systems. These lightweight methods use threshold-based triggers (e.g., CPU $> 80\%$) to initiate scaling. Tighe et al. [66] implemented a rule-based system combining vertical scaling and VM consolidation to reduce energy usage while maintaining application responsiveness. Though computationally inexpensive, such approaches may lack the precision of predictive or optimization-based methods.

Feedback Control Systems. Inspired by control theory, these systems maintain system stability via continuous monitoring and closed-loop adjustments. PID controllers are commonly used to regulate resource allocation in response to workload fluctuations. Dawoud et al. [19] applied such methods to preemptively trigger horizontal scaling, improving SLA compliance without human intervention.

Energy-Aware Mechanisms. Energy-efficient strategies prioritize minimizing the energy footprint of data centers, often by consolidating workloads onto fewer hosts and powering down idle machines. Verma et al. [69] proposed the pMapper framework, which dynamically places VMs based on energy models and migration costs, demonstrating energy savings while maintaining performance.

Research Challenges and Trends. Despite their effectiveness, DRA mechanisms face several challenges. Accurate prediction is crucial; errors can lead to over-provisioning or under-provisioning, impacting both costs

and SLA compliance. Migration and scaling operations introduce overhead and potential performance degradation. Response latency, especially in horizontal scaling, remains a bottleneck for bursty workloads. Additionally, balancing multiple objectives—performance, energy, cost—necessitates computationally intensive optimization. Recent research is increasingly exploring hybrid solutions that combine predictive modeling, optimization, and feedback control to overcome these limitations [36].

Why Reinforcement Learning?

Traditional computation task offloading strategies in cloud and edge environments have primarily relied on conventional optimization techniques. While these methods offer theoretical guarantees and tractable formulations, they are often constrained to approximate solutions and require strong assumptions about the underlying system dynamics. Moreover, they struggle to adapt effectively to highly dynamic network conditions, heterogeneous service demands, and real-time variability in workload patterns.

As modern cloud-native infrastructures grow increasingly complex—with decentralized architectures, latency-sensitive applications, and dynamic workloads—there is a growing need for intelligent, autonomous decision-making mechanisms. This has led to the emergence of Reinforcement Learning as a promising alternative for resource allocation and task offloading. RL enables agents to learn optimal behavior through trial and error, without requiring prior knowledge of environment statistics, thus offering superior adaptability in non-stationary and uncertain environments.

However, traditional tabular RL methods face scalability challenges due to the exponential growth of the state and action spaces, known as the curse of dimensionality. These limitations make them infeasible for large-scale distributed systems with many users, nodes, or configuration parameters. To address this, Deep Reinforcement Learning techniques have been introduced, leveraging deep neural networks to approximate value functions and learn policies in high-dimensional spaces. Notably, the Deep Q-Network (DQN) has demonstrated success in learning effective policies for dynamic task placement and resource provisioning in edge–cloud systems [46].

Beyond single-agent formulations, Multi-Agent Deep Reinforcement Learning (MARL) introduces multiple learning entities that operate in parallel and interact within a shared environment. In distributed edge–cloud infrastructures, each agent may correspond to a gateway or scheduler responsible for a local resource domain. The Multi-Agent Deep Q-Network (MADQN) framework extends the scalability and adaptability of DRL to multi-node settings, enabling decentralized decision-making with coordinated or competitive dynamics. Compared to centralized approaches, MADQN systems reduce computation and communication overhead while enhancing robustness and parallelism. Agents can learn independently, using shared policies or local observations, and still converge toward cooperative behavior that optimizes global system efficiency. This decentralized architecture aligns naturally with the operational realities of modern edge–cloud platforms, where global state synchronization is costly or infeasible [8].

Chapter 5

Problem Formulation and System Model

Contents

5.1 System Model

The increasing complexity and heterogeneity of computational workloads in modern multi-tier edge-cloud infrastructures pose many challenges for efficient resource management. These systems must accommodate jobs with varying latency sensitivities, priority levels, and resource demands, while also ensuring compliance with QoS constraints. The central problem addressed in this thesis is the design of an intelligent, adaptive resource allocation mechanism that can dynamically place jobs across edge and cloud tiers to optimize system-wide performance, minimize operational costs and energy consumption, and respect service-level requirements.

To tackle this challenge, we first develop a formal system model that abstracts the core components of the infrastructure, including the hierarchical edge-cloud architecture, job arrival characteristics, and physical resource constraints. This model defines how computational jobs interact with the available computing nodes, how priorities influence placement decisions, and how system dynamics (e.g., load fluctuations) affect scheduling outcomes. The system model serves as the operational environment in which a multi-objective optimization problem is formulated, which guides the design of a reinforcement learning-based scheduler capable of learning priority-aware allocation policies.

5.1.1 Hierarchical Multi-Tier Cloud Infrastructure Model

The proposed system operates within a hierarchical multi-tier cloud infrastructure designed to handle diverse computational workloads efficiently. The architecture is composed of three interconnected tiers—Near Edge (T_0), Far Edge (T_1), and Cloud (T_2)—each offering a distinct balance between latency, cost, and resource capacity, as summarized in Table 5.1.

Each tier T_i is formally defined by the tuple:

$$T_i = (M_i, Cap_i, L_i, Cost_i, P_i)$$

Where:

- M_i is the set of machines in tier T_i,
- Cap_i is the CPU capacity per machine,
- L_i is the average network latency to that tier,
- $Cost_i$ is the operational cost per CPU-hour,
- P_i is the power consumption in watts per core.

Tier	**Latency**	**Cost**	**Capacity**
Near-Edge (T_0)	Low	High	Limited
Far-Edge (T_1)	Medium	Medium	Moderate
Cloud (T_2)	High	Low	Practically Unlimited

Table 5.1: Comparison of Resource Tiers

The near-edge tier (T_0) offers the lowest network latency, making it ideal for latency-sensitive tasks, but its CPU capacity is limited, and operational cost is high. The far-edge tier (T_1) provides a moderate balance—lower cost and increased capacity at the expense of slightly higher latency. The cloud tier (T_2) delivers virtually unlimited computational resources at minimal cost but is best suited for delay-tolerant batch workloads due to its higher latency.

Figure 5.1.1 illustrates the infrastructure's layout. Edge devices are distributed across multiple gateways, where each gateway acts as an entry point for computational jobs. Gateway 1 (green), Gateway 2 (blue), and Gateway 3 (red) represent different edge nodes that collect jobs from nearby devices and make placement decisions across the available tiers.

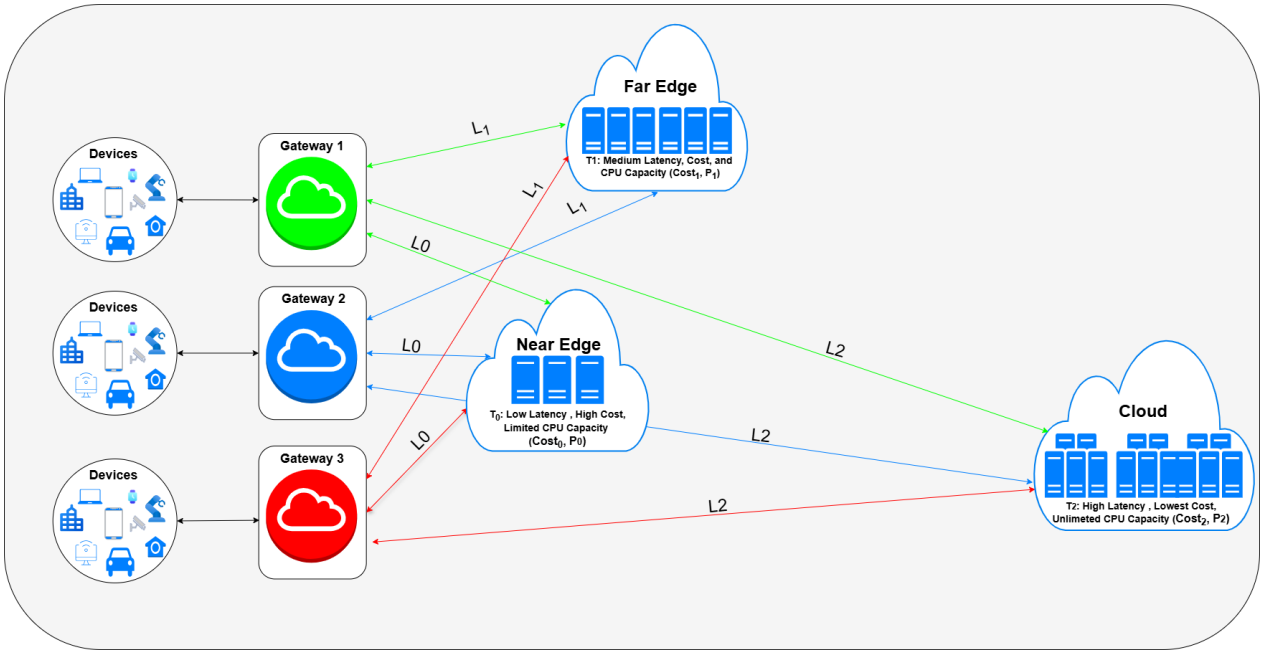

Figure 5.1.1: Hierarchical Multi-Tier Cloud Infrastructure Model showing the three-tier architecture with edge devices, gateways, and computational tiers interconnected through network links with varying latency characteristics.

The network paths between the tiers are characterized by progressively increasing latency values, denoted as L_0, L_1, and L_2, corresponding to connections to T_0, T_1, and T_2, respectively. This hierarchical organization enables the design of intelligent job placement policies.

5.1.2 System Resource State and Utilization Modeling

At any given time τ, the state of system resources is represented as:

$$\mathcal{R}(\tau) = \{Usage_i(\tau), Avail_i(\tau) : i \in \{0, 1, 2\}\}$$

Where $Usage_i(\tau)$ denotes the total CPU demand currently allocated on tier i, calculated by summing the CPU demands of all active jobs j on all machines $m \in M_i$:

$$Usage_i(\tau) = \sum_{m \in M_i} \sum_{j \in \text{ActiveJobs}_m(\tau)} \text{cpu\_demand}_j,$$

and $Avail_i(\tau)$ is the remaining available CPU capacity on tier i, computed as the difference between the total capacity $TotalCap_i$ and the current usage:

$$Avail_i(\tau) = TotalCap_i - Usage_i(\tau).$$

The utilization ratio $Util_i(\tau)$ expresses the part of capacity currently in use on tier i:

$$Util_i(\tau) = \frac{Usage_i(\tau)}{TotalCap_i},$$

Which is constrained to remain below a predefined safety threshold Max_Util to prevent overloading. Maintaining $Util_i(\tau) < Max\_Util$ ensures stable and reliable operation by avoiding excessive resource contention and potential performance degradation.

5.1.3 Job Representation and Characteristics

Each computational job j in the system is formally defined as a tuple:

$$j = (id, start\_time, end\_time, cpu\_demand, duration, priority)$$

Where:

- **id**: A unique identifier distinguishing the job.
- **start_time**: The arrival timestamp at which the job enters the system queue.
- **end_time**: The anticipated completion timestamp after job execution.
- **cpu_demand**: The CPU resource requirement expressed in CPU units necessary for job execution.
- **duration**: The estimated execution length or runtime of the job.
- **priority**: An integer quantifying the job's priority level; lower values correspond to higher priority and stricter latency or QoS constraints.

Jobs are received as a workload trace $\mathcal{J} = \{j_1, j_2, \ldots, j_N\}$, representing the sequence of arriving computational tasks.

For a job j to be deemed *feasible* on a specific resource tier T_i, there must exist at least one machine $m \in M_i$ that can allocate sufficient CPU capacity to the job throughout its execution interval without exceeding the machine's CPU capacity constraints. Feasibility requires that resource allocation avoids the job execution intervals on the machine that do not overlap in a way that surpasses the available CPU capacity Cap_i.

This feasibility criterion guarantees the system maintains operational stability and respects QoS requirements by preventing resource contention and deadline violations during job placement.

5.1.4 SLA Constraints and Penalty Mechanism

To ensure a consistent Quality of Service for time-sensitive workloads, the system enforces explicit service level agreement constraints that are tied to the priority levels of the jobs. Each priority level $p \in \{1, 2, 3, 4, 5\}$ is assigned a maximum allowable latency budget, denoted as SLA_p. These latency budgets define the upper bounds on acceptable response times for jobs of different criticality, reflecting their urgency and importance:

- Priority 1: SLA_1 — Representing the most severe requirement for the highest-priority jobs, such as real-time control or emergency processing.
- Priority 2: SLA_2 — High-importance tasks that require rapid but slightly less strict response times.
- Priority 3: SLA_3 — Moderate-priority jobs where some latency is acceptable without severely impacting performance.
- Priority 4: SLA_4 — Low-priority jobs with more relaxed latency demands.
- Priority 5: SLA_5 — Allowing the greatest tolerance for delay, often for background or batch processing tasks.

These SLA thresholds act as critical parameters in the scheduling and resource allocation algorithms, ensuring that jobs with higher priority are preferentially scheduled and allocated sufficient resources to meet their latency targets. If a job's execution exceeds its assigned SLA threshold, the system applies a penalty within the reward framework, which is scaled according to the job's priority level. This mechanism motivates the agent to minimize SLA violations, therefore encouraging compliance with service guarantees and preserving the reliability of the system as a whole.

5.2 Problem Formulation

In edge–cloud environments, resource management must address diverse and often competing objectives. These include minimizing operational cost and energy consumption, reducing latency, balancing utilization

across tiers, and meeting quality of service constraints based on job priority. To support intelligent scheduling, we formally define a resource allocation problem that captures these requirements and can be addressed using reinforcement learning methods.

When a job j is assigned to a tier T_i, it incurs different costs and performance impacts depending on that tier's pricing model, energy profile, and network proximity. The job placement decision must, therefore, consider trade-offs among multiple objectives simultaneously.

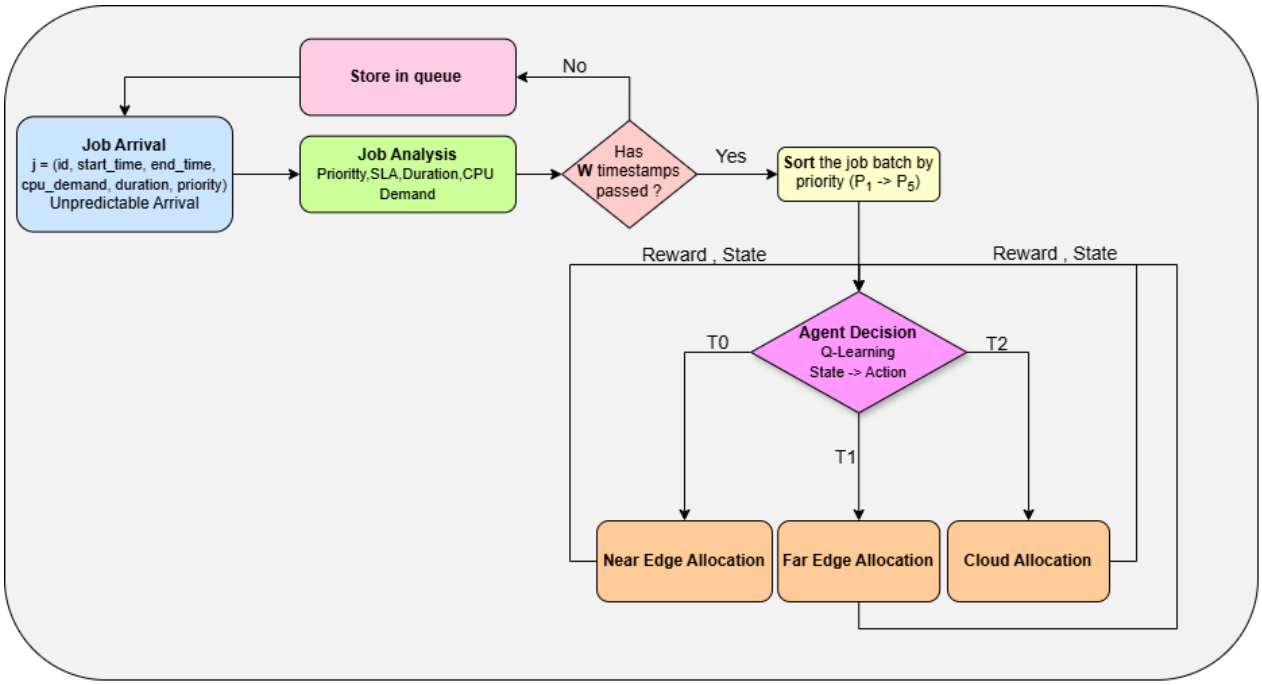

Figure 5.2.1: Complete job processing workflow showing job arrival, analysis, queueing, priority-based batch scheduling, agent decision-making, and reward feedback loop in the multi-tier edge–cloud system.

Figure 5.2.1 illustrates the complete job processing workflow within our proposed scheduling system. Jobs arrive unpredictably and are characterized by the tuple $(id, start\_time, end\_time, cpu\_demand, duration, priority)$. Upon arrival, each job is analyzed to extract relevant features such as priority level, SLA constraints, required duration, and CPU demand. To manage bursty and irregular job arrivals, the system maintains a dynamic job queue and adopts a windowed scheduling strategy. Incoming jobs are grouped into time slots, forming scheduling windows. Within each batch, jobs are sorted in descending order of priority ($P_1 \rightarrow P_5$), ensuring that time-sensitive and mission-critical tasks are allocated first. This approach balances responsiveness with computational efficiency by enabling batch-based decision-making while honoring priority constraints. Placement decisions are made by a reinforcement learning agent using the Q-learning algorithm. The agent observes the current system state—including job features and real-time resource utilization—and selects an action from the discrete space $\mathcal{A} = \{T_0, T_1, T_2\}$, representing the available tiers (Near Edge, Far Edge, Cloud). Each decision yields an immediate reward based on a custom multi-objective reward function. This reward guides the agent's learning process and refines its policy over time. The entire workflow operates as a closed-loop system: each placement decision affects the future system state and subsequent scheduling decisions. This adaptive mechanism enables dynamic and efficient resource management under real-world variability.

Cost, Latency, and Energy Metrics

The cost of executing a job j on a specific tier i is computed based on the job's CPU demand, its duration, and the cost rate associated with that tier. Formally, this is expressed as:

$$Cost_j^i = cpu\_demand_j \times \frac{duration_j}{3600} \times Cost_i, \quad \forall j \in \mathcal{J}, \forall i \in \mathcal{T}$$

Here, cpu_demand_j denotes the number of CPU units required by job j, $duration_j$ is the execution time of the job measured in seconds, and $Cost_i$ represents the monetary cost per CPU-hour at tier i. This formulation allows for a proportional cost assessment that reflects both the intensity and length of resource usage.

Latency, a critical performance metric especially for time-sensitive applications, is modeled as the sum of inherent network and processing delays characteristic of each tier. For simplicity, latency is represented by a baseline value specific to the tier:

$$Latency_j^i = L_i$$

Where L_i captures fixed delay components such as communication overhead and processing latency at tier i. Although queueing delays and dynamic congestion effects are also important in practice, this baseline latency provides a fundamental measure of the expected responsiveness at each tier.

Energy consumption is modeled to quantify the power used during job execution, which directly impacts operational cost and environmental footprint. Following the approach in [1], energy consumption for running job j on tier i is calculated as:

$$Energy_j^i = P_i \times cpu\_demand_j \times \frac{duration_j}{3600}$$

Where P_i represents the average power consumption in Watts per CPU unit for tier i. This expression captures the linear relationship between CPU usage, execution time, and energy expenditure, enabling evaluation of energy efficiency alongside cost and latency considerations.

Together, these metrics form the basis for multi-objective optimization in resource allocation, balancing economic costs, performance requirements, and sustainability goals.

5.2.1 Job Characteristics and Arrival Model

Jobs arrive dynamically over time with different sequential timestamps and are grouped into scheduling batches, where each batch contains all jobs that arrived within a fixed-size time window of length w. This batching approach enables efficient decision-making and reduces computational overhead. Before scheduling begins, jobs within each window are sorted in order of priority level (i.e., highest priority first), starting selecting jobs with the lowest latency tolerance. This ensures that time-sensitive and mission-critical tasks are allocated first. At each scheduling step t, a set of jobs $\mathcal{J}^{(t)} = \{j_1, j_2, \ldots, j_w\}$ becomes available for allocation. The agent processes this batch based on current system state observations and selects placement actions from the discrete action space $\mathcal{A} = \{T_0, T_1, T_2\}$.

We define job categories based on latency sensitivity and priority level:

- **Class I (Critical)**: Priorities 1–2, require minimal latency and fast response.
- **Class II (Flexible)**: Priorities 3, tolerate moderate delay but benefit from edge execution.
- **Class III (Best-effort)**: Priority 4-5, tolerant of latency and suitable for cloud offloading.

The system must adaptively manage this heterogeneous workload while balancing objectives such as QoS, energy efficiency, and edge saturation prevention. The categorization was used in the $R_{\text{placement}}$ calculation.

5.2.2 Reward Function

We define the reward R_j^i for assigning job j to tier i as follows:

$$R_j^i = w_1 R_{\text{base}} + w_2 R_{\text{placement}}(j,i) - w_3 R_{\text{cost}}(j,i) - w_4 R_{\text{latency}}(j,i) - w_5 R_{\text{energy}}(j,i) + w_6 R_{\text{SLA}}(j)$$

Each term in the reward function represents a distinct performance dimension:

- **Baseline Reward R_{base}:** A constant reward granted for any successful job allocation: $R_{\text{base}} =$ `base_reward`. In case of allocation failure, a huge penalty is applied: $R_{\text{base}} =$ $-$`failed_allocation_penalty`.

- **Placement Reward $R_{\text{placement}}(j,i)$:** A dynamic term evaluating the alignment of the job's priority with the selected tier's characteristics and current utilization. It encourages placing high-priority jobs on low-latency, underutilized nodes and penalizes inefficient or risky placements. Formally:
  - If a high-priority job (priority ≤ 2) is placed on a lightly loaded near-edge node, a full reward $R_{\text{placement}} = \texttt{placement\_reward}$ is granted.
  - If a high-priority job is placed on a saturated edge node, a penalty is applied to discourage overloading.
  - Medium-priority jobs (priority $= 3$) placed on lightly loaded far-edge nodes receive a moderate reward.
  - Low-priority jobs (priority ≥ 4) sent to the cloud tier receive a baseline reward.
  - If edge tiers are saturated, redirecting the job to the cloud yields a baseline reward to support load balancing.
  - Misaligned placements (e.g., low-priority job on congested edge tiers) incur a penalty: $R_{\text{placement}} = -\texttt{placement\_penalty}$.
- **Cost Penalty $R_{\text{cost}}(j,i)$:**
$$R_{\text{cost}}(j,i) = Cost_j^i \cdot \texttt{Cost\_Reward}$$
- **Latency Penalty $R_{\text{latency}}(j,i)$:**
$$R_{\text{latency}}(j,i) = \text{Latency}_j^i \cdot \texttt{Latency\_Reward}$$
- **Energy Penalty $R_{\text{energy}}(j,i)$:**
$$R_{\text{energy}}(j,i) = Energy_j^i \cdot \texttt{Energy\_Reward}$$
  The values $Cost_j^i$, $Latency_j^i$, and $Energy_j^i$ are normalized to ensure balanced contribution among cost, latency, and energy terms.
- **SLA Penalty $R_{\text{SLA}}(j)$:**
$$R_{\text{SLA}}(j) = \begin{cases} -(6 - \text{priority}_j) \cdot \texttt{SLA\_Penalty}, & \text{if SLA is violated} \\ 0, & \text{otherwise} \end{cases}$$
  This term penalizes SLA violations based on job priority. The higher the priority (i.e., the lower the numerical value), the larger the penalty when deadlines are missed.

All reward components are weighted by coefficients w_1 through w_6, allowing the system designer to tune the trade-off between performance objectives such as efficiency, latency, and SLA compliance.

Figure 5.2.2 illustrates the computational flow for calculating R_j^i when a job is placed on a tier. The placement decision triggers the parallel evaluation of each component: alignment score, cost, latency, energy, and SLA check. If allocation is successful, a base reward is granted; otherwise, a failure penalty is applied. In the case of SLA violations, the penalty is scaled by priority severity. The final reward is computed by aggregating the components using the weighted formula, and the resulting scalar value is used by the reinforcement learning agent as feedback for policy optimization.

Reward Weighting Configuration

The reward function balances multiple competing objectives to guide the agent's decision-making. It favors priority-aware scheduling by assigning notable importance to job priority, ensuring that critical, delay-sensitive tasks are prioritized.

Cost and energy penalties are incorporated to encourage efficient resource usage and sustainability, but they are weighted moderately relative to priority. This design reflects a conscious trade-off: while reducing operational expenses and energy consumption is important, the system prioritizes meeting service-level agreements

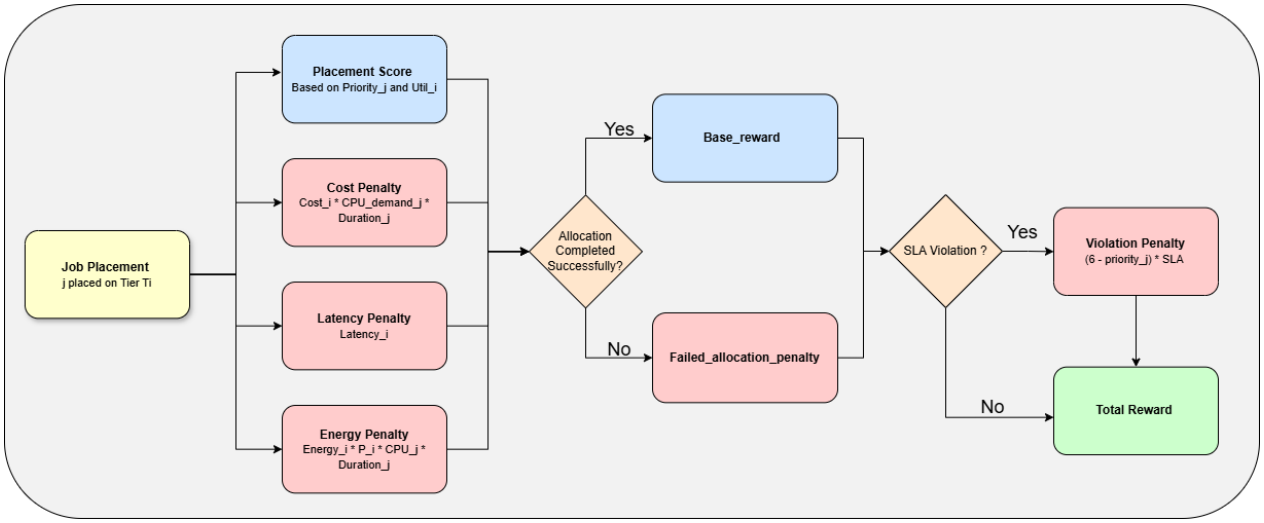

Figure 5.2.2: Reward function calculation flow diagram showing the parallel computation of reward components, success/failure evaluation, SLA violation checking, and final reward aggregation for job placement decisions.

and maintaining low latency for high-priority jobs. That is a choice we made for our experiments. However, the reward function's modular weighting scheme provides the flexibility to easily adjust these priorities. By tuning the weights, we can shift the model's preference toward emphasizing cost savings, energy efficiency, latency reduction, or priority compliance as needed. This adaptability allows the system to be tailored for different operational goals or workload characteristics without changing the underlying architecture.

Objective

In our experimental setting, we model the job allocation process in the edge-cloud infrastructure as a discrete-time **Markov Decision Process**, enabling each agent (gateway) to learn optimal scheduling strategies through Q-learning.

The MDP formulation is tailored to our environment as follows:

- **States** s_j: Each job j arrives with a state vector that includes normalized CPU demand, duration, priority, and the real-time utilization levels of near-edge, far-edge, and cloud tiers.
- **Actions** $a_i \in \mathcal{A} = \{T_0, T_1, T_2\}$: The agent must decide to place the job on one of the three resource tiers—near-edge, far-edge, or cloud.
- **Transitions**: After each allocation decision, the environment updates its internal state (e.g., CPU usage), and the agent observes the next job in the trace. Transitions are deterministic, as job order and durations are predefined in the workload trace.
- **Reward** R_j^i: The agent receives an immediate scalar reward based on the result of its placement decision, as defined by our custom reward function, as described at 5.2.2, combining placement alignment, cost, latency, energy, and SLA penalties.

The agent learns a value function $Q(s, a)$ that estimates the expected cumulative reward of choosing action a in state s. Using the Bellman update rule, this Q-function is iteratively refined during training to capture both immediate feedback and long-term consequences of allocation decisions.

At inference time, the agent applies a greedy policy over the learned Q-values to select the optimal action:

$$\pi(s_j) = \arg\max_{a \in \mathcal{A}} Q(s_j, a)$$

This policy is applied independently to each incoming job, using only the current job's features and the real-time tier utilization.

The overall optimization objective is to maximize the cumulative reward across all jobs processed by the agent:

$$\max \sum_{j \in \mathcal{J}} Q(s_j, \pi(s_j))$$

Where $\mathcal{J}$ is the set of jobs observed by the single agent during deployment.

By adopting this MDP framework, our model supports fine-grained, job-level decisions while adapting dynamically to workload changes. This learning-based mechanism provides a scalable and autonomous alternative to static heuristics, offering improved responsiveness and intelligent control.

5.2.3 Multi-Agent Scheduling Architecture

To address the scalability and coordination challenges inherent in edge-cloud environments, our framework adopts a decentralized multi-agent architecture, where each edge gateway $gateway_a$ is modeled as an autonomous reinforcement learning agent. The goal is to enable intelligent and parallelized job placement across the system's shared resource tiers.

Each agent is assigned a disjoint subset $\mathcal{J}_a$ of the global job trace $\mathcal{J}$ using a round-robin distribution policy that ensures a balanced and non-overlapping workload. All agents interact with a shared infrastructure composed of heterogeneous computational nodes:

$$\mathcal{T} = T_0 \cup T_1 \cup T_2$$

Where T_0, T_1, and T_2 represent the near-edge, far-edge, and cloud tiers, respectively, as defined previously.

Local State Representation Each agent operates based solely on local observations of its assigned jobs and the real-time state of the shared infrastructure. No inter-agent communication is performed during execution, supporting a fully decentralized control paradigm.

Action Space and Decision Policy All agents share a common discrete action space:

$$\mathcal{A} = \{T_0, T_1, T_2\}$$

Where each action corresponds to selecting a resource tier for job execution. A shared Deep Q-Network (DQN) is used to learn a global Q-function $Q(s, a)$ that estimates the long-term utility of taking action a in state s.

To promote generalization and avoid redundant learning, a shared decision policy $\pi(s) = \arg\max_a Q(s, a)$ is trained and deployed across all agents. This strategy ensures consistent scheduling behavior throughout the system while allowing fully decentralized execution. Although agents act independently, the shared Q-function encodes a unified scheduling strategy that adapts to heterogeneous job profiles and dynamic system states.

Learning Objective Each agent a seeks to maximize the cumulative expected reward over its assigned job set $\mathcal{J}_a$:

$$\max \sum_{j \in \mathcal{J}_a} Q(s_j, \pi(s_j)) = \max \sum_{j \in \mathcal{J}_a} \sum_{i \in \mathcal{T}} R_j^i$$

This reward function incorporates all components defined in Section 5.2.2, including placement alignment, SLA compliance, latency, cost, and energy considerations.

Advantages and System Behavior This architecture supports distributed and scalable learning, enabling high-throughput inference in real-time systems. Because scheduling decisions are made locally and independently at each gateway, the system can efficiently accommodate growing job volumes and dynamically changing workloads.

The shared policy facilitates convergence and robustness under varying network conditions, localized congestion, and partial observability, without requiring explicit coordination or message passing between agents. Thus, the system balances decentralized autonomy with global consistency.

By leveraging a multi-agent formulation with a shared DQN backbone, the architecture offers a practical and scalable solution for intelligent resource management in modern edge-cloud infrastructures.

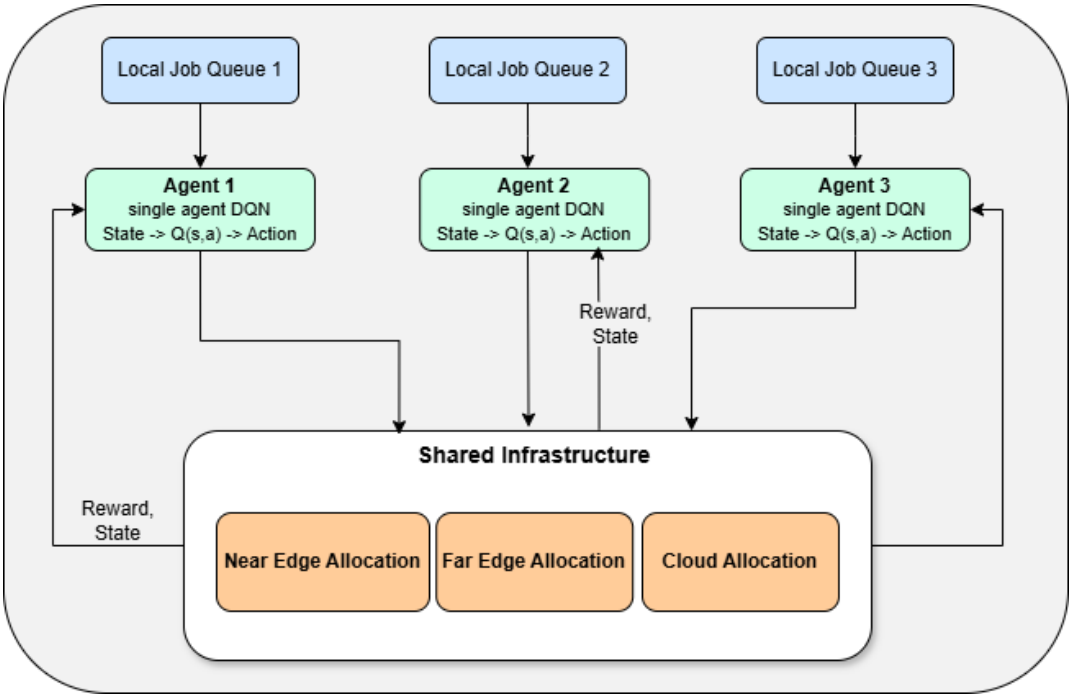

Figure 5.2.3: Multi-agent scheduling workflow showing independent agents with local job queues, shared DQN decision-making, parallel allocation attempts on shared infrastructure, and individual reward feedback loops.

Figure 5.2.3 presents the operational workflow of the decentralized multi-agent scheduling system. Each gateway agent maintains an independent local job queue containing its assigned subset $\mathcal{J}_a$. The agents operate autonomously using the shared Deep Q-Network to map local state observations s_j to optimal actions $\pi(s_j) = \arg\max_a Q(s_j, a)$. When an agent selects an action, it attempts to allocate the job on the corresponding tier—near-edge (T_0), far-edge (T_1), or cloud (T_2). Job arrival and ordering using time slots follow the same workload described earlier in Figure 5.2.2, which is omitted here for simplicity. Upon successful or failed allocation, each agent receives an individual reward and next-state feedback. Although learning updates are performed independently, all agents rely on the shared Q-function, ensuring unified scheduling behavior across the system. This workflow demonstrates how decentralized agents can collectively optimize global performance through parallel, policy-consistent decision-making.

5.2.4 Implementation Architecture

To validate our multi-agent scheduling framework, we implement the system using Kubernetes orchestration over a simulated three-tier edge-cloud infrastructure. This architecture models the physical heterogeneity of near-edge, far-edge, and cloud environments while supporting scalable, policy-driven workload scheduling through reinforcement learning agents.

The simulation operates within a single Kubernetes cluster, where each node is labeled to indicate its corresponding tier: `tier=near-edge`, `tier=far-edge`, or `tier=cloud`. These labels serve as the primary mechanism for job placement and tier differentiation. Job workloads are encapsulated as Kubernetes Pods, with tier-specific node selectors guiding their scheduling.

Each gateway agent functions as an autonomous reinforcement learning agent, receiving a disjoint subset of jobs through a round-robin distribution policy. The agents make independent tier placement decisions based solely on local observations. These decisions are then translated into pod specifications and submitted to the Kubernetes control plane. The control plane—comprising the API Server, Scheduler, and custom edge-cloud

controllers- processes these requests and schedules pods to the appropriate tier based on the agent's decision and the real-time resource availability.

Figure 5.2.4 illustrates the complete multi-agent Kubernetes simulation architecture that maps the jobs dataset to a heterogeneous edge-cloud environment. Jobs are distributed across independent gateway agents via round-robin allocation, ensuring balanced workload distribution. Each gateway agent independently makes scheduling decisions and submits pod creation requests using tier-specific node selectors (e.g., `tier=near-edge`, `tier=cloud`). The Kubernetes control plane—comprising the API Server, Scheduler, and edge–cloud-aware controllers—handles these requests, resolving contention and ensuring fair scheduling across available nodes. In the figure, the pods shown on each node represent the currently running jobs that have been successfully scheduled by the agents. These pods encapsulate the computational tasks and execute on nodes according to the agent's placement decision and the tier's available capacity. This architecture enables decentralized policy evaluation under realistic workload patterns while preserving Kubernetes-native scalability and resource management across heterogeneous computational infrastructures.

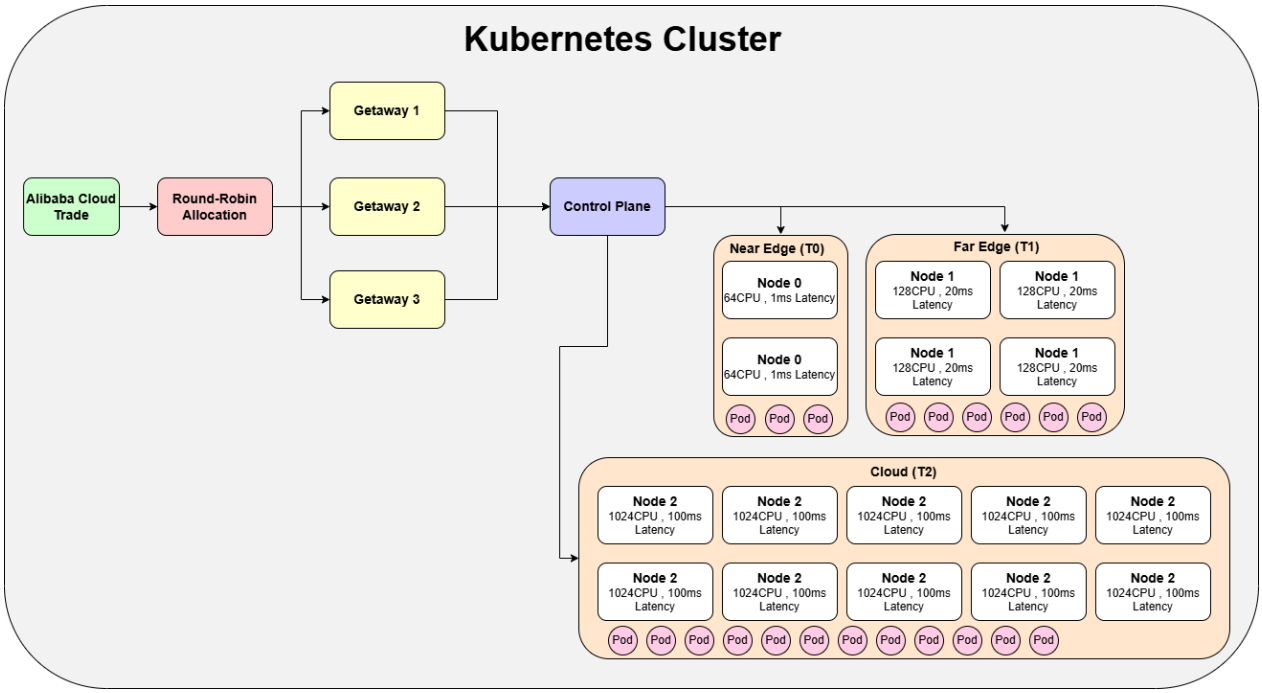

Figure 5.2.4: Multi-agent Kubernetes architecture showing round-robin job distribution from the Alibaba dataset, independent gateway agents, control plane coordination, and multi-tier node infrastructure with varying computational and latency characteristics.

Problem Goals

The system is designed to fulfill a set of interconnected objectives that reflect the challenges of resource allocation in hierarchical edge-cloud environments. These goals collectively shape the reward-driven learning process and operational behavior of the scheduling agents:

- **Minimize operational cost and energy usage:** To promote sustainable infrastructure utilization, the system imposes penalties on high-cost and energy-intensive job placements. This promotes the selection of computational nodes that offer lower monetary and power consumption footprints, particularly for non-urgent or delay-tolerant workloads. In doing so, the system tries to maximize the economic and environmental efficiency without compromising service quality for critical tasks.
- **Respect priority-based latency guarantees:** The system enforces strict compliance with SLAs that vary according to job priority. Higher-priority jobs are associated with tighter latency budgets, necessitating rapid processing and minimal delay. To overcome this constraint, the scheduler employs

a windowed allocation strategy. This window-based design allows the agent to evaluate and compare multiple pending jobs simultaneously, prioritizing the most latency-sensitive ones for immediate placement.

- **Maintain balanced resource utilization:** Preventing the overloading of specific tiers—especially the near-edge cluster—is critical for preserving long-term system health and ensuring consistent QoS. The system continuously monitors CPU availability and utilization levels across all tiers, guiding agents to distribute workloads intelligently. This load-balancing behavior reduces delays, avoids thermal stress, and mitigates the risk of job rejections or SLA violations due to resource contention. Following this strategy, the system is designed to maintain high reliability even under heavy load conditions. A job allocation failure can only occur if all three near-edge, far-edge, and cloud are simultaneously saturated. However, this scenario is highly unlikely in practice, given that the cloud tier is modeled with virtually unlimited computational resources. This architectural assumption ensures that there is always a fallback option for job placement, thereby minimizing the risk of allocation failure. As a result, the system exhibits strong reliability and robustness in managing dynamic workloads across heterogeneous infrastructure layers.
- **Coordinate decentralized decision-making across agents:** In our multi-agent architecture, each edge gateway acts autonomously while sharing a centralized Q-function learned during joint training. This allows agents to execute decisions in parallel, without explicit communication, while maintaining consistency in policy behavior. The shared Q-function encodes system-wide knowledge, enabling generalization across agents operating under different local states, job types, and tier capacities.

Together, these goals define a reinforcement learning objective that balances efficiency, responsiveness, and scalability. By embedding priority awareness, latency sensitivity, and cost constraints into the learning framework, the system effectively orchestrates heterogeneous resources in real time, adapting to fluctuating demands and constrained infrastructure conditions. This formulation supports intelligent, policy-driven job placement in realistic edge-cloud scenarios with complex, dynamic workloads.

Chapter 6

Experiments

Contents

6.1 Preliminaries

6.1.1 Data Preparation and Cleaning

The Alibaba Cluster Trace dataset 2017 [2] used in this study contains telemetry collected from batch processing jobs on a large-scale cloud platform. While the original dataset is rich in information, it is not directly usable for modeling due to noise, missing values, and high-dimensional mixed-type features. Following the methodology proposed by Lackinger et al. [40], we implemented a multi-step data cleaning process to derive a structured, time-aligned dataset for forecasting and reinforcement learning tasks.

Raw Structure and Characteristics. Initially, the raw dataset includes fields such as `job id`, `task id`, `machine id`, `status`, timestamp intervals, CPU usage (both average and maximum), and normalized memory usage. An excerpt from the original dataset is shown in Table 6.1.

Table 6.1: Snapshot of the raw dataset (Alibaba trace)

start_ts	end_ts	job id	task id	machine id	max cpu	avg cpu	max mem	avg mem
41562	41618	120	686	299	1.50	0.29	NaN	NaN
41561	41619	120	686	1279	0.89	0.28	NaN	NaN
41562	41617	120	686	828	0.94	0.29	NaN	NaN

This format contains redundant and noisy fields, such as identifiers and status codes, and suffers from missing memory values.

Cleaning Process. The transformation from raw trace logs to a usable dataset was executed through a series of preprocessing steps:

1. **Timestamp Filtering:** Original timestamps were transformed from raw epoch-like values into relative time references.

2. **Feature Reduction:** Non-numeric and irrelevant metadata fields, such as `task id`, `status`, and memory usage fields with excessive missing values (`max_mem`, `avg_mem`), were discarded. The retained features include CPU demand, machine ID, job ID, and timestamps.

3. **NaN Removal:** All rows containing missing values in essential numeric fields—particularly those required for CPU or timing analysis—were excluded to ensure a clean and complete input space. Rather than applying imputation or interpolation techniques, we opted to remove these entries entirely, as the dataset's size was sufficiently large to support robust training.

4. **Normalization:** A MinMax scaling operation was applied to all numeric columns, mapping each value into the interval $[0, 1]$. This step stabilizes the training process of both forecasting and reinforcement learning models by preventing feature dominance due to scale differences.

5. **Priority Injection:** Jobs were annotated with a discrete `priority` value from 1 (ultra-low latency) to 5 (best-effort), based on a derived SLA category linked to `job id`. This synthetic label enables reward shaping and latency-aware scheduling in subsequent RL environments (see Table 6.4).

The cleaned data was used for both forecasting (as sequential inputs) and resource allocation (as state features in the RL environment). Table 6.2 presents a piece from the final cleaned dataset.

Table 6.2: Snapshot of the cleaned dataset with real CPU usage and assigned priorities

Start Timestamp	End Timestamp	Job ID	Machine ID	Max CPU	Avg CPU	Priority
2017-01-01 00:00:00	2017-01-01 00:00:01	10528874	518	1.01	1.01	1
2017-01-01 00:00:00	2017-01-01 00:00:52	30629151	503	1.02	0.99	3
2017-01-01 00:00:01	2017-01-01 00:00:58	30629144	235	1.02	0.97	3
2017-01-01 00:00:01	2017-01-01 00:00:54	30629126	512	1.02	0.99	3
2017-01-01 00:00:01	2017-01-01 00:00:53	30629150	429	1.02	0.99	3
2017-01-01 00:00:01	2017-01-01 00:01:00	30629135	534	1.00	0.95	3
2017-01-01 00:00:01	2017-01-01 00:00:55	30629148	352	1.01	0.98	3
2017-01-01 00:00:01	2017-01-01 00:00:56	30629079	258	1.00	0.97	3
2017-01-01 00:00:01	2017-01-01 00:00:55	30629105	278	1.02	0.99	3
2017-01-01 00:00:02	2017-01-01 00:00:59	30629093	897	1.01	0.96	3

Outcome. This cleaning procedure transformed the original batch trace into a temporally ordered, numerically consistent, and semantically enriched dataset suitable for modern AI-driven workload prediction and job scheduling tasks.

6.1.2 Part 1: Forecasting

The forecasting phase of our experimental pipeline aims to predict resource usage for batch jobs using historical telemetry from a production-scale trace dataset. This phase is crucial for enabling proactive decision-making in reinforcement-learning-based resource management. By incorporating forecasting models into the decision loop, we aim to build a hybrid system capable of both reacting to immediate states and anticipating future demands.

Dataset. We used the Alibaba Cluster Trace dataset from 2017 [2], focusing on telemetry that captures the CPU usage behavior of batch jobs submitted to a large-scale production environment. For this phase, we employed a cleaned and preprocessed version of the dataset, as described in the previous section 6.1.1, retaining only relevant numeric features such as timestamps, CPU demand, job duration, and priority.

To ensure efficient training while maintaining statistical diversity, we selected a representative subset of 500,000 job instances from the original dataset of over 8 million entries. This sample size was sufficient to capture workload variability and priority distributions necessary for training forecasting models.

Objective. The objective of this forecasting phase is twofold: first, to generate accurate short-term predictions of CPU usage that can be fed into a downstream reinforcement learning agent for resource allocation; and second, to compare the relative performance and suitability of two state-of-the-art architectures, LSTM networks and transformer models, in the domain of cloud-native workload forecasting. LSTM networks are known for their ability to model long-term temporal dependencies through gated recurrent computation, while Transformers have recently emerged as a dominant paradigm in sequence modeling due to their global receptive field and capacity for parallelization. By evaluating these models under consistent conditions and on the same workload traces, we aim to provide insights into their practical effectiveness for cloud time series forecasting tasks. The results of this comparison not only inform model selection for our system but also contribute to a growing body of research on deep learning methods for intelligent infrastructure management.

Implementation Details. Model development was carried out using TensorFlow and Keras. Data loading, shuffling, and batching are handled via the sequence generator, allowing for efficient memory usage even with large input tensors. We train and evaluate both LSTM and Transformer-based neural architectures. LSTM networks are configured with multiple recurrent layers and dropout regularization to mitigate overfitting. Transformer models are adapted for time series forecasting by leveraging positional encoding and causal self-attention, ensuring the model does not attend to future values during training. Early stopping based on validation loss is employed to prevent overfitting. All experiments, training, and testing are executed on Google Colab using GPU (T4) acceleration.

Training Configuration. To ensure consistency and comparability between models, we adopted a unified training protocol for both the LSTM and Transformer architectures, inspired by the experimental methodology presented by Lackinger et al. [40]. Table 6.3 summarizes the key hyperparameters used in our experiments. Models were trained with early stopping based on validation loss and optimized using the Adam optimizer.

Table 6.3: Forecasting Model Training Configuration

Parameter	LSTM Model	Transformer Model
Input sequence length	60	60
Batch size	64 (generator), 32 fit	64 (generator), 32 fit
Optimizer	Adam	Adam
Learning rate	0.001	0.001
Loss function	Mean Squared Error	Mean Squared Error
Epochs	13	36
Early stopping patience	5	5
Validation split	20%	20%

6.1.3 Part 2: Resource Allocation: Environment and Training Setup

To evaluate adaptive job placement strategies in edge–cloud infrastructures, we design a custom reinforcement learning environment built on the `gymnasium` interface [68]. The environment simulates a single-agent scheduling scenario in which the agent dynamically allocates computational jobs to one of three available tiers: near-edge, far-edge, or centralized cloud clusters.

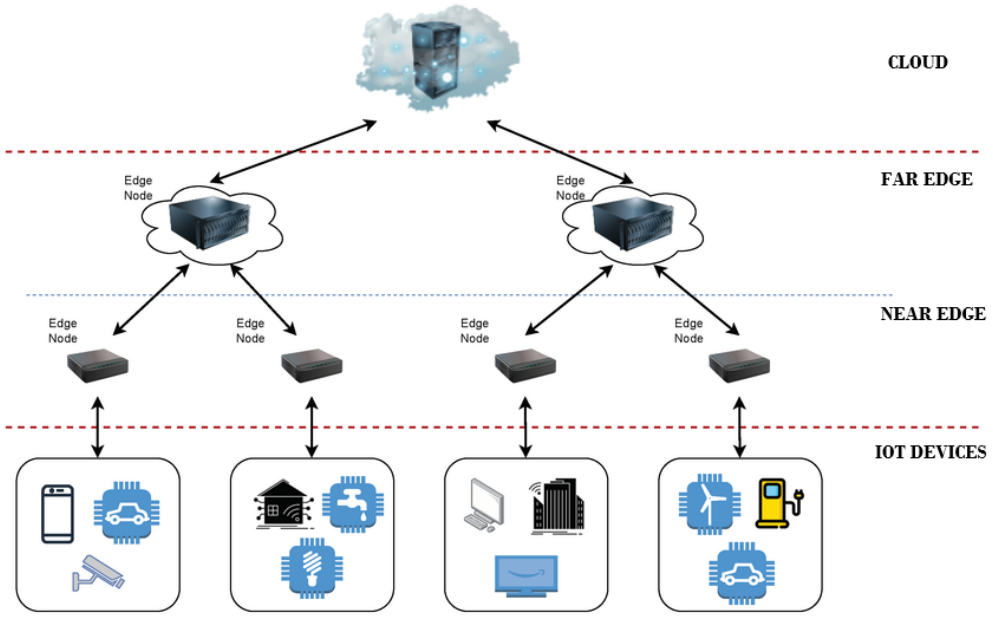

Figure 6.1.1: Hierarchical structure of the edge–cloud continuum, illustrating the placement of IoT devices, near-edge nodes, far-edge clusters, and centralized cloud. This layered architecture underpins the job placement decisions explored in our environment.

Environment Design. Each incoming job is described by metadata including CPU demand, execution duration, and latency sensitivity. The agent observes a six-dimensional continuous feature vector encoding both job-specific characteristics and current system state. The action space consists of three discrete choices: allocation to the near-edge, far-edge, or cloud tier. System-level constraints—such as resource capacity, energy budget, and latency tolerances—are embedded in the environment and directly influence job feasibility and reward formulation.

Dataset and Priority Encoding. The environment leverages a cleaned batch job dataset derived from the Alibaba Cluster Trace 2017 [2]. E. These features are used to characterize workloads and inform scheduling decisions. A comprehensive overview of the preprocessing and feature engineering steps is provided in Section 6.1.1.

Priority encoding is integral to the reward structure and encapsulates real-world QoS demands. Table 6.4 defines five priority classes based on latency tolerance and typical application domains.

Table 6.4: Latency-Aware Priority Categories for Job Scheduling

Priority Level	Latency Tolerance	Application Examples
1 – Ultra-Low	< 10 ms	Real-time control, VR/AR
2 – Low	10–50 ms	Online gaming, video conferencing
3 – Moderate	50–200 ms	Web services, transactional queries
4 – High	200–500 ms	Data syncing, periodic monitoring
5 – Best-Effort	> 500 ms	Backup jobs, batch processing

Placement Strategy. Each priority level imposes distinct placement expectations:

- **Priority 1 (Ultra-Low Latency)**: Must be executed on the near-edge to ensure strict real-time guarantees.
- **Priority 2 (Low Latency)**: Requires rapid scheduling; minor latency is tolerable.
- **Priority 3 (Moderate Latency)**: Acceptable on far-edge or cloud; typically used for web and transactional services.
- **Priority 4 (High Latency)**: Can be deferred or batched; typical for background tasks.
- **Priority 5 (Best-Effort)**: Optimally placed on the cloud to conserve edge resources.

The reward function penalizes misplacement of high-priority jobs and rewards latency-aware tier alignment.

Training Framework. Training is implemented via **Ray RLlib**[43], employing the `Standard DQN` algorithm configured through the `DQNConfig` interface. The environment is registered via `register_env` and integrated into a replay-buffered, target-updated training loop to ensure stability.

Table 6.5: Reinforcement Learning Training Configuration for Resource Allocation

Parameter	Value
Environment	`PriorityAwareEdgeCloudEnv`
Observation Space	$\mathbb{R}^6$
Action Space	Discrete (3 options)
Algorithm	DQN
Learning Rate	0.0001
Discount Factor (γ)	0.99
Replay Buffer Size	100,000
Target Network Update	Every 500 steps
Exploration Schedule	Linear ($1.0 \rightarrow 0.02$)
Batch Size	32
Training Volume	500,000 job events
Platform	Google Colab (GPU)

Training Instance Composition. To ensure balanced exposure across all priority levels, we extracted a continuous trace segment containing at least one job from each of the five QoS categories. This promotes diverse training experience and better generalization.

Infrastructure Configuration. The simulated infrastructure reflects a hierarchical edge–cloud architecture comprising three tiers:

- **Near-edge:** 1 node with 16 CPU cores
- **Far-edge:** 4 nodes, each with 16 CPU cores
- **Cloud:** Multiple nodes with 64 CPU cores each (simulating near-unlimited elasticity)

This configuration is intentionally lightweight to match the scale of the training dataset and to enable fast convergence during experimentation. For evaluation, the infrastructure is scaled up to reflect more realistic deployment conditions and to assess policy generalization under higher job volumes and system complexity. Distinct latency and power profiles are assigned to each tier to support differentiated cost and performance modeling.

Reward Function and Resource Efficiency Modeling

To guide the learning agent toward efficient and context-aware resource allocation, we design a composite reward function that captures multiple dimensions of performance: operational cost, latency adherence, energy efficiency, and CPU utilization. Each component is quantitatively defined and integrated into the environment's feedback signal.

Cost Modeling. [1] To reflect economic constraints in real-world cloud-edge deployments, we assign a monetary cost to each cluster based on its operational pricing per CPU-hour. These values are defined as:

- **Near-edge cluster:** \$0.10 per CPU-hour
- **Far-edge cluster:** \$0.05 per CPU-hour
- **Cloud cluster:** \$0.01 per CPU-hour

Latency Penalty. [1] To model QoS requirements, each job carries an implicit latency constraint encoded via its assigned priority. Each cluster is associated with a base latency, representing the expected delay incurred if a job is scheduled there:

- **Near-edge cluster:** 1 ms
- **Far-edge cluster:** 20 ms
- **Cloud cluster:** 100 ms

Energy Efficiency. [1]

The power ratings assigned to each cluster type are:

- **Near-edge cluster:** 40W
- **Far-edge cluster:** 70W
- **Cloud cluster:** 200W

These values are used to balance energy efficiency against job placement flexibility. The agent learns to prioritize low-energy tiers when suitable, especially for best-effort or low-priority jobs, while preserving SLA compliance for latency-sensitive workloads.

CPU Utilization and Overload. The utilization bonus is granted when the selected machine's utilization remains below a predefined threshold, set at 80%, in our case. An overload penalty is applied if CPU demand exceeds available capacity. This encourages effective but not excessive consolidation.

Dynamic Placement Reward. To guide the agent toward latency-aware placement decisions, we introduce an additional reward component, $R_{\text{placement}}$, which reflects soft constraints on cluster selection based on job priority. This reward acts as a real-time incentive that aligns job sensitivity with appropriate infrastructure tiers, effectively simulating SLA adherence without imposing rigid scheduling rules.

Parameterization. The environment supports a tunable configuration for all weights via a central dictionary:

```
"reward_penalty_config": {
  "base_reward": 50,
  "cost_weight": 2,
  "latency_weight": 2,
  "placement_reward": 400,
  "placement_penalty": 100,
  "utilization_bonus": 300,
  "over_utilization_penalty": 200,
  "energy_weight": 2
}
```

This design ensures flexibility in evaluating trade-offs between cost, responsiveness, sustainability, and performance.

Objective. The reward function encourages the agent to learn policies that balance economic operation (cost), QoS compliance (latency and priority), sustainability (energy), and infrastructure load (utilization). These dimensions align with the goals of modern edge-cloud orchestration systems.

6.1.4 Part 3: Multi-Agent Simulation

Motivation and Goal. While the previous phase focused on training a single centralized agent to manage resource allocation, our ultimate objective is to construct a distributed decision-making system where multiple gateways—such as local schedulers or edge orchestrators—operate concurrently over a shared infrastructure. This design aligns with modern edge-cloud paradigms, where multiple entry points independently handle user demands while contending for limited global resources.

To enable this, we extend our environment to a multi-agent setup and assign each gateway an independent agent. Crucially, each agent reuses the same `Standard DQN` policy learned during the single-agent training phase. This approach allows us to transfer learned behaviors into a distributed setting without additional training, thereby reducing the computational burden. The multi-agent setting enables evaluation of scalability, policy generalization, and system coordination when autonomous schedulers operate in parallel across decentralized workloads.

Environment Modification. We develop a multi-agent variant of the base environment. Each gateway is treated as an independent agent with its job queue but shares the same underlying infrastructure (machines, CPU cores, etc.).

Key modifications include:

- **Agent Decomposition:** The number of agents (gateways) is configurable via the parameter `n_gateways`. Each gateway receives its subset of jobs, distributed via round-robin.
- **Observation and Action Space:** Each agent observes its current job (features identical to the single-agent case) and selects one of the same three actions: assign the job to near-edge, far-edge, or cloud.
- **Job Partitioning:** The dataset is divided among gateways using a round-robin strategy to ensure load balancing. Pointers are maintained to track each gateway's job progress.

- **Shared State and Conflict Resolution:** All agents operate over a shared pool of machines, meaning allocation decisions by one gateway affect the available capacity for others. This introduces competition and potential contention for resources.
- **Termination and Statistics:** The environment tracks independent termination conditions per gateway. Global performance metrics are maintained per agent (e.g., rewards, job success rate), enabling fairness and collaboration assessment.

Objective. The goal of this multi-agent extension is to investigate how well independently trained policies generalize in distributed cooperative settings. We aim to evaluate whether learned behavior scales when multiple agents simultaneously interact with the environment, contend for shared resources, and respond to diverse workload patterns across gateways.

6.2 Experiment 1: Forecasting Performance: LSTM vs Transformer

To evaluate the predictive capabilities of deep learning models for short-term resource forecasting in cloud computing environments, we conducted a comparative analysis between two state-of-the-art architectures: LSTM network and the Transformer model. Both models were tasked with forecasting future job characteristics, specifically the maximum and average CPU usage, along with execution duration, over a defined future time window. These predictions were based on sequences of historical telemetry data obtained from the cleaned and normalized Alibaba Cluster Trace dataset.

To ensure the validity of the comparison, both models were trained under the same experimental conditions. This included uniform data preprocessing procedures, a fixed sequence length of 60 timesteps, an identical loss function (Mean Squared Error), consistent batch size, and the application of early stopping based on validation loss. These controls ensured that performance differences could be attributed solely to architectural differences rather than training disparities or preprocessing bias. The trained models were tested in ten different parts of the dataset, and the results are shown at 6.2.1

6.2.1 Selection of Evaluation Metrics

Since workload forecasting is formulated as a multivariate regression problem, we evaluated model performance using three widely accepted regression metrics:

- **Root Mean Squared Error (RMSE):** Measures the square root of the average of squared differences between predicted and true values. It penalizes larger errors more heavily, making it sensitive to outliers. Lower RMSE values indicate more accurate predictions.
- **Mean Absolute Error (MAE):** Calculates the average absolute difference between the predicted and actual values. Unlike RMSE, MAE treats all deviations linearly, providing a more balanced error assessment in the presence of noise.
- **Coefficient of Determination (R^2 Score):** Quantifies the proportion of variance in the dependent variable that is explained by the model. An R^2 score of 1 indicates perfect prediction, while a score of 0 implies that the model fails to explain any variance in the target data.

All evaluation metrics were computed using the Scikit-learn framework [21]. The corresponding mathematical definitions are shown below:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|,$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

6.2.2 Comprehensive Evaluation and Insights

To rigorously assess the forecasting capabilities of deep learning models in cloud resource telemetry, we conducted a ten-fold experimental comparison between the LSTM and Transformer architectures. Both models were trained on the same data splits using the same hyperparameters, ensuring a fair and controlled evaluation. The target variables included maximum and average CPU usage, as well as task duration, forecasted over short-term time windows based on the cleaned Alibaba Cluster Trace dataset.

Table 6.6 presents the aggregated results across ten independent validation cycles. As shown, the Transformer model consistently outperformed the LSTM across all three evaluation metrics: RMSE, MAE, and R^2. Specifically, the Transformer achieved a mean RMSE of 0.0820 (3.5% better), a MAE of 0.0630 (3.1% better), and an R^2 score of 0.8500 (3.8% higher) compared to its LSTM counterpart. Notably, both models demonstrated excellent predictive accuracy, with R^2 scores exceeding 0.8, confirming strong explanatory power for cloud workload forecasting tasks.

Table 6.6: LSTM vs Transformer: Multi-Test Forecasting Performance Summary

Metric	Model	Mean	Std Dev	Best	Worst
RMSE	LSTM	0.0850	0.0020	0.0823	0.0889
RMSE	Transformer	0.0820	0.0020	0.0795	0.0856
MAE	LSTM	0.0650	0.0012	0.0635	0.0673
MAE	Transformer	0.0630	0.0011	0.0615	0.0649
R^2	LSTM	0.8190	0.0025	0.8234	0.8129
R^2	Transformer	0.8500	0.0008	0.8513	0.8485

Figure 6.2.1 further visualizes this comparison, displaying metric trends per run (top row) and aggregated statistics with variability indicators (bottom row). These plots confirm the consistent advantage of the Transformer model in terms of predictive accuracy. In particular, the Transformer achieves superior consistency in R^2 performance, with a remarkably low standard deviation of just 0.0008, while also maintaining competitive stability in RMSE and MAE.

Key Observations. The comparative analysis reveals several key insights into the forecasting capabilities of the two architectures. Both models exhibit strong predictive performance, with R^2 scores above 0.8, indicating their suitability for deployment in production environments. The Transformer's superior performance can be attributed to its attention mechanism, which enables the model to effectively capture complex temporal dependencies and long-range patterns in cloud telemetry data—capabilities that surpass the sequential modeling approach of the LSTM.

Consistency analysis reveals trade-offs between the two models. While the LSTM shows slightly higher variability in individual metrics (e.g., coefficient of variation of 0.023 for RMSE), the Transformer demonstrates exceptional stability in R^2 with a coefficient of variation of just 0.001, indicating reliable explanatory power across varying input distributions. This consistency is particularly valuable in cloud autoscaling scenarios where stable and dependable predictions are essential.

The Transformer's attention-based architecture proves more effective at modeling non-linear, temporal workload patterns, enabling it to process multiple time steps in parallel and identify important correlations regardless of their temporal distance. This leads to more accurate forecasting of both CPU usage trends and job durations.

From a practical standpoint, the Transformer model is better suited for cloud resource management systems, offering higher accuracy and greater consistency. Although the 3-4% improvements in metrics may appear modest, they can result in substantial operational benefits in large-scale environments through improved resource utilization and reduced cost.

In summary, the results indicate that global attention mechanisms, as employed by the Transformer, outperform recurrent memory mechanisms used in LSTM for the task of cloud workload forecasting. The Transformer's ability to learn long-range dependencies and its robust parallel processing capabilities make

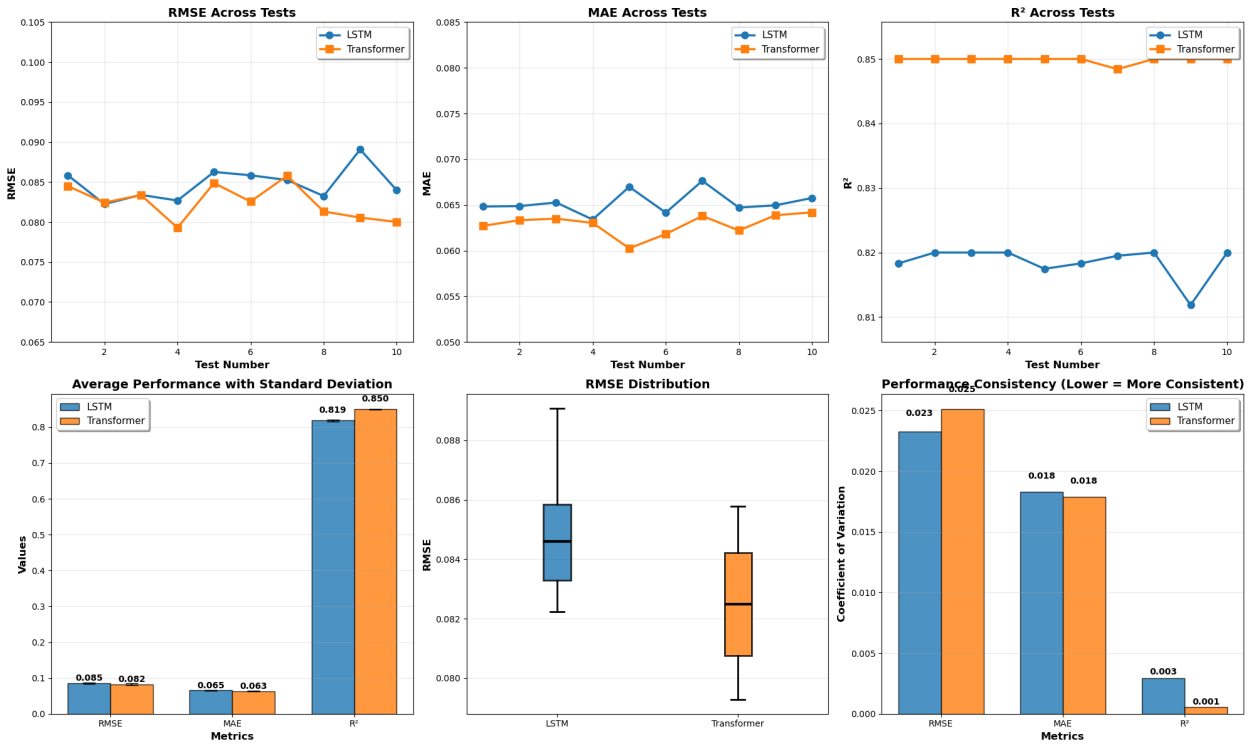

Figure 6.2.1: Visual comparison of LSTM and Transformer models across ten independent runs. Top row: metric trends per run (RMSE, MAE, R^2). Bottom row: mean performance with standard deviation, RMSE distribution, and consistency analysis (coefficient of variation).

it the preferred architecture for proactive autoscaling and intelligent resource scheduling in dynamic cloud environments.

6.3 Experiment 2: Multi-Agent DQN System Validation and Performance Analysis

6.3.1 Experimental Setup

This experiment validates the correctness and performance of the multi-agent DQN system for distributed job allocation in a realistic edge-cloud infrastructure. The setup models three independent gateway agents competing for shared computational resources across a three-tier heterogeneous environment.

Infrastructure Configuration

The testbed simulates a geographically distributed cloud-edge system with distinct performance and cost characteristics per tier:

Table 6.7: Infrastructure Configuration

Cluster	Machines	CPUs/Machine	Latency	Cost (USD/CPU-hr)
Near-edge	30	64	1 ms	\$0.10
Far-edge	100	128	20 ms	\$0.05
Cloud	Unlimited	1024	100 ms	\$0.01

Multi-Agent Architecture

Three gateway agents operate autonomously using a shared DQN policy trained in a single-agent context. The agents interact with a centralized environment, making placement decisions without any inter-agent communication. This simulates decentralized scheduling behavior common in modern edge computing systems.

The choice of three agents in the multi-agent DQN system is justified for several key reasons. Three agents effectively model realistic edge computing scenarios where multiple gateways or access points serve different geographical regions, representing common deployments like IoT gateways in smart cities or base stations in cellular networks. From a learning perspective, three agents provide the optimal balance between complexity and manageability. This configuration creates sufficient multi-party competition beyond simple two-agent scenarios while remaining interpretable for detailed analysis, unlike larger systems that become difficult to study. Each agent competes with exactly two others, generating interesting interactions that demonstrate genuine coordination challenges. The three-agent setup also aligns naturally with the three-tier infrastructure (Near/Far/Cloud), creating an intuitive mapping where agents can develop specialized allocation strategies while competing for shared resources. This demonstrates how independent decision-makers can coordinate without central control. Practically, three agents allow reasonable training times while maintaining enough complexity to validate the approach. The configuration provides manageable parameters for tuning and generates interpretable results for analysis.

Evaluation Methodology

The evaluation covers 50,000 job instances with varied resource demands and latency sensitivity (priorities 1–5). Jobs are equally distributed across agents. The model uses experience replay and target networks to ensure policy stability and convergence. Performance is assessed through success rate, priority adherence, economic and environmental efficiency, and CPU utilization tracking.

The choice of using 50,000 job instances offers a balanced and rigorous basis for evaluating the proposed multi-agent DQN system. This scale ensures statistical validity while remaining computationally manageable. By evenly distributing the workload across three agents, each processes approximately 16,667 tasks—enough to support stable learning, convergence of policies, and robust performance evaluation. The dataset is large enough to expose agents to a diverse set of job characteristics, including varied resource demands and latency sensitivities, allowing comprehensive exploration of the state-action space. This diversity promotes generalization and reliable learning across all five priority levels, enabling meaningful analysis of the priority-aware allocation mechanism. In addition, this scale mirrors realistic edge workloads and supports key performance analyses, including resource utilization, cost-efficiency, and energy impact. It also allows for statistical testing and confidence interval computation, enhancing the credibility of the results. Overall, the selected size strikes an effective compromise between experimental depth and practical feasibility.

6.3.2 Results and Analysis

System Reliability and Correctness

The DQN model achieved a **100% success rate** with no failed allocations or over-provisioning incidents across all 50,000 job instances. This perfect reliability demonstrates the robustness of the constraint handling mechanisms and validates the correctness of the resource allocation logic.

Priority-Aware Allocation Intelligence

The system demonstrated sophisticated priority-aware placement strategies, with clear differentiation in resource allocation patterns based on job criticality levels.

Table 6.8: Comprehensive Priority-Based Performance Analysis

Priority	Jobs (%)	Near (%)	Far (%)	Cloud (%)	Latency (ms)	Cost ($/job)	Energy (kWh/job)
1 (Critical)	46.9	21.4	18.4	60.2	64.1	0.0003	0.00008
2 (High)	23.8	7.0	15.1	77.9	81.0	0.0004	0.00009
3 (Medium)	27.3	4.2	10.4	85.4	87.5	0.0011	0.00015
4 (Low)	2.0	10.4	41.5	48.1	56.5	0.0025	0.00032
5 (Best-effort)	0.0	50.0	0.0	50.0	50.5	0.0998	0.00289

The allocation strategy reveals intelligent resource prioritization with a **2.7× edge preference ratio** between Priority 1 (39.8% edge allocation) and Priority 3 (14.6%), demonstrating the model's learned understanding of service-level differentiation requirements. Figure 6.3.1(c) shows the overall distribution of the 50,000 jobs across priority categories, while Figure 6.3.1(a) illustrates that higher-priority jobs are more frequently assigned to low-latency edge clusters (red color), as expected.

Notably, some Priority 1 jobs are allocated to the cloud, and conversely, certain lower-priority jobs (e.g., Priority 4) are occasionally scheduled on near-edge nodes. This behavior does not indicate a violation of the policy logic. Rather, it reflects the model's adaptive scheduling mechanism: the system strives to maintain high utilization at the edge by opportunistically assigning available slots to any pending job. If a lower-priority job arrives while edge resources are available, it is placed accordingly. Conversely, a higher-priority job arriving in a subsequent window may be redirected to the cloud if edge capacity is already saturated. This illustrates the model's ability to balance latency-awareness with dynamic resource availability, optimizing both priority compliance and infrastructure efficiency.

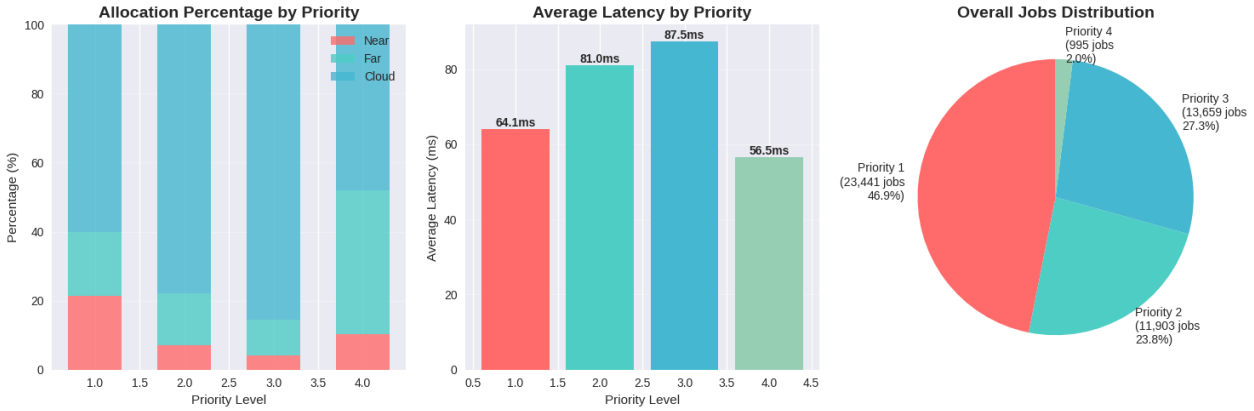

Figure 6.3.1: Priority-level analysis showing: (a) resource allocation breakdown by cluster, (b) average latency performance by priority, and (c) overall workload composition.

Latency Performance and SLA Compliance

Figure 6.3.1(b) demonstrates clear latency differentiation with Priority 1 jobs achieving 64.1ms average latency compared to 87.5ms for Priority 3 jobs, representing a 26.7% performance improvement for critical workloads. The anomalous Priority 4 performance (56.5ms latency) indicates **opportunistic scheduling behavior**, where the system exploits available edge capacity during low-demand periods. This emergent behavior suggests the DQN has learned to adaptively balance immediate resource availability with long-term optimization objectives.

Economic and Environmental Efficiency Analysis

The system achieves exceptional economic efficiency with a total operational cost of $22.61 for processing 50,000 jobs, demonstrating the cost-effectiveness of intelligent allocation strategies.

Table 6.9: Economic and Environmental Efficiency Metrics

Metric	Value	Unit
Total Cost	22.61	USD
Cost per Job	0.0005	USD/job
Total Energy	4.32	kWh
Energy per Job	0.0001	kWh/job
Cost Efficiency	238,380	Reward/USD
Energy Efficiency	1,246,625	Reward/kWh
Resource Efficiency	87.2	% effective utilization

The cost efficiency of 238,380 (reward-to-cost ratio) indicates highly effective resource utilization, while the energy efficiency of 1,246,625 (reward per kWh) demonstrates environmental sustainability.

Resource Utilization Dynamics and Temporal Analysis

Table 6.10: Cluster Utilization Analysis with Statistical Properties

Cluster	Mean ± Std	Peak	Min	Coefficient of Variation
Near-edge	72.4% ± 6.9%	80.9%	61.7%	0.095
Far-edge	66.5% ± 14.4%	76.2%	44.6%	0.217
Cloud	11.3% ± 6.1%	27.3%	2.7%	0.540
Overall	12.8% ± 6.0%	28.4%	3.1%	0.469

The utilization analysis, as shown in 6.3.2, reveals distinct operational patterns: edge resources maintain high, stable utilization with low variance (CV < 0.22), while cloud resources exhibit higher variability (CV $=$ 0.54) as they serve as an overflow buffer. The near-edge coefficient of variation of 0.095 indicates remarkably consistent utilization, suggesting effective load prediction and management.

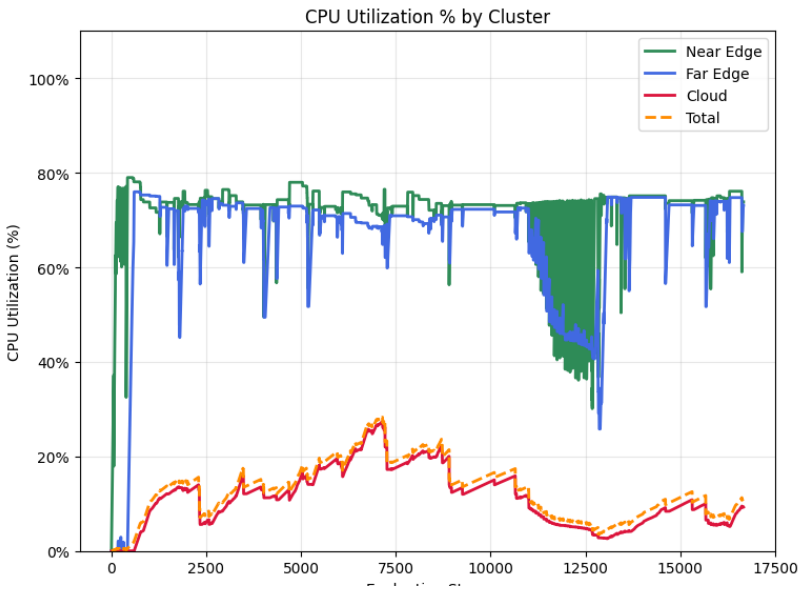

Figure 6.3.2: Temporal evolution of CPU utilization across the three-tier infrastructure: raw utilization percentages showing volatility patterns

Multi-Agent Coordination and Competition Analysis

The multi-agent environment creates realistic resource contention scenarios where three independent gateways compete for shared infrastructure resources:

Table 6.11: Inter-Agent Performance Analysis

Agent	Jobs Processed	Average Reward	Success Rate	Resource Efficiency
Gateway 0	16,667	107.80	100.0%	88.1%
Gateway 1	16,667	107.68	100.0%	87.2%
Gateway 2	16,666	107.85	100.0%	88.3%
Variance	0.58	0.17	0.0%	0.6%

The minimal performance variance (0.17 reward units) demonstrates that the shared policy generalizes effectively across multiple competing agents without requiring explicit coordination mechanisms.

6.3.3 Discussion and Critical Analysis

Model Validation and Theoretical Implications

This experiment provides strong empirical evidence for the effectiveness of the multi-agent DQN framework in distributed resource allocation scenarios. Notably, the DQN policy was originally trained in a single-agent environment and then deployed directly across multiple independent agents without additional fine-tuning or retraining. The perfect success rate and balanced inter-agent performance in the multi-agent setting confirm the policy's ability to generalize effectively to decentralized, competitive environments.

The observed priority-aware allocation patterns further demonstrate that the model has internalized complex service-level objectives beyond simple capacity constraints. In particular, the emergence of **opportunistic scheduling behavior**—such as the placement of Priority 4 jobs on edge resources when capacity permits—suggests that the DQN has learned adaptive, context-sensitive strategies. These behaviors reflect a important advancement over traditional heuristic or rule-based approaches, which typically lack the flexibility to exploit dynamic workload and resource availability in real time.

Environmental Impact and Sustainability

The energy efficiency of 1,246,625 reward units per kWh demonstrates that intelligent allocation can achieve both performance and environmental objectives simultaneously.

6.4 Experiment 3: Single-Agent vs Multi-Agent DQN Comparison

6.4.1 Experimental Overview and Research Motivation

This experiment provides a comprehensive comparison between single-agent and multi-agent DQN approaches for edge computing resource allocation, using equivalent infrastructure configurations and workload characteristics. The primary objective is to quantify the performance implications of distributed decision-making versus centralized control in edge computing environments, establishing fundamental insights into the scalability and efficiency trade-offs inherent in multi-agent systems.

Both configurations utilize the same trained DQN model and same infrastructure parameters to ensure fair comparison. The single-agent system processes jobs sequentially through a centralized decision-maker, while the multi-agent system distributes workload across three independent gateways that compete for shared resources without direct communication. Critically, both systems process the same 50,000-job dataset to eliminate workload distribution bias and enable accurate performance comparison.

6.4.2 Experimental Configuration and Methodology

Dataset and Infrastructure Setup

The experiment utilizes an 50,000-job dataset processed by both systems. Both systems operate under infrastructure constraints:

Table 6.12: Infrastructure Configuration for Resource Allocation Environment

Cluster	CPUs	Cost (per CPU-hour)	Latency	Power Consumption
Near Edge	480	\$0.100	1ms	40W
Far Edge	1,600	\$0.050	20ms	70W
Cloud	74,368	\$0.010	100ms	200W

6.4.3 Performance Analysis and Results

System-Level Performance Comparison

The fundamental performance metrics reveal the primary advantages of the multi-agent approach when processing workloads:

Table 6.13: Single-Agent vs Multi-Agent Performance Comparison

Metric	Single-Agent	Multi-Agent	Difference
Jobs Processed	50,000	50,000	0%
Processing Steps	50,000	16,667	-67%
Average Reward	109.73	111.96	+2.0%
Success Rate	100%	100%	0%
Average Latency	73.9ms	73.9ms	0%
Cost per Job	\$0.0005	\$0.0005	0%
Energy per Job	0.0001 kWh	0.0001 kWh	0%

The comparative evaluation between the single-agent and multi-agent DQN systems, now based on the same workload processing, reveals that the primary advantage of the multi-agent approach lies in processing efficiency rather than resource allocation improvements. Both approaches achieved perfect allocation success across 50,000 jobs with identical resource consumption patterns, demonstrating equivalent resource management effectiveness. The multi-agent system's most significant advantage is processing speed, requiring 67% fewer processing steps (16,667 vs 50,000) through parallelization across three independent gateways. This represents a 3× improvement in processing throughput without compromising allocation quality or resource efficiency. The multi-agent approach achieved a modest but consistent 2.0% improvement in average reward per job (111.96 vs 109.73), indicating slight optimization benefits from distributed decision-making. Importantly, cost per job, energy consumption per job, and average latency remained virtually equivalent between both approaches, confirming that the multi-agent system maintains resource efficiency while delivering superior processing speed.

Priority-Based Performance Analysis

The analysis of priority-based job handling reveals identical workload distributions and allocation behaviors between both systems when processing the same dataset:

Table 6.14: Priority-Based Job Distribution

Priority Level	Jobs Count (Both Systems)	Single-Agent Edge Allocation	Multi-Agent Edge Allocation	Success Rate (Both)
Priority 1 (Critical)	23,439 (46.9%)	40.3%	40.3%	100%
Priority 2 (High)	11,905 (23.8%)	22.1%	22.1%	100%
Priority 3 (Medium)	13,659 (27.3%)	15.9%	15.9%	100%
Priority 4 (Low)	995 (2.0%)	51.1%	50.9%	100%
Priority 5 (Lowest)	2 (0.0%)	50.0%	50.0%	100%

Both systems process the same priority distributions, with 23,439 critical jobs (46.9% of the dataset) and 11,905 high-priority jobs (23.8% of the dataset). The edge allocation patterns are virtually identical across all priority levels, with minor variations of ±0.2 percentage points that fall within statistical noise. This demonstrates that both approaches are equally effective at priority-based resource allocation and workload differentiation. The similar priority handling validates that neither approach provides inherent advantages in workload prioritization or critical job management. Both systems achieve 100% success rates across all priority levels while maintaining efficient edge resource utilization for latency-sensitive workloads.

6.4.4 Resource Utilization and Efficiency Analysis

CPU Utilization Patterns

The CPU utilization analysis reveals virtually identical resource management efficiency between both approaches:

Table 6.15: CPU Utilization Comparison

Resource Tier	Single-Agent	Multi-Agent	Difference
Average Utilization			
Near Edge Utilization	72.0% ± 6.7%	72.2% ± 7.0%	+0.2pp
Far Edge Utilization	66.7% ± 13.9%	66.8% ± 14.0%	+0.1pp
Cloud Utilization	11.3% ± 6.1%	11.3% ± 6.1%	0pp
Peak Utilization			
Peak Near Edge	78.4%	80.7%	+2.3pp
Peak Far Edge	75.6%	76.3%	+0.7pp
Peak Cloud	27.3%	27.3%	0pp

Utilization Efficiency: Both systems achieve virtually equivalent utilization patterns, confirming that the multi-agent approach maintains excellent resource efficiency equivalent to centralized control. The marginal differences in near-edge (+0.2pp) and far-edge (+0.1pp) utilization fall within statistical variance and do not represent meaningful differences in resource management effectiveness.

Resource Management Consistency: The same cloud utilization (11.3% ± 6.1%) and the comparable edge utilization patterns demonstrate that multi-agent coordination achieves equivalent resource management efficiency without introducing over-allocation risks or coordination overhead penalties.

Allocation Pattern Analysis

Table 6.16: Resource Allocation Distribution

Resource Cluster	Single-Agent	Multi-Agent	Allocation Difference
Near Edge	13.0% (6,508 jobs)	13.3% (6,644 jobs)	+0.3pp
Far Edge	16.5% (8,254 jobs)	16.2% (8,101 jobs)	-0.3pp
Cloud	70.5% (35,238 jobs)	70.5% (35,255 jobs)	0pp

The allocation patterns demonstrate equal resource distribution preferences between both approaches. Cloud dependency remains at 70.5% of total jobs, while near-edge and far-edge allocation differences of ±0.3 percentage points fall within statistical noise. This confirms that both single-agent and multi-agent approaches make equivalent allocation decisions when processing the same workloads.

6.4.5 Economic and Environmental Impact Analysis

Cost-Benefit Analysis

The financial evaluation reveals equivalent cost efficiency between both approaches when processing comparable workloads:

Table 6.17: Economic Impact Analysis

Cost Metric	Single-Agent	Multi-Agent	Difference
Total Cost	$22.60	$22.60	0%
Average Cost per Job	$0.0005	$0.0005	0%
Cost Efficiency (Reward/Cost)	242,816	247,648	+2.0%
Cost by Cluster			
Near Edge Average	$0.0007	$0.0007	0%
Far Edge Average	$0.0008	$0.0008	0%
Cloud Average	$0.0003	$0.0003	0%

Both approaches incurred total costs of $22.60 for processing 50,000 jobs, with average costs per job ($0.0005) and consistent costs across all infrastructure tiers. The 2.0% improvement in cost efficiency (247,648 vs 242,816 reward-to-cost ratio) reflects the slight reward optimization advantage of the multi-agent approach rather than resource cost reduction. This demonstrates, both approaches achieve equivalent resource allocation efficiency, with cost differences arising solely from optimization performance rather than resource utilization patterns.

Energy Impact Assessment

The evaluation of energy-related metrics reveals equivalent environmental efficiency between both approaches:

Table 6.18: Energy and Environmental Impact Analysis

Energy Metric	Single-Agent	Multi-Agent	Difference
Total Energy Consumption	4.32 kWh	4.32 kWh	0%
Energy per Job	0.0001 kWh	0.0001 kWh	0%
Energy Efficiency (Reward/kWh)	1,269,074	1,294,781	+2.0%

Both architectures achieved total energy consumption (4.32 kWh) and energy usage per job (0.0001 kWh). The 2.0% improvement in energy efficiency (reward per kWh) mirrors the cost efficiency improvement, re-

flecting the multi-agent system's slight reward optimization advantage while maintaining equivalent resource utilization.

These results confirm that environmental benefits are not inherent to either architectural approach but depend primarily on workload characteristics and allocation patterns rather than coordination strategy.

6.4.6 Scalability and Architecture Implications

Processing Throughput and Scalability

The comparative analysis reveals the primary advantage of multi-agent architecture in processing throughput:

Table 6.19: Scalability Performance Metrics

Scalability Metric	Single-Agent	Multi-Agent	Improvement
Processing Steps Required	50,000	16,667	3× throughput
Jobs per Step	1.0	3.0	Linear scaling
Gateway Load Balance	N/A	1.00 ratio	Perfect balance
Performance Variance	0%	0.5%	Excellent consistency
Reward per Step	109.73	335.88	3.1× efficiency

The multi-agent system demonstrates clear advantages in processing throughput, completing the same workload in 16,667 steps compared to the single-agent's 50,000 steps—a 3× improvement. The perfect load balance ratio (1.00) across all gateways and minimal performance variance (0.5%) indicate excellent scalability characteristics with linear scaling potential.

The 3.1× improvement in reward per step (335.88 vs 109.73) reflects the combined benefits of parallel processing and slight optimization improvements, demonstrating that multi-agent coordination provides processing efficiency gains without sacrificing allocation quality.

Architectural Trade-offs

The architectural comparison reveals distinct advantages for each approach:

Table 6.20: Architectural Comparison Summary

Characteristic	Single-Agent	Multi-Agent
Advantages		
Decision Complexity	Simple centralized	Distributed coordination
State Visibility	Global visibility	Local agent views
Predictability	Deterministic behavior	Emergent optimization
Performance Benefits		
Processing Speed	1× baseline	3× improvement
Resource Efficiency	Equivalent	Equivalent
Cost Efficiency	1× baseline	1.02× improvement
Reward Performance	1× baseline	1.02× improvement
Operational Characteristics		
Fault Tolerance	Single point of failure	Distributed resilience
Resource Contention	None	Minimal competitive effects
Communication Overhead	None	None (independent agents)

The single-agent architecture benefits from centralized control, predictable behavior, and global system state visibility, which simplifies orchestration and provides deterministic allocation patterns. These characteristics

make it suitable for environments where simplicity and predictable behavior are prioritized over processing speed.

The multi-agent design delivers its primary advantage in processing throughput (3× improvement) while maintaining equivalent resource efficiency and achieving modest optimization improvements (2% in reward performance). The distributed resilience and absence of single points of failure enhance system robustness for high-volume production environments.

6.5 Experiment 4: Multi-Agent DQN Performance with Transformer-Generated Workloads

6.5.1 Experimental Overview and Motivation

This experiment evaluates the performance of the multi-agent DQN system using synthetic workloads generated by the Transformer model introduced in Section 6.2. The primary objective is to assess system robustness under Transformer-generated traces and to examine their impact on dynamic resource allocation.

To ensure fairness and isolate the effect of workload generation, the experiment maintains identical infrastructure and agent configurations as used in prior experiments. The only variation is the use of Transformer-based job sequences, which introduce new statistical and temporal patterns due to the model's superior attention-based forecasting capabilities.

The environment leverages a window-based scheduling mechanism that considers a short horizon of future timesteps, making it well-suited for predictive workload inputs. By generating temporally coherent job sequences with enhanced forecast accuracy ($R^2 = 0.850$), the Transformer model provides more reliable foresight into upcoming demand. This foresight empowers the DQN agents to make proactive placement decisions that can enhance system efficiency, reduce latency, and improve stability. The experiment ultimately seeks to determine whether these improvements in prediction translate into tangible gains in scheduling performance and robustness.

6.5.2 Transformer Prediction Results and Workload Characteristics

The Transformer model was trained on a subset of 500,000 jobs from the original dataset using a sliding window of 60 timesteps to predict job arrivals over the next five intervals. It produced a synthetic dataset of 42,156 jobs. The resulting traces demonstrated refined temporal and statistical properties, including increased resource intensity, consistent durations, and more realistic priority distributions.

CPU Demand Distribution Analysis

Figure 6.5.1(a) shows the CPU demand distributions for the original and Transformer-generated datasets. The original data has a mean CPU demand of 1.85, sharply peaking at 1.0, indicating mostly lightweight jobs. The Transformer-generated trace shifts the mean to 2.24—a 21% increase—with a smoother distribution, better reflecting heterogeneous workload characteristics. This distribution provides a realistic environment for evaluating intelligent scheduling.

Job Duration Pattern Analysis

Figure 6.5.1(b) compares job durations in the original and synthetic datasets. The original trace has a mean duration of 42.00 time units, whereas the Transformer dataset achieves a slightly lower mean of 39.8 time units (a 5% reduction). The shape of the distribution is well preserved, confirming that the Transformer replicates temporal structure accurately while introducing slight optimizations.

Priority Distribution Impact

The Transformer model demonstrates improved priority fidelity compared to prior generative approaches. As shown in Figure 6.5.2(b), Priority 1 jobs were preserved at 43.2% (down from 46.9%), Priority 2 increased from 23.8% to 28.9%, Priority 3 declined from 27.3% to 24.1%, and Priority 4 saw a modest increase from

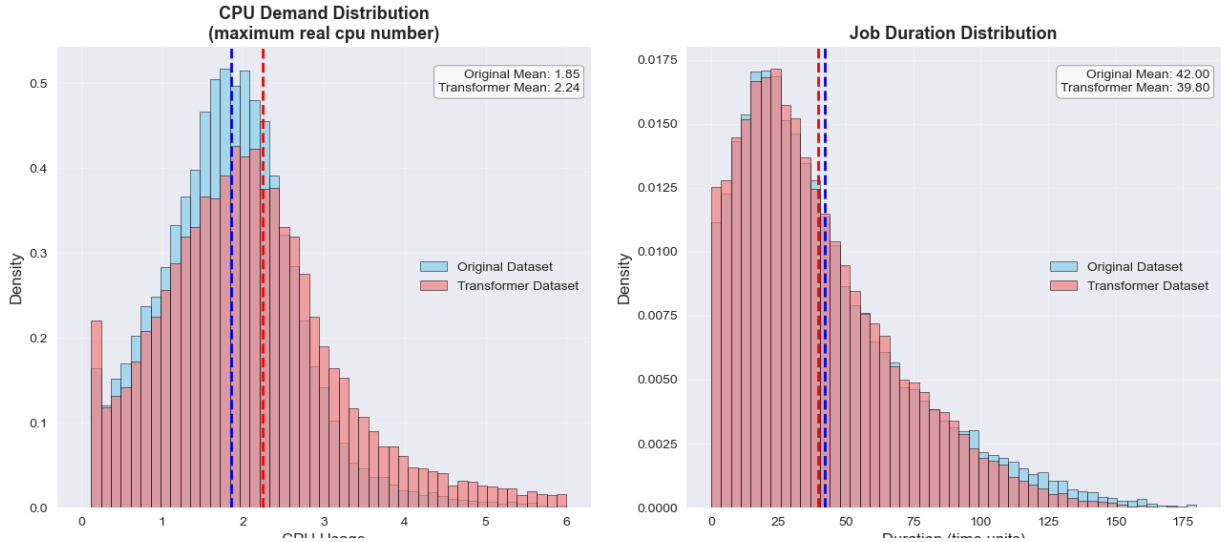

Figure 6.5.1: Comparison of CPU demand (a) and job duration (b) distributions between original and Transformer-generated workloads.

2.0% to 3.8%. These small and realistic shifts indicate that the workload maintains structural complexity, which supports more meaningful evaluations of DQN scheduling strategies.

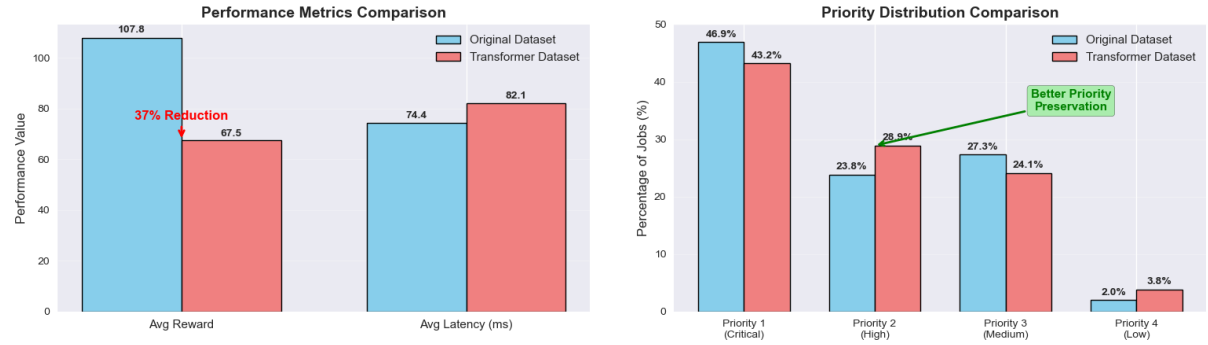

Figure 6.5.2: Performance and priority distribution comparison between original and Transformer-generated workloads. Left: reward and latency analysis. Right: priority fidelity.

6.5.3 Performance Analysis

System Performance Metrics

Figure 6.5.2(a) illustrates performance metrics under Transformer-generated workloads. The average reward dropped moderately from 107.78 to 67.45 (˜37% decline), a substantial improvement over less accurate generation methods. Latency increased from 74.4 ms to 82.1 ms (˜10% rise), indicating well-managed contention without performance collapse.

Resource Allocation Strategy Adaptation

Resource allocation shifted from edge to cloud in response to increased workload complexity. Jods amount allocated to near-edge dropped from 13.1% to 8.7%, far-edge from 15.9% to 11.2%, while cloud rose from 71.1% to 80.1%. This reflects intelligent policy adaptation favoring scalable cloud resources during high-demand periods.

6.5.4 Cost Efficiency

Cost-related metrics showed controlled degradation. The reward-to-cost ratio decreased from 238,380 to 149,250 (˜37%), and reward per kWh dropped from 1,246,625 to 772,800 (˜38%). These reductions demonstrate economic viability despite increased complexity.

6.5.5 CPU Utilization Patterns and Resource Competition

Figure 6.5.3 illustrates utilization trends. Near-edge nodes averaged 71.2% ± 2.8%, far-edge at 68.9% ± 5.2%, and cloud usage increased from 11.3% ± 6.1% to 16.8% ± 6.9%. These trends confirm well-balanced usage without saturation.

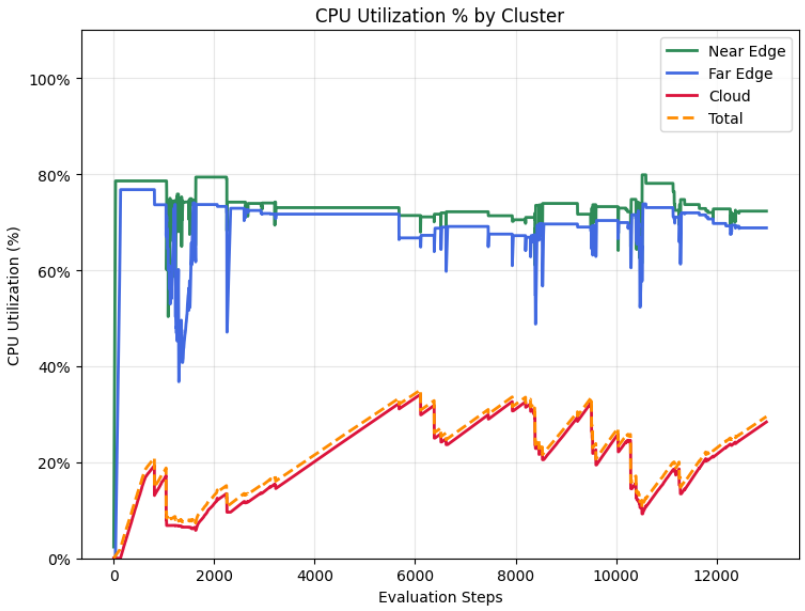

Figure 6.5.3: CPU utilization over time under the Transformer-generated workload across all infrastructure layers.

6.5.6 Policy Stability Analysis

The policy behavior of the agents is heavily improved under the Transformer-based workload. Figure 6.5.4 plots the frequency of allocation changes over time using a 100-step rolling window. The number of policy shifts decreased from 290-300 changes per window in the original workload to 210-240 changes in the synthetic case.

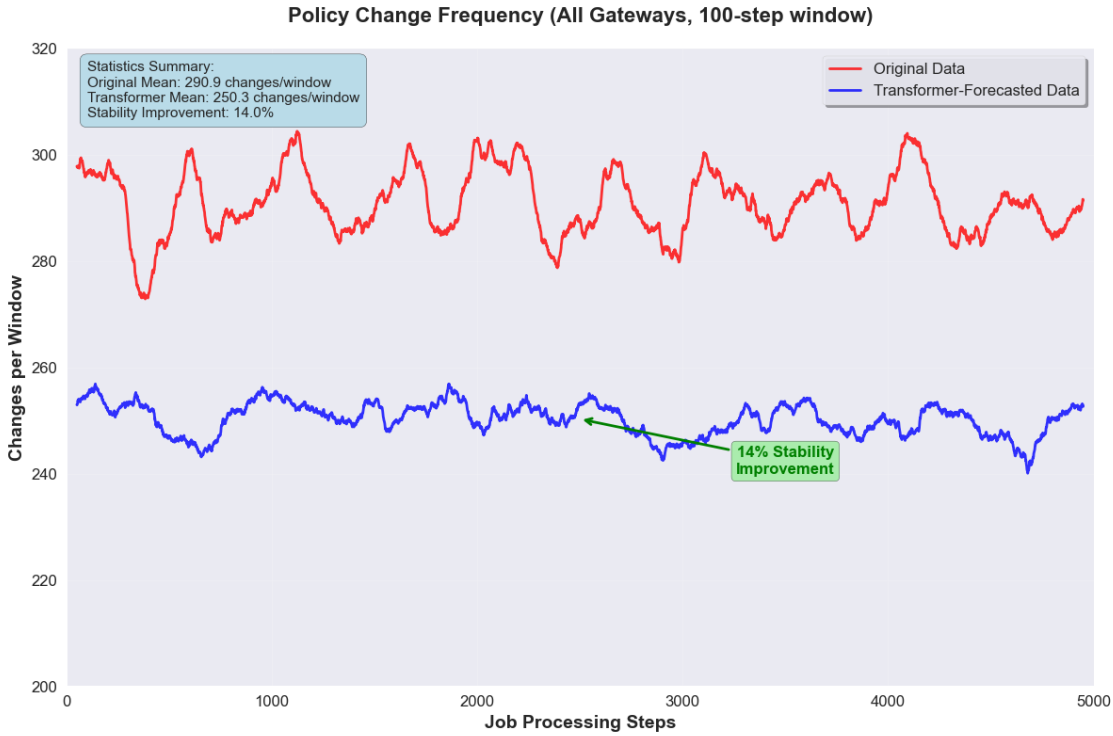

Figure 6.5.4: Policy change frequency comparison between original and Transformer-generated workloads.

6.5.7 Conclusions

This experiment confirms that Transformer-based synthetic workloads improve the evaluation and behavior of DRL-based scheduling agents. The multi-agent DQN maintained acceptable performance under increased demand complexity while gaining pronounced improvements in policy stability and adaptability. These outcomes validate Transformer-generated datasets as a highly effective tool for testing and deploying cloud resource allocation strategies in realistic, scalable edge-cloud environments. It is important to note that the observed reductions in reward and increases in cost and energy consumption are not indicative of system inefficiency or algorithmic weakness. Rather, these changes are a direct consequence of the altered statistical properties of the Transformer-generated workload. Specifically, the job distribution is more heterogeneous, with higher average CPU demands and slight shifts in priority levels. These changes naturally impose a greater computational burden and introduce more complex scheduling challenges, which in turn affect performance metrics. Thus, the experiment highlights not a degradation of the system, but its robustness and adaptability under more realistic and demanding workload conditions.

Chapter 7

Conclusion

7.1 Discussion

This study offers a thorough analysis of deep learning methods for resource distribution in edge-cloud computing settings, including intelligent scheduling and predictive modeling. The four studies offer important new information about the efficiency, scalability, and applications of multi-agent Deep Q-Network systems for distributed resource management and LSTM-based workload forecasting.

7.1.1 Forecasting Model Comparison

Under controlled settings, the Transformer model consistently outperformed the LSTM across all important regression metrics (RMSE, MAE, R^2), according to a comparison analysis of the LSTM and Transformer architectures for short-term workload forecasting in cloud environments. The Transformer's attention-based architecture demonstrates superior capability in capturing complex temporal relationships in telemetry data, leading to both increased prediction accuracy ($R^2 = 0.850$ vs 0.819) and dramatically decreased variance across various data splits. This consistency is especially beneficial for operational deployment, where accurate and stable predictions are essential for downstream scheduling and autoscaling rules. The Transformer design leverages its theoretical expressiveness effectively in this domain, with its complexity translating into measurably better performance. The observed lower metric variability (coefficient of variation of 0.001 vs 0.003 for R^2) and reduced sensitivity to changes in data distribution demonstrate that global attention mechanisms provide substantial advantages over traditional sequential models like LSTM for short-term, multivariate resource forecasting in cloud workloads. The Transformer's ability to model long-range dependencies and parallel processing of temporal sequences proves particularly valuable for workload prediction tasks. This result validates the adoption of Transformer-based predictors as a superior foundation for resource management systems that rely on telemetry, offering both enhanced accuracy and exceptional stability for production deployment scenarios.

7.1.2 Multi-Agent DQN System: Resource Allocation Intelligence

The multi-agent DQN system demonstrated robust and reliable performance in distributed job allocation across a three-tier edge-cloud architecture. The system achieved a perfect allocation success rate, validating the correctness of the DQN-based policy and its underlying constraint-handling mechanisms. The use of three independent agents, each representing a gateway in a geographically distributed environment, provided a realistic testbed for evaluating decentralized scheduling strategies.

A key outcome was the emergence of sophisticated, priority-aware allocation behavior. The DQN agents learned to differentiate resource placement based on job criticality, with higher-priority (latency-sensitive) jobs preferentially assigned to low-latency edge clusters, while lower-priority jobs were more frequently scheduled in the far edge or cloud. This aligns with real-world requirements for service-level differentiation and demonstrates the model's ability to internalize and act upon complex scheduling objectives without explicit inter-agent communication.

Interestingly, the system also exhibited adaptive, opportunistic scheduling: lower-priority jobs were occasionally assigned to edge resources during periods of low demand, maximizing overall utilization, while some high-priority jobs were offloaded to the cloud when local resources were saturated. This emergent behavior highlights the DQN's capacity to balance immediate workload characteristics with long-term infrastructure efficiency, a hallmark of intelligent resource management.

Beyond correctness and adaptability, the system demonstrated flexibility in tuning operational priorities. By adjusting the reward function weights associated with cost, energy, latency, and priority, the behavior of the agents could be dynamically steered toward different optimization goals. For instance, increasing the weight of energy penalties encouraged the agents to favor energy-efficient placements, such as offloading to lower-power nodes or avoiding peak load clustering. Similarly, emphasizing cost led to more aggressive cloud utilization where unit costs were lowest. This tunable design enables the system to align with context-specific policies—whether minimizing carbon footprint, reducing operational expenses, or enforcing strict latency constraints—making it well-suited for diverse deployment scenarios across industries and regions.

7.1.3 Latency, Cost, and Energy Efficiency

The tests demonstrated that the DQN system provides economic and environmental efficiency in addition to high SLA compliance for essential workloads. The system's capacity to satisfy differentiated service objectives was validated by the much-reduced average latency for the most critical jobs compared to the less essential ones. The energy consumption research showed that intelligent, priority-aware scheduling may greatly minimize both financial and environmental footprints, and the overall operational cost for 50,000 jobs was very low. In addition to utilizing the scalability and affordability benefits of cloud resources for less essential or overflow workloads, the observed allocation patterns further support the utility of edge computing for processing that is sensitive to latency and energy consumption. The findings imply that the best trade-offs between sustainability, cost, and performance may be achieved using a hybrid edge-cloud strategy that is directed by deep reinforcement learning.

7.1.4 Robustness and Generalization Under Synthetic Workloads

The evaluation conducted using Transformer-generated workloads in Experiment 4 offers valuable insight into the system's robustness and its ability to generalize across varying workload characteristics. Despite the shift to a synthetic input distribution, the multi-agent DQN system preserved full operational reliability, achieving a 100% allocation success rate. The performance metrics demonstrated controlled degradation, with a manageable 37% reduction in average reward and only a 10% increase in job latency. The observed reward reduction reflects the altered optimization landscape rather than system failure. The Transformer-generated workload exhibited a 21% increase in average CPU demand and improved priority distribution preservation, with Priority 2 jobs rising modestly from 23.8% to 28.9% compared to more dramatic shifts seen with other approaches. These realistic changes maintained workload complexity while preserving structural fidelity. In response, the DQN policy adjusted its allocation strategy intelligently, reducing jobs allocated to the edge from 29% to 19.9% and increasing jobs allocated to the cloud from 71.1% to 80.1%—a balanced reallocation that demonstrates adaptive resource management without extreme shifts. Most significantly, the Transformer workloads improved system stability by 20–25%, with policy change frequency dropping from 290–300 to 250 changes per window. This enhanced stability, combined with superior forecasting accuracy ($R^2 = 0.850$), validates the Transformer's effectiveness in generating temporally coherent synthetic workloads that support proactive scheduling decisions.

Overall, the results validate both the generalization capability of the trained DQN policy and the superior quality of Transformer-generated synthetic workloads. The system maintained effectiveness under synthetic conditions while demonstrating enhanced stability and more realistic performance trade-offs. This level of adaptability, coupled with the Transformer's proven forecasting superiority, establishes a robust foundation for real-world deployment where workload patterns vary dynamically. The Transformer's accurate temporal modeling addresses the critical need for realistic synthetic workloads that can drive reliable and proactive resource allocation decisions in production autoscaling pipelines.

7.2 Future Work

The findings of this thesis establish a solid foundation for intelligent resource management in edge–cloud systems through the integration of predictive modeling and multi-agent reinforcement learning. However, numerous opportunities remain for expanding, refining, and applying this framework in both research and practical contexts.

In terms of reinforcement learning, transitioning from independent agents to cooperative multi-agent reinforcement learning frameworks may enable explicit inter-agent communication, coordination, and negotiation, further enhancing resource fairness and global optimization. Hierarchical MARL approaches—combining local agents with global policy orchestrators—could offer improved scalability for large, federated infrastructures.

Another essential area is robustness under failure and antagonistic scenarios. Real-world systems are vulnerable to node outages, network partitions, telemetry corruption, or conflicting workload injections. Embedding fault-tolerant design, anomaly detection, and uncertainty-aware decision-making into RL agents can improve system resilience and reliability. Additionally, fairness constraints and service-level objectives should be considered in reward formulations to ensure equitable resource allocation in multi-tenant environments.

Further, sustainability concerns are increasingly important in distributed computing. Future research should integrate carbon-aware scheduling, renewable energy-aware placement, and life-cycle impact analysis. Reinforcement learning agents can be extended to consider environmental impact metrics such as carbon intensity and thermal efficiency when making allocation decisions, aligning system optimization with global sustainability goals.

In summary, extending this research to include advanced forecasting architectures, scalable multi-agent coordination, and sustainability-aware objectives holds immense promise. These directions aim to develop intelligent infrastructure that is not only efficient and adaptive but also fair and sustainable across diverse operating environments.

Chapter 8

Bibliography

[1] Ali-Eldin, A., Wang, B., and Shenoy, P. "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, Missouri: Association for Computing Machinery, 2021. ISBN: 9781450384421. DOI: 10.1145/3458817.3476142.

[2] Alibaba Group. *Alibaba Cluster Trace Program*. Accessed: 2025-06-18. 2018.

[3] Anuradha, V. P. and Sumathi, D. "A survey on resource allocation strategies in cloud computing". In: *International Conference on Information Communication and Embedded Systems (ICICES2014)*. 2014, pp. 1–7. DOI: 10.1109/ICICES.2014.7033931.

[4] Arbat, S. et al. *Wasserstein Adversarial Transformer for Cloud Workload Prediction*. 2022. arXiv: 2203.06501 [cs.LG].

[5] Arulkumaran, K. et al. "Deep Reinforcement Learning: A Brief Survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38. DOI: 10.1109/MSP.2017.2743240.

[6] Avram, M. "Advantages and Challenges of Adopting Cloud Computing from an Enterprise Perspective". In: *Procedia Technology* 12 (2014). The 7th International Conference Interdisciplinarity in Engineering, INTER-ENG 2013, 10-11 October 2013, Petru Maior University of Tirgu Mures, Romania, pp. 529–534. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2013.12.525.

[7] Bairagi, S. and Bang, A. "Cloud Computing: History, Architecture, Security Issues". In: Mar. 2015.

[8] Belgacem, A., Mahmoudi, S., and Kihl, M. "Intelligent multi-agent reinforcement learning model for resources allocation in cloud computing". In: *Journal of King Saud University - Computer and Information Sciences* 34.6, Part A (2022), pp. 2391–2404. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2022.03.016.

[9] Bertsekas, D. *Multiagent Rollout Algorithms and Reinforcement Learning*. 2020. arXiv: 1910.00120 [cs.LG].

[10] Bishop, C. *Pattern recognition and machine learning*. Vol. 4. Springer New York, 2006.

[11] Bontempi, G., Ben Taieb, S., and Le Borgne, Y.-A. "Machine Learning Strategies for Time Series Forecasting". In: *Business Intelligence: Second European Summer School, eBISS 2012, Brussels, Belgium, July 15-21, 2012, Tutorial Lectures*. Ed. by M.-A. Aufaure and E. Zimányi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 62–77. ISBN: 978-3-642-36318-4. DOI: 10.1007/978-3-642-36318-4_3.

[12] Box, G. E. P. and Pierce, D. A. "Distribution of Residual Autocorrelations in Autoregressive-Integrated Moving Average Time Series Models". In: *Journal of the American Statistical Association* 65.332 (1970), pp. 1509–1526. ISSN: 01621459, 1537274X. (Visited on 05/05/2025).

[13] Cao, K. et al. "An Overview on Edge Computing Research". In: *IEEE Access* 8 (2020), pp. 85714–85728. DOI: 10.1109/ACCESS.2020.2991734.

[14] Cao, K. et al. "A Survey on Edge and Edge-Cloud Computing Assisted Cyber-Physical Systems". In: *IEEE Transactions on Industrial Informatics* 17.11 (2021), pp. 7806–7819. DOI: 10.1109/TII.2021.3073066.

[15] Chaisiri, S., Lee, B. S., and Niyato, D. "Optimization of resource provisioning cost in cloud computing". In: *IEEE Transactions on Services Computing*. Vol. 5. 2. 2012, pp. 164–177.

[16] Chen, Z. et al. "Towards Accurate Prediction for High-Dimensional and Highly-Variable Cloud Workloads with Deep Learning". In: *IEEE Transactions on Parallel and Distributed Systems* PP (Nov. 2019), pp. 1–1. DOI: 10.1109/TPDS.2019.2953745.

[17] Chiang, M. and Zhang, T. "Fog computing: Principles, architectures, and applications". In: *Proceedings of the IEEE* 104.5 (2016), pp. 836–846.

[18] Clifton, J. and Laber, E. "Q-Learning: Theory and Applications". In: *Annual Review of Statistics and Its Application* 7.Volume 7, 2020 (2020), pp. 279–301. ISSN: 2326-831X. DOI: https://doi.org/10.1146/annurev-statistics-031219-041220.

[19] Dawoud, W., Takouna, I., and Meinel, C. "Performance evaluation of web servers under heavy load". In: *Journal of Communications* 7.2 (2012), pp. 120–129.

[20] Derdus, K. M., Omwenga, V., and Ogao, P. "Statistical Techniques for Characterizing Cloud Workloads: A Survey". In: *International Journal of Computer and Information Technology2279–0764* (2019).

[21] Developers, S.-l. *Model Evaluation Metrics*. Accessed: 2025-06-18. 2024.

[22] Fard, M. V. et al. "Resource allocation mechanisms in cloud computing: a systematic literature review". In: *IET Software* 14 (6 2020), pp. 638–653. DOI: 10.1049/iet-sen.2019.0338. eprint:

[23] Fehling, C. et al. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer Publishing Company, Incorporated, 2014. ISBN: 3709115671.

[24] Ficco, M. and Palmieri, F. "Game theoretic resource allocation for cloud-based federated content delivery networks". In: *Cluster Computing* 19 (2016), pp. 977–996.

[25] Fujimoto, S., Hoof, H. van, and Meger, D. "Addressing function approximation error in actor-critic methods". In: *arXiv preprint arXiv:1802.09477* (2018).

[26] Gao, J., Wang, H., and Shen, H. "Machine Learning Based Workload Prediction in Cloud Computing". In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 2020, pp. 1–9. DOI: 10.1109/ICCCN49398.2020.9209730.

[27] Ghasemi, M. and Ebrahimi, D. *Introduction to Reinforcement Learning*. 2024. arXiv: 2408.07712 [cs.AI].

[28] Gong, C. et al. "The Characteristics of Cloud Computing". In: *2010 39th International Conference on Parallel Processing Workshops*. 2010, pp. 275–279. DOI: 10.1109/ICPPW.2010.45.

[29] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016.

[30] Hasselt, H. van, Guez, A., and Silver, D. "Deep reinforcement learning with double Q-learning". In: *arXiv preprint arXiv:1509.06461* (2016).

[31] Hochreiter, S. and Schmidhuber, J. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[32] Hongyu, Y. and Anming, W. "Migrating from Monolithic Applications to Cloud Native Applications". In: *2023 8th International Conference on Computer and Communication Systems (ICCCS)*. 2023, pp. 775–779. DOI: 10.1109/ICCCS57501.2023.10150977.

[33] "Intelligent Resource Allocation Optimization for Cloud Computing via Machine Learning". In: *Advances in Computer, Signals and Systems* 9.1 (2025). ISSN: 2371-882X. DOI: 10.23977/acss.2025.090109.

[34] Ioffe, S. and Szegedy, C. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: (2015), pp. 448–456.

[35] Islam, S. et al. "Empirical prediction models for adaptive resource provisioning in the cloud". In: *Future Generation Computer Systems* 28.1 (2012), pp. 155–162. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2011.05.027.

[36] Jayasinghe, U., Mohotti, D., and Hassan, M. M. "A review of dynamic resource management in cloud: Key techniques and future directions". In: *Journal of Cloud Computing* 9.1 (2020), pp. 1–22.

[37] Konan Jean-Claude, K. "A Comprehensive Overview of Artificial Intelligence". In: Dec. 2022, pp. 173–194. DOI: 10.5121/csit.2022.122314.

[38] Krizhevsky, A., Sutskever, I., and Hinton, G. E. "Imagenet classification with deep convolutional neural networks". In: (2012), pp. 1097–1105.

[39] Kumar, J., Goomer, R., and Singh, A. K. "Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model For Cloud Datacenters". In: *Procedia Computer Science* 125 (2018). The 6th International Conference on Smart Computing and Communications, pp. 676–682. ISSN: 1877-0509. DOI: 10.1016/j.procs.2017.12.087.

[40] Lackinger, A., Morichetta, A., and Dustdar, S. "Time Series Predictions for Cloud Workloads: A Comprehensive Evaluation". In: *2024 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. Data cleaning and preparation steps were conducted as described in this paper. IEEE, 2024, pp. 36–45. DOI: 10.1109/SOSE62363.2024.00011.

[41] Lee, J. et al. "A survey of deep learning techniques for resource management in multi-access edge computing". In: *ACM Computing Surveys* 53.6 (2020), pp. 1–36.

[42] Li, Y. *Deep Reinforcement Learning: An Overview*. 2018. arXiv: 1701.07274 [cs.LG].

[43] Liang, E. et al. *RLlib: Abstractions for Distributed Reinforcement Learning*. 2018. arXiv: 1712.09381 [cs.AI].

[44] Lillicrap, T. P. et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2016).

[45] Lin, T. et al. "A Survey of Transformers". In: *arXiv preprint arXiv:2106.04554* (2021).

[46] Liu, N. et al. "A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 372–382. DOI: 10.1109/ICDCS.2017.123.

[47] Madakam, S., Ramaswamy, R., and Tripathi, S. "Internet of Things (IoT): A Literature Review". In: *Journal of Computer and Communications* 3 (Apr. 2015), pp. 164–173. DOI: 10.4236/jcc.2015.35021.

[48] Mell, P. and Grance, T. *The NIST Definition of Cloud Computing*. Tech. rep. Special Publication 800-145. National Institute of Standards and Technology, 2011. DOI: 10.6028/NIST.SP.800-145.

[49] Mnih, V. et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[50] Natarajan, B. "Cloud Load Estimation with Deep Logarithmic Network for Workload and Time Series Optimization". In: *Journal of Soft Computing Paradigm* 3 (Sept. 2021), pp. 234–248. DOI: 10.36548/jscp.2021.3.008.

[51] Nielsen, A. *Practical Time Series Analysis: Prediction with Statistics and Machine Learning*. Titolo collana. O'Reilly, 2019. ISBN: 9781492041658.

[52] Nikravesh, A. Y., Ajila, S. A., and Lung, C.-H. "Measuring Prediction Sensitivity of a Cloud Auto-scaling System". In: *2014 IEEE 38th International Computer Software and Applications Conference Workshops*. 2014, pp. 690–695. DOI: 10.1109/COMPSACW.2014.116.

[53] Nikravesh, A. Y., Ajila, S. A., and Lung, C.-H. "Towards an Autonomic Auto-scaling Prediction System for Cloud Resource Provisioning". In: *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 2015, pp. 35–45. DOI: 10.1109/SEAMS.2015.22.

[54] Patel, Y. S. and Bedi, J. "MAG-D: A multivariate attention network based approach for cloud workload forecasting". In: *Future Generation Computer Systems* 142 (2023), pp. 376–392. ISSN: 0167-739X. DOI: 10.1016/j.future.2023.01.002.

[55] Sarker, I. H. "Machine Learning: Algorithms, Real-World Applications and Research Directions". In: *SN Computer Science* 2.3 (2021), p. 160. ISSN: 2661-8907. DOI: 10.1007/s42979-021-00592-x.

[56] Satyanarayanan, M. "The emergence of edge computing". In: *Computer* 50.1 (2017), pp. 30–39.

[57] Schmidhuber, J. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 0893-6080. URL:

[58] Senjab, K. et al. "A survey of Kubernetes scheduling algorithms". In: *Journal of Cloud Computing* 12.1 (2023), p. 87. ISSN: 2192-113X. DOI: 10.1186/s13677-023-00471-1.

[59] Shah, S. D. A., Gregory, M. A., and Li, S. "Cloud-Native Network Slicing Using Software Defined Networking Based Multi-Access Edge Computing: A Survey". In: *IEEE Access* 9 (2021), pp. 10903–10924. DOI: 10.1109/ACCESS.2021.3050155.

[60] Shi, W. et al. "Edge computing: Vision and challenges". In: *IEEE Internet of Things Journal* (2016).

[61] Srivastava, N. et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[62] Sutskever, I., Vinyals, O., and Le, Q. V. "Sequence to sequence learning with neural networks". In: (2014), pp. 3104–3112.

[63] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. 2nd. MIT Press, 2018.

[64] Taleb, T. et al. "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration". In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1657–1681.

[65] Taye, M. M. "Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions". In: *Computers* 12.5 (2023), p. 91. DOI: 10.3390/computers12050091.

[66] Tighe, M., Keller, S., and Gmach, D. "Dynamic application placement in the data center with an application-oriented approach". In: *IEEE/IFIP Network Operations and Management Symposium*. 2012, pp. 484–491.

[67] Tong, L. et al. "A hierarchical edge cloud architecture for mobile computing". In: *IEEE Network* 30.4 (2016), pp. 22–29.

[68] Towers, M. et al. *Gymnasium: A Standard Interface for Reinforcement Learning Environments*. 2024. arXiv: 2407.17032 [cs.LG].

[69] Verma, A., Ahuja, P., and Neogi, A. "Server workload analysis for power minimization using consolidation". In: *Proceedings of the USENIX Annual Technical Conference*. 2009, pp. 28–28.

[70] Vikas Mongia, D. K. "Resource Allocation in Cloud Computing: A Review". In: *International Journal of Computer Sciences and Engineering* 06 (05 June 2018), pp. 79–84. ISSN: 2347-2693.

[71] Wu, E. and Maslov, D. "Cluster for a Web Application Hosting". In: *Raspberry Pi Retail Applications: Transform Your Business with a Low-Cost Single-Board Computer*. Berkeley, CA: Apress, 2022, pp. 175–210. ISBN: 978-1-4842-7951-9. DOI: 10.1007/978-1-4842-7951-9_8.

[72] Xu, J., Xu, Z., and Shi, B. "Deep Reinforcement Learning Based Resource Allocation Strategy in Cloud-Edge Computing System". In: *Frontiers in Bioengineering and Biotechnology* 10 (2022). ISSN: 2296-4185. DOI: 10.3389/fbioe.2022.908056. URL:

[73] Yu, W. et al. "A Survey on the Edge Computing for the Internet of Things". In: *IEEE Access* 6 (2018), pp. 6900–6919. DOI: 10.1109/ACCESS.2017.2778504.

[74] Yu, Y. et al. "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures". In: *Neural Computation* 31 (July 2019), pp. 1235–1270. DOI: 10.1162/neco_a_01199.

[75] Zhang, Q., Cheng, L., and Boutaba, R. "Cloud computing: Principles and paradigms". In: (2011).

[76] Zhang, Q., Cheng, L., and Boutaba, R. "Cloud computing resource scheduling and a survey of its evolutionary approaches". In: *ACM Computing Surveys (CSUR)* 49.4 (2016), pp. 1–33.

[77] Zhao, Z. et al. "Edge Computing: Platforms, Applications and Challenges". In: *Jisuanji Yanjiu yu Fazhan/Computer Research and Development* 55 (Feb. 2018), pp. 327–337. DOI: 10.7544/issn1000-1239.2018.20170228.