Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

τομεας τεχνολογιας πληροφορικης και υπολογιστων

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

DNN Offloading and Resource Management over Edge Computing Systems

Μελέτη και υλοποίηση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ορφέα Φιλιππόπουλου

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
Αθήνα, Οκτώβριος 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

DNN Offloading and Resource Management over Edge Computing Systems

Μελέτη και υλοποίηση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ορφέα Φιλιππόπουλου

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 31η Οκτωβρίου, 2023.

.....	.....	.....
Δημήτριος Σούντρης	Παναγιώτης Τσανάκας	Σωτήριος Ξύδης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2023

.....

Onoma
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

στην οικογένεια μου

Περίληψη

Στη σημερινή ψηφιακή εποχή, η ραγδαία αύξηση των δεδομένων και η αυξανόμενη χρήση της τεχνητής νοημοσύνης (AI) έχουν καταστήσει τα βαθιά νευρωνικά δίκτυα βασικό άξονα της σύγχρονης πληροφορικής. Αυτά τα ευέλικτα μοντέλα τεχνητής νοημοσύνης χρησιμεύουν ως βάση για ποικίλες εφαρμογές, από την αναγνώριση εικόνας έως την επεξεργασία φυσικής γλώσσας, μεταμορφώνοντας τις βιομηχανίες και το ψηφιακό μας τοπίο. Καθώς τα νευρωνικά δίκτυα βρίσκονται στο επίκεντρο, η ανάγκη για την αποτελεσματική εκτέλεσή τους γίνεται όλο και πιο σημαντική.

Παραδοσιακά, τα νευρωνικά δίκτυα εκτελούνται σε περιβάλλοντα υπολογιστικού νέφους (Cloud computing), τα οποία είναι γνωστά για τους εκτεταμένους υπολογιστικούς πόρους που βρίσκονται εντός των κέντρων δεδομένων (data centers). Ενώ η προσέγγιση αυτή προσφέρει σημαντική υπολογιστική ισχύ, εισάγει προκλήσεις που σχετίζονται με την καθυστέρηση και τη διαθεσιμότητα του δικτύου. Οι προκλήσεις αυτές μπορεί να είναι ιδιαίτερα περιοριστικές για εφαρμογές που απαιτούν απόκριση σε πραγματικό χρόνο.

Σε αυτή την εργασία, εκτελούμε τα νευρωνικά δίκτυα στο Edge (Edge computing). Το Edge computing αντιπροσωπεύει μια εναλλακτική προσέγγιση για την ανάπτυξη νευρωνικών δικτύων, επιδιώκοντας να αντιμετωπίσει τους περιορισμούς των παραδοσιακών περιβαλλόντων νέφους. Φέρνει τον υπολογισμό πιο κοντά στις πηγές δεδομένων, επιτρέποντας την επεξεργασία δεδομένων σε πραγματικό χρόνο. Με αυτόν τον τρόπο μειώνουμε την καθυστέρηση εκτέλεσης των νευρωνικών δικτύων και ενισχύουμε την ανταπόκριση των εφαρμογών, καθιστώντας το ιδανικό για σενάρια όπου η έγκαιρη λήψη αποφάσεων είναι κρίσιμη. Ωστόσο, το Edge computing έχει και αυτό ένα σύνολο προκλήσεών. Οι συσκευές που λειτουργούν στο Edge ποικίλλουν σε μεγάλο βαθμό ως προς την υπολογιστική ικανότητα, από συσκευές υψηλής απόδοσης έως συσκευές IoT με περιορισμένους πόρους. Η διαχείριση αυτής της ετερογένειας και η αποτελεσματική κατανομή των πόρων για τη διασφάλιση της βέλτιστης εκτέλεσης των νευρωνικών δικτύων είναι πολύπλοκη. Για το λόγο αυτό, κάνουμε χρήση του Serverless computing. Το Serverless Computing αφαιρεί τις πολυπλοκότητες της διαχείρισης υποδομών, απλοποιώντας την κλιμάκωση και βελτιστοποιώντας τη χρήση των πόρων, μειώνοντας τα λειτουργικά έξοδα. Αυτή η προσέγγιση ευθυγραμμίζεται απρόσκοπτα με τα περιβάλλοντα Edge.

Αξιοποιώντας λοιπόν το Serverless Computing στο Edge, σχεδιάσαμε και αναπτύξαμε ένα πλήρες και σταθερό framework για την ανάπτυξη νευρωνικών δικτύων σε ένα σύμπλεγμα Edge συσκευών.

Στην κορυφή του πλαισίου μας βρίσκεται ένας αλγόριθμος Ενισχυτικής Μάθησης (Reinforcement Learning). Η βασική αποστολή του είναι διττή: πρώτον, να διασφαλίζει ότι η καθυστέρηση εκτέλεσης των νευρωνικών δικτύων είναι εντός των καθορισμένων SLAs (Συμφωνία Επιπέδου Υπηρεσιών), ικανοποιώντας τους στόχους για το χρόνο απόκρισης και δεύτερον, να βελτιστοποιεί την κατανάλωση ενέργειας κατανέμοντας εργασίες σε συσκευές με χαμηλή ενεργειακή κατανάλωση, όποτε αυτό είναι εφικτό. Αυτός ο αλγόριθμος ενισχυτικής μάθησης παίζει καθοριστικό ρόλο στην ενίσχυση της συνολικής αποδοτικότητας και απόκρισης του συστήματός μας.

Λέξεις Κλειδιά — Νευρωνικά Δίκτυα, Μηχανική Μάθηση, Βαθειά Μηχανική Μάθηση, Ενισχυτική Μάθηση, Υπολογιστικό νέφος (Cloud Computing), Edge Computing, Serverless Computing

Abstract

In today's digital era, the explosive growth of data and the increasing prominence of artificial intelligence (AI) have made neural networks a linchpin of modern computing. These versatile AI models serve as the foundation for diverse applications, from image recognition to natural language processing, transforming industries and our digital landscape. As neural networks take center stage, the demand for their efficient execution becomes increasingly critical.

Traditionally, neural networks are deployed in Cloud (Cloud computing) environments, which are known for their extensive computational resources located within data centers. While this approach offers significant computational power, it introduces challenges related to latency and network availability. These challenges can be particularly limiting for applications that demand real-time responsiveness.

In this work, We deploy neural networks in the Edge (Edge computing). Edge computing represents an alternative approach to neural network deployment, seeking to address the limitations of traditional cloud environments. It brings computation closer to data sources, enabling real-time data processing. In this way We reduce exeuction latency of neural networks and enhance the responsiveness of applications, making it ideal for scenarios where timely decision-making is critical. However, the edge environment presents its set of challenges. The devices operating at the edge vary widely in computational capacity, from high-performance servers to resource-constrained IoT devices. Managing this heterogeneity and efficiently allocating resources to ensure optimal neural network execution is complex. For this reason, We make use of Serverless computing. Serverless computing abstracts the complexities of infrastructure management, simplifying resource scaling, reducing operational overhead and optimizing resource utilization. This approach aligns seamlessly with edge environments.

By leveraging Serverless computing in the Edge, We designed and developed a complete and robust framework for deploying neural networks in an edge cluster.

On top of our framework is a Reinforcement Learning (RL) algorithm. Its core mission is twofold: first, to ensure that neural networks execution latency is within the defined SLAs, meeting response time targets; second, to optimize energy consumption by allocating tasks to energy-efficient devices whenever this is feasible. This RL algorithm plays a pivotal role in enhancing the overall efficiency and responsiveness of our system.

Keywords — Neural Networks, Machine Learning, Deep Learning, Reinforcement Learning, Cloud Computing, Edge Computing, Serverless Computing

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κ.Δημήτριο Σούντρη και τους υποψήφιους διδάκτορες Δημοσθένη Μασούρο και Μανώλη Κατσαραγάκη για την επίβλεψη αυτής της διπλωματικής εργασίας και για την ευκαιρία που μου έδωσαν να την εκπονήσω στο εργαστήριο Μικροεπεξεργαστών και Ψηφιακών Συστημάτων. Επίσης, θα ήθελα να ευχαριστήσω την οικογένεια μου για την καθοδήγηση και την ηθική συμπαράσταση που μου προσέφεραν όλα αυτά τα χρόνια. Τέλος θα ήθελα να ευχαριστήσω τους φίλους μου που ήταν δίπλα μου κατά τη διάρκεια της φοίτησής μου.

Φιλιππόπουλος Ορφέας
Οκτώμβριος 2023

Contents

Figure List

Table List

Εκτεταμένη Ελληνική Περίληψη

0.1 Εισαγωγή

Στον χώρο της σύγχρονης πληροφορικής, η αποδοτική εκτέλεση νευρωνικών δικτύων σε μια ποικίλη και σύνθετη γκάμα περιβαλλόντων αποτελεί μια σοβαρή πρόκληση. Αυτή η πρόκληση αποτελείται από την οργάνωση της εκτέλεσης των επιπέδων του νευρωνικού δικτύου ενώ ταυτόχρονα την πλήρη αξιοποίηση των διαθέσιμων υλικών πόρων. Καθώς τα νευρωνικά δίκτυα εξελίσσονται σε πολυπλοκότητα και εξειδίκευση, η αντιμετώπιση αυτής της πρόκλησης γίνεται όλο και πιο κρίσιμη. Υπάρχει η ανάγκη για μια καινοτόμα λύση ικανή να βελτιστοποιήσει τη διαδικασία εκτέλεσης διανέμοντας με έξυπνο τρόπο τα επίπεδα του νευρωνικού δικτύου σε ένα σύμπλεγμα υπολογιστικών μηχανών. Αυτό περιλαμβάνει την εκτέλεση των επιπέδων των νευρωνικών δικτύων στο πλέον κατάλληλο υλικό περιβάλλον, είτε αυτό είναι ο επεξεργαστής (CPU) είτε η κάρτα γραφικών (GPU), λαμβάνοντας υπόψη τα μοναδικά χαρακτηριστικά του. Επιπλέον, αυτή η λύση πρέπει να παρέχει την ευελιξία να εξυπηρετεί μια ευρεία γκάμα αρχιτεκτονικών και ρυθμίσεων νευρωνικών δικτύων. Ο στόχος είναι η εξομάλυνση της διαδικασίας εκτέλεσης, η μείωση της καθυστέρησης, τηρώντας τα SLAs (Service Level Agreements), βελτίωση της αξιοποίησης των πόρων και, τελικά, η αξιοποίηση των δυνατοτήτων των νευρωνικών δικτύων σε πρακτικές εφαρμογές.

Επιπλέον, ένα αναπόσπαστο κομμάτι αυτής της πρόκλησης βρίσκεται στο δυναμική τοποθέτηση των επιπέδων των νευρωνικών δικτύων στις υπολογιστικές συσκευές, μια πολύπλοκη εργασία που απαιτεί προσεκτική εξέταση των απαιτήσεων κάθε επιπέδου και των διαθέσιμων υπολογιστικών πόρων στο σύμπλεγμα των υπολογιστικών μηχανών. Το πρόβλημα της δυναμικής τοποθέτησης των επιπέδων των νευρωνικών δικτύων προσθέτει μια επιπλέον στρώση πολυπλοκότητας, απαιτώντας μια ευφυή προσέγγιση για να καθοριστεί πού και πώς θα εκτελεστεί κάθε επίπεδο με τον πιο βέλτιστο τρόπο.

Αυτή η διπλωματική εργασία εξερευνά και παρουσιάζει ακριβώς μια τέτοια λύση - ένα βελτιστοποιημένο framework που αξιοποιεί τεχνολογίες όπως το Kubernetes, το Knative, το Deep Learning, και πιο συγκεκριμένα το Reinforcement Learning, για να αντιμετωπίσει την περίπλοκη αλληλεπίδραση μεταξύ της αποτελεσματικής εκτέλεσης και του δυναμικού προγραμματισμού σε αποσπώμενες αναπτύξεις νευρωνικών δικτύων.

0.2 Σχετική Βιβλιογραφία

Οι Hyuk-Jin κ.ά. [1] προτείνουν μια partitioning-based τεχνική εκφόρτωσης DNN για υπολογιστές ακμής (edge devices). Η προτεινόμενη τεχνική, IONN, διαμερίζει τα επίπεδα DNN σε partitions και τα στέλνει σταδιακά στο cloud, για να επιτρέψει τη συνεργατική εκτέλεση από τον πελάτη και το cloud, ακόμη και πριν μεταφορτωθεί ολόκληρο το μοντέλο DNN. Τα πειραματικά αποτελέσματα καταδεικνύουν την αποτελεσματικότητα του IONN όσον αφορά τη βελτίωση τόσο της απόδοσης των αιτημάτων όσο και της κατανάλωσης ενέργειας κατά τη μεταφόρτωση του μοντέλου DNN, σε σύγκριση με μια απλή προσέγγιση all-at-once.

Οι Laskaridis et al. [2] προτείνουν ένα νέο κατανεμημένο σύστημα εκτέλεσης βαθιών νευρωνικών δικτύων που στοχεύει στην αντιμετώπιση των περιορισμών των υφιστάμενων προσεγγίσεων με τη χρήση μιας μεθόδου προοδευτικής εκτέλεσης βαθιών νευρωνικών δικτύων. Το προτεινόμενο σύστημα, SPINN εισάγει ένα

σχήμα διανομής προοδευτικών μοντέλων με πρώιμες εξόδους σε συσκευή και server, στο οποίο μια έξοδος είναι πάντα παρούσα στη συσκευή, εγγυώμενη τη διαθεσιμότητα ενός αποτελέσματος ανά πάσα στιγμή. Επιπλέον, το SPINN θέτει το κάτω όριο πρόβλεψης ως ρυθμιζόμενη παράμετρο για την προσαρμογή του συμβιβασμού μεταξύ ακρίβειας-ταχύτητας. Παράλληλα, το σύστημα προτείνει έναν νέο χρονοπρογραμματιστή εκτέλεσης (run-time scheduler) που συντονίζει από κοινού το σημείο διαχωρισμού και την πολιτική πρόωρης εξόδου του προοδευτικού μοντέλου, αποδίδοντας μια συμπεριφορά προσαρμοσμένη στις απαιτήσεις απόδοσης της εφαρμογής υπό δυναμικές συνθήκες. Μια ολοκληρωμένη αξιολόγηση των επιδόσεων του συστήματος δείχνει ότι το SPINN υπερτερεί έναντι των σύγχρονων ομολόγων του συνεργατικής εκτέλεσης βαθιών νευρωνικών δικτύων έως και 2 φορές στην επιτυγχανόμενη απόδοση υπό ποικίλες συνθήκες δικτύου, μειώνει το κόστος του server έως και 6.8 φορές και βελτιώνει την ακρίβεια κατά 20.7 % υπό περιορισμούς καθυστέρησης, παρέχοντας παράλληλα εύρωστη λειτουργία υπό αβέβαιες συνθήκες συνδεσιμότητας και σημαντική εξοικονόμηση ενέργειας σε σύγκριση με την εκτέλεση στο cloud.

Οι Xueyu Hou et al. [3] προτείνουν το Dystri, ένα καινοτόμο πλαίσιο που σχεδιάστηκε για να διευκολύνει τη δυναμική εκτέλεση DNN σε κατανεμημένες υποδομές ακραίων σημείων, εξυπηρετώντας έτσι πολλαπλούς ετερογενείς χρήστες. Η αρχιτεκτονική περιλαμβάνει κατανεμημένους ελεγκτές και έναν γενικό συντονιστή, επιτρέποντας προσαρμογές της ποιότητας υπηρεσιών ανά αίτηση και ανά χρήστη, εξασφαλίζοντας άμεσο, ευέλικτο και διακριτό έλεγχο. Το πλαίσιο αξιολογείται σε τρεις ετερογενείς πλατφόρμες υπηρεσιών εκτέλεσης DNN πολλαπλών χρηστών που αναπτύσσονται σε κατανεμημένη υποδομή edge συσκευών και περιλαμβάνουν επτά εφαρμογές DNN. Τα αποτελέσματα δείχνουν ότι το Dystri επιτυγχάνει σχεδόν μηδενικές απώλειες προθεσμιών και υπερέχει στην προσαρμογή σε ποικίλους αριθμούς χρηστών και εντάσεις αιτημάτων. Επιπλέον, το Dystri υπερτερεί έναντι των βασικών λύσεων με βελτίωση της ακρίβειας έως και 95%.

Οι Yanan Yang et al. [4] προτείνουν το INFless. Το INFless παρέχει μια ενοποιημένη, ετερογενή αφαίρεση πόρων μεταξύ CPU και επιταχυντών, η οποία επιτρέπει την αποτελεσματική κατανομή πόρων με βάση τις απαιτήσεις του φόρτου εργασίας. Το σύστημα επιτυγχάνει υψηλή απόδοση χρησιμοποιώντας ενσωματωμένους μηχανισμούς ομαδοποίησης και μη ομοιόμορφης κλιμάκωσης, ενώ υποστηρίζει επίσης χαμηλή καθυστέρηση μέσω συντονισμένης διαχείρισης του χρόνου αναμονής στην ουρά των παρτίδων (batches), του χρόνου εκτέλεσης και του ρυθμού cold start. Η αρχιτεκτονική του INFless έχει σχεδιαστεί για να αντιμετωπίζει τις προκλήσεις της διαχείρισης υβριδικών συστημάτων CPU/επιταχυντή, της επιλογής κατάλληλων μεγεθών παρτίδων (batches) και πόρων και της ελαχιστοποίησης της επιβάρυνσης εκτέλεσης. Μια λεπτομερής επισκόπηση της αρχιτεκτονικής του INFless περιλαμβάνει την πύλη (gate), τον χρονοπρογραμματιστή (run-time scheduler), το μοντέλο πρόβλεψης, τα προφίλ χειριστών (operator profiles), τον κατανεμητή, το φόρτο εργασίας δέσμης (batching workload), τους κόμβους CPU/GPU, τον διαχειριστή του cold start, τη μηχανή αυτόματης κλιμάκωσης και άλλα.

Οι Lockhar et al. [5] προτείνουν το Scission, ένα εργαλείο για την αυτοματοποιημένη σύγκριση και αξιολόγηση DNN σε ένα δεδομένο σύνολο συσκευής-στόχου, ακμής (edge) και υπολογιστικού νέφους (cloud) πόρων για τον προσδιορισμό της βέλτιστης κατάτμησης για τη μεγιστοποίηση της απόδοσης των DNN. Υποστηρίζεται από μια προσέγγιση συγκριτικής αξιολόγησης που καθορίζει τον συνδυασμό των πιθανών πόρων υλικού-στόχου και την ακολουθία των επιπέδων που πρέπει να κατανεμηθούν για τη μεγιστοποίηση της κατανεμημένης απόδοσης των DNN, λαμβάνοντας υπόψη τους στόχους που καθορίζονται από τον χρήστη. Το Scission βασίζεται σε εμπειρικά δεδομένα και δεν εκτιμά την απόδοση κάνοντας υποθέσεις για το υλικό-στόχο ή τα στρώματα DNN. Πραγματοποιήθηκαν πειραματικές μελέτες σε 18 διαφορετικά DNN για να αποδειχθεί ότι το Scission είναι ένα χρήσιμο εργαλείο για την εκτέλεση κατανεμημένων DNN με επίγνωση του πλαισίου και αποδοτική απόδοση. Το Scission μπορεί επίσης να λάβει αποφάσεις που δεν μπορούν να ληφθούν χειροκίνητα από έναν άνθρωπο λόγω της πολυπλοκότητας και του αριθμού των διαστάσεων που επηρεάζουν τον χώρο αναζήτησης.

Οι Kakolyris et al. [6] προτείνουν το RoaD-RuNNer, ένα συνεργατικό πλαίσιο για την κατάτμηση και την εκφόρτωση βαθιών νευρωνικών δικτύων (DNN) σε ετερογενή συστήματα ακμών. Αυτή η καινοτόμος προσέγγιση αντιμετωπίζει τις προκλήσεις διαχείρισης των πόρων των DNN που αναπτύσσονται σε συστήματα υπολογιστών άκρων και προσφέρει μια πολλά υποσχόμενη λύση για αποτελεσματική κατάτμηση και εκφόρτωση DNN. Οι συγγραφείς προτείνουν έναν μηχανισμό δυναμικής εκφόρτωσης που χρησιμοποιεί συνεργατικό φιλτράρισμα για την πρόβλεψη του χρόνου εκτέλεσης και της κατανάλωσης ενέργειας των

επιμέρους επιπέδων σε διαφορετικές αρχιτεκτονικές CPU/GPU. Το πλαίσιο περιλαμβάνει επίσης έναν μηχανισμό δυναμικής κατάτμησης που διαχωρίζει και εκφορτώνει αποτελεσματικά τα στρώματα DNN. Οι συγγραφείς διεξήγαγαν εκτεταμένη πειραματική αξιολόγηση του προτεινόμενου πλαισίου τους, συγκρίνοντας το με βασικούς αλγορίθμους και σύγχρονες προσεγγίσεις εκφόρτωσης DNN σε πραγματικό υλικό και δίκτυο. Τα αποτελέσματα δείχνουν ότι το RoaD-RuNNer υπερτερεί έναντι των υφιστάμενων προσεγγίσεων επιτυγχάνοντας έως και 9,58× επιτάχυνση κατά μέσο όρο και έως και 88,73% λιγότερη κατανάλωση ενέργειας κατά μέσο όρο.

Οι Patterson κ.ά. [7] προτείνουν μια αρχιτεκτονική συστήματος υλικού-λογισμικού, το HiveMind, που επιτρέπει την προγραμματιζόμενη εκτέλεση σύνθετων ροών εργασιών μεταξύ υπολογιστικού νέφους και πόρων ακραίων σημείων με αποδοτικό και κλιμακούμενο τρόπο. Αυτή η αρχιτεκτονική αντιμετωπίζει τις προκλήσεις του χειροκίνητου καταμερισμού των εργασιών και της αλλαγής του τόπου εκτέλεσης των υπολογισμών, οι οποίες μπορούν να επηρεάσουν την απαιτούμενη υποδομή λογισμικού. Το εργαλείο σύνθεσης προγράμματος διερευνά αυτόματα στρατηγικές τοποθέτησης εργασιών, απλοποιώντας την προγραμματισιμότητα των εφαρμογών cloud-edge. Το επαναδιαμορφώμενο υλικό επιτάχυνσης για πρόσβαση σε δίκτυο και απομακρυσμένη μνήμη συμβάλλει στην απόδοση και την επεκτασιμότητα της πλατφόρμας. Συνολικά, αυτή η αρχιτεκτονική παρουσιάζει μια ολοκληρωμένη λύση στις προκλήσεις των αυτόνομων συσκευών, καθιστώντας την χρήσιμη συνεισφορά στον τομέα.

Οι Bin Wang κ.ά. [8] προτείνουν την πλατφόρμα LaSS, μια νέα αρχιτεκτονική για την αντιμετώπιση των προκλήσεων του serverless computing στο edge. Η πλατφόρμα χρησιμοποιεί προσεγγίσεις βασισμένες σε μοντέλα για την ακριβή πρόβλεψη των πόρων που απαιτούνται για τις λειτουργίες serverless παρουσία δυναμικών φόρτων εργασίας. Η LaSS χρησιμοποιεί μεθόδους βασισμένες σε ουρές αναμονής για τον προσδιορισμό της κατάλληλης κατανομής για κάθε φιλοξενούμενη λειτουργία και την αυτόματη κλιμάκωση των κατανεμημένων πόρων σε απόκριση στη δυναμική του φόρτου εργασίας. Η πλατφόρμα χρησιμοποιεί επίσης μεθόδους ανάκτησης πόρων που βασίζονται στην αποσυμπίεση και τον τερματισμό containers για την εκ νέου ανάθεση πόρων από λειτουργίες με υπερβολική παροχή σε λειτουργίες με χαμηλή παροχή. Οι συγγραφείς υλοποιούν ένα πρωτότυπο της προσέγγισής τους στο OpenWhisk [9] και διεξάγουν μια λεπτομερή πειραματική αξιολόγηση, αποδεικνύοντας την ικανότητα του LaSS να ανταποκρίνεται στους στόχους του επιπέδου εξυπηρέτησης και να λειτουργεί με εγγυήσεις δίκαιης κατανομής σε σενάρια υπερφόρτωσης.

0.3 Επισκόπηση βασικών εννοιών

Τεχνητή Νοημοσύνη - Νερωνικά Δίκτυα

Τα Τεχνητά Νευρωνικά Δίκτυα, ένα κύριο κομμάτι της Βαθιάς Μάθησης, είναι υπολογιστικά μοντέλα εμπνευσμένα από τη νευρική δομή του ανθρώπινου εγκεφάλου. Στην καρδιά τους, τα Τεχνητά Νευρωνικά Δίκτυα αποτελούνται από διασυνδεδεμένες μονάδες που ονομάζονται νευρώνες. Αυτοί οι νευρώνες επεξεργάζονται πληροφορίες, παρόμοια με τους βιολογικούς νευρώνες, λαμβάνοντας εισόδους, εκτελώντας υπολογισμούς και παράγοντας μια έξοδο. Μέσα σε ένα νευρωνικό δίκτυο, η πληροφορία ρέει από τους είσοδο νευρώνες προς τους εξοδο νευρώνες μέσω ενός περίπλοκου δικτύου συνδεδεμένων στρωμάτων. Ένα παράδειγμα ενός τέτοιου δικτύου απεικονίζεται στην εικόμα

Ένα σημαντικό στοιχείο των Τεχνητά Νευρωνικά Δίκτυα είναι η συνάρτηση ενεργοποίησης. Εισάγει μη γραμμικότητα στο μοντέλο, επιτρέποντας του να μάθει πολύπλοκες σχέσεις μέσα στα δεδομένα. Συνηθισμένες συναρτήσεις ενεργοποίησης περιλαμβάνουν τη σιγμοειδή συνάρτηση (sigmoid), την υπερβολική εφαπτομενική (tanh) συνάρτηση τη συνάρτηση του γραμμικού ρελέ (ReLU) και τη μοναδιαία βηματική συνάρτηση (step). Κάθε μια έχει τα δικά της χαρακτηριστικά που επηρεάζουν τη δυνατότητα του δικτύου να μοντελοποιήσει και να προσαρμόσει τα δεδομένα. Και οι τέσσερεις απεικονίζονται στο figure 0.3.2.

Τα Τεχνητά Νευρωνικά Δίκτυα εμφανίζονται σε διάφορες μορφές, καθεμία εκ των οποίων σχεδιάστηκε για να ανταποκρίνεται σε διάφορα είδη εργασιών και δεδομένων. Τα Νευρωνικά Δίκτυα Με Προώθηση (FNN) [11] αποτελούν τη βάση των Τεχνητών Νευρωνικών Δίκτυων, όπου η πληροφορία ρέει μόνο από την είσοδο στην έξοδο, καθιστώντας τα ιδανικά για προβλήματα όπου αυτή η γραμμική ροή είναι κατάλληλη. Αυτά τα δίκτυα είναι εξαιρετικά αποτελεσματικά στην αναγνώριση προτύπων και την επίλυση προβλημάτων όπου τα δεδομένα είναι σειριακά.

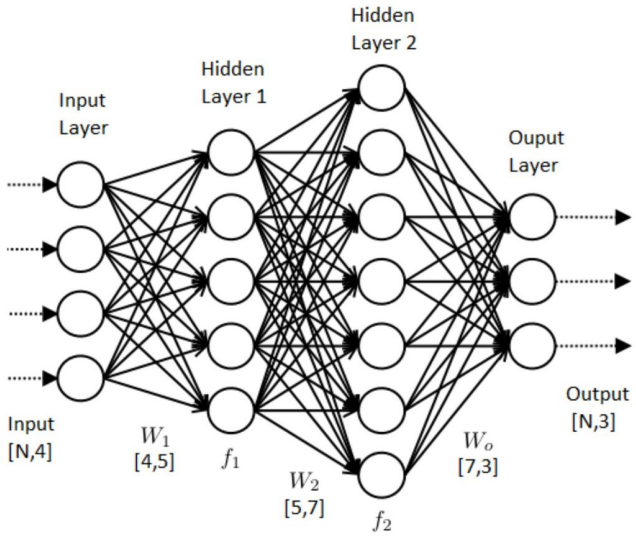
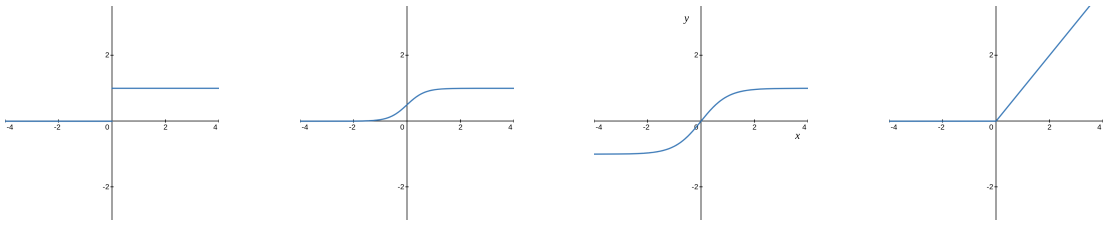


Figure 0.3.1: Παράδειγμα Τεχνητού Νευρωνικού Δίκτυου [10]



(a) Step function (b) Sigmoid function (c) tanh function (d) ReLU function

Figure 0.3.2: Most popular activation functions

Τα Επαναληπτικά Νευρωνικά Δίκτυα (RNNs) [12], από την άλλη πλευρά, είναι κατάλληλα για εργασίες που συνδέονται με ακολουθιακά δεδομένα, όπως η μεταγλώττιση προτάσεων και η πρόβλεψη χρονοσειρών. Επιτρέπουν τη διατήρηση εσωτερικής μνήμης, καθιστώντας τα ικανά να αντιμετωπίζουν μακροπρόθεσμες εξαρτήσεις στα δεδομένα. Ενδεικτικές παραλλαγές των RNNs περιλαμβάνουν τις μονάδες Long Short-Term Memory (LSTM) [13] και Gated Recurrent Unit (GRU) [14], οι οποίες έχουν αναδειχθεί ως ιδιαίτερα αποτελεσματικές στην αντιμετώπιση μακροπρόθεσμων εξαρτήσεων.

Τα Συνελικτικά Νευρωνικά Δίκτυα (CNNs) [15] είναι εξειδικευμένα για εργασίες που σχετίζονται με πλέγματα δεδομένων, όπως εικόνες ή φασματογράμματα ήχου. Χρησιμοποιούν επίπεδα συνέλιξης για να αναγνωρίσουν χαρακτηριστικά στα εισερχόμενα δεδομένα, λαμβάνοντας υπόψη τις χωρικές σχέσεις. Τα CNNs έχουν ανατρέψει τον κλάδο της όρασης υπολογιστών και της αναγνώρισης εικόνας, επιτρέποντας την εξαγωγή σημαντικών πληροφοριών από πολύπλοκες δομές δεδομένων.

Αυτές οι παραλλαγές των Τεχνητά Νευρωνικά Δίκτυα είναι μόνο μια επισκόπηση του ποικίλου τοπίου της Βαθιάς Μάθησης. Έχουν ωθήσει σε εντυπωσιακές προόδους σε πεδία όπως η επεξεργασία φυσικής γλώσσας (NLP), η αναγνώριση φωνής και οι αυτόνομοι ρομποτικοί χειρισμοί, σχηματίζοντας την επανάσταση της Τεχνητής Νοημοσύνης στην εποχή μας.

Ενισχυτική μάθηση (Reinforcement Learning)

Η Ενισχυτική Μάθηση (Reinforcement Learning - RL) είναι ένας προηγμένη κλάδος της Τεχνητής Νοημοσύνης που επικεντρώνεται στη λήψη αποφάσεων και την εκπαίδευση από, ήδη υπάρχουσα, εμπειρία (Experience replay). Επικεντρωνεται στην αυτόνομη λήψη αποφάσεων, μεθοδολογία που βασίζεται στην ανταμοιβή και το πείραμα. Σε ένα περιβάλλον, ένας αλγόριθμος RL αλληλεπιδρά και παρατηρεί διαρκώς με

το αυτό και εκτελεί δράσεις με σκοπό να μεγιστοποιήσει μια συγκεκριμένη ανταμοιβή. Κατά τη διάρκεια αυτών των αλληλεπιδράσεων, ο αλγόριθμος αναπτύσσει μια στρατηγική για τη λήψη αποφάσεων που του επιτρέπει να επιτυγχάνει τους στόχους του με τον καλύτερο τρόπο. Κάθε δράση έχει συνέπειες στην κατάσταση του περιβάλλοντος και στην ανταμοιβή που λαμβάνει ο αλγόριθμος, και αυτός μαθαίνει από τα αποτελέσματα των δράσεών του. Οι επιλογές που οδηγούν σε υψηλότερες ανταμοιβές προτιμώνται, και αυτός είναι ο τρόπος με τον οποίο ο αλγόριθμος μαθαίνει να προτιμά τις βέλτιστες δράσεις. Η εικόνα 0.3.3 δείχνει ακριβώς αυτό που αναλύθηκε παραπάνω, δηλαδή τη δράση-αντίδραση μεταξύ RL αλγορίθμου και περιβάλλοντος.

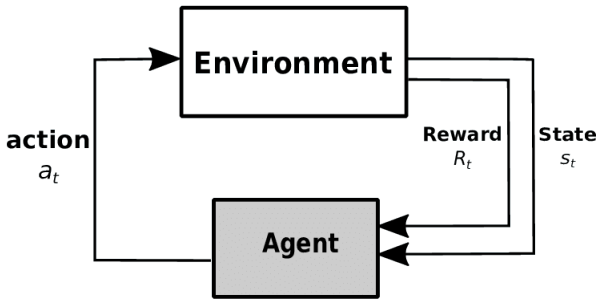


Figure 0.3.3: Παράδειγμα δράσης-αντίδρασης ενός αλγορίθμου Ενισχυτικής Μάθησης με το περιβάλλον του [16]

Για την ανάλυση και τον υπολογισμό των βέλτιστων στρατηγικών, τα RL χρησιμοποιούν τις εξισώσεις Bellman . Οι εξισώσεις αυτές εκφράζουν την αναμενόμενη ανταμοιβή για κάθε κατάσταση και δράση, λαμβάνοντας υπόψη το μέλλον και την πιθανότητα μετάβασης σε νέες καταστάσεις. Οι αλγόριθμοι RL στοχεύουν στον υπολογισμό των βέλτιστων στρατηγικών που θα μεγιστοποιήσουν την αναμενόμενη ανταμοιβή. Μαθηματικά οι εξισώσεις αυτές μπορούν να εκφραστούν ως:

$$V_\pi(s) = \sum_{s'}[P_{\pi(s)}(s'|s,\pi(s)) * (R(s,\pi(s),s') + \gamma * V\pi(s'))] \tag{0.3.1}$$

Όπου το s είναι η κατάσταση του περιβάλλοντος αυτή τη στιγμή, s' είναι η επόμενη κατάσταση του περιβάλλοντος, π είναι η πολιτική που ακολουθείται, $P_{\pi(s)}(s'|s,\pi(s))$ είναι η πιθανότητα, δεδομένου ότι βρισκόμαστε στη κατάστασ s και ακολουθείτε η πολιτική π, να μεταβούμε στη κατάσταση περιβάλλοντος s'. Τέλος το $R(s,\pi(s),s')$ είναι η ανταμοιβή εάν από τη κατάσταση s του περιβάλλοντος, ακολουθώντας τη πολική π, μεταβούμε στη κατάσταση s', γ ονομάζεται ρυθμός έκπτωσης και προσδιορίζει τη σημασία (ποσοτικά) των μελλοντικών ανταμοιβών στη παρούσα κατάσταση και τέλος το $V_\pi s$ που προσδιορίζει την αναμενόμενη ανταμοιβή στη κατάσταση s ακολουθώντας τη πολιτική π.

Υπάρχουν διάφοροι αλγόριθμοι ενισχυτικής μάθησης που χρησιμοποιούνται για την επίλυση προβλημάτων. Μερικοί από αυτούς περιλαμβάνουν τον αλγόριθμο Q-Learning [17], που είναι κατάλληλος για προβλήματα ελέγχου, και τον αλγόριθμο PPO (Proximal Policy Optimization) [18], που είναι αποτελεσματικός για στοχαστικά προβλήματα ελέγχου. Οι αλγόριθμοι αυτοί βασίζονται στις αρχές της ενισχυτικής μάθησης για την εκπαίδευση των μοντέλων και τη λήψη αποφάσεων. Πέρα από αυτούς, υπάρχουν πολλοί άλλοι αλγόριθμοι που προσφέρουν μοναδικές δυνατότητες για την επίλυση διαφόρων προβλημάτων σε ποικίλους τομείς, καθώς και πολλές παραλλαγές των υπαρχόντων αλγορίθμων που προσαρμόζονται στις διάφορες απαιτήσεις των προβλημάτων.

Οι εν λόγω αλγόριθμοι RL έχουν εφαρμογές σε πολλούς τομείς, όπως η αυτόνομη πλοήγηση ρομπότ, η διαχείριση αποθεμάτων, η αυτόνομη οδήγηση οχημάτων, και ακόμη και στην ανάπτυξη παιχνιδιών, καθώς επιτρέπουν σε μηχανές να μάθουν και να προσαρμόζονται σε αντίστοιχα περιβάλλοντα.

Serverless Computing

Το Serverless Computing αντιπροσωπεύει μια εξελισσόμενη προσέγγιση στην ανάπτυξη και εκτέλεση λογισμικών εφαρμογών. Σε αυτό το μοντέλο, οι προγραμματιστές δεν χρειάζεται πλέον να διαχειρίζονται διακομιστές, να φροντίζουν για την κλιμάκωση του υλικού ή να ανησυχούν για τη διαθεσιμότητα των πόρων. Αντίθετα, μπορούν να επικεντρωθούν αποκλειστικά στην ανάπτυξη του κώδικά τους και τη δημιουργία λειτουργικών εφαρμογών.

Ένα από τα κύρια χαρακτηριστικά του Serverless είναι το autoscaling. Αυτό σημαίνει ότι η υπηρεσία ανταποκρίνεται αυτόματα σε αυξήσεις ή μειώσεις του φορτίου εργασίας, αυξάνοντας ή μειώνοντας δυναμικά τους πόρους που διατίθενται για την εκτέλεση των λειτουργιών. Αυτό εξαλείφει την ανάγκη για προετοιμασία των διακομιστών για αναμενόμενες αλλαγές στο φόρτο εργασίας και εξασφαλίζει ότι η εφαρμογή σας παραμένει αποδοτική ακόμα και κατά τις περιόδους αυξημένου φόρτου.

Εκτός από την autoscaling και την αποτελεσματικότητα στη χρήση πόρων, το serverless computing άλλο ένα πολύ σημαντικό χαρακτηριστικό είναι το μοντέλο χρέωσης *pay-as-you-go* το οποίο αποτελεί μια σημαντική αλλαγή από τις παραδοσιακές ρυθμίσεις υποδομής. Με το serverless, οι οργανισμοί χρεώνονται μόνο για τους πραγματικούς υπολογιστικούς πόρους που καταναλώνονται κατά τη διάρκεια της εκτέλεσης των λειτουργιών ή των εφαρμογών τους. Αυτή η μέθοδος χρέωσης είναι ιδιαίτερα πλεονεκτική, καθώς εξαλείφει την ανάγκη για αρχικές επενδύσεις σε υποδομές και παρέχει προβλεψιμότητα του κόστους, καθιστώντας το ελκυστική επιλογή για startups, μικρές επιχειρήσεις και οργανισμούς που λαμβάνουν υπόψη τους το κόστος. Με τον τρόπο αυτό, το serverless επιτρέπει στις επιχειρήσεις να βελτιστοποιήσουν τον προϋπολογισμό τους και να διαθέτουν πόρους πιο αποτελεσματικά.

Τέλος, ένα από τα πιο βασικά χαρακτηριστικά της υπολογιστικής αρχιτεκτονικής serverless, είναι η event-driven αρχιτεκτονική του. Σε αυτήν την παραδοχή, οι λειτουργίες του serverless σχεδιάζονται για να ανταποκρίνονται δυναμικά σε συγκεκριμένα γεγονότα (events). Αυτά τα γεγονότα μπορούν να περιλαμβάνουν μια ευρεία γκάμα δραστηριοτήτων, όπως οι εισερχόμενες HTTP αιτήσεις (HTTP requests), οι αλλαγές στην κατάσταση μιας βάσης δεδομένων ή το upload αρχείων στον χώρο αποθήκευσης του νέφους (cloud). Για παράδειγμα, όταν ένας χρήστης υποβάλλει μία φόρμα σε μια διαδικτυακή εφαρμογή, μια λειτουργία serverless μπορεί να ενεργοποιηθεί για να επεξεργαστεί τα δεδομένα και να παρέχει αμέσως μια απάντηση. Αυτή η αρχιτεκτονική που βασίζεται σε γεγονότα αυξάνει όχι μόνο την αποδοτικότητα των εφαρμογών αλλά επίσης συμβαδίζει απόλυτα με τις σύγχρονες αρχές σχεδιασμού εφαρμογών, καθιστώντας το serverless μια ιδανική επιλογή για την ανάπτυξη σε πραγματικό χρόνο (real-time development), διαδραστικών και ευέλικτων λογισμικών λύσεων.

Το Serverless Computing βασίζεται σε πολλές τεχνολογίες και πλατφόρμες που επιτρέπουν στους προγραμματιστές να δημιουργούν, αναπτύσσουν και διαχειρίζονται εφαρμογές serverless με ευκολία. Ένα από τα πιο γνωστά εργαλεία σε αυτόν τον τομέα είναι το AWS Lambda [19], που παρέχεται από το Amazon Web Services. Το AWS Lambda επιτρέπει στους προγραμματιστές να εκτελούν κώδικα χωρίς την προμήθεια ή διαχείριση διακομιστών. Υποστηρίζει μια ευρεία γκάμα γλωσσών προγραμματισμού, καθιστώντας το ευέλικτο για διάφορους τύπους εφαρμογών.

Ένας άλλος σημαντικός παίκτης είναι το Azure Functions [20] από το Microsoft Azure, που παρέχει μια υπηρεσία υπολογισμού χωρίς διακομιστή και ενσωματώνεται σε άλλες υπηρεσίες του Azure. Το Google Cloud Functions [21], που είναι μέρος της Google Cloud Platform, παρέχει και αυτό μια παρόμοια υπηρεσία με event-driven serverless functions.

Το Knative [22], μια ανοικτού κώδικα πλατφόρμα βασισμένη στο Kubernetes [23] και έχει κερδίσει προσοχή τελευταία. Το Knative αποσυνδέει την πολυπλοκότητα της διαχείρισης της υποκείμενης υποδομής, επιτρέποντας στους προγραμματιστές να επικεντρώνονται μόνο στη λογική της εφαρμογής τους.

Τεχνολογίες όπως το Docker, που αφορούν την υποδομή υπολογισμού σε containers, παίζουν έναν ουσιαστικό ρόλο στον τομέα του serverless computing, επιτρέποντας στους προγραμματιστές να συσκευάζουν (containerize) τις εφαρμογές και τις εξαρτήσεις τους σε containers, τα οποία μπορούν στη συνέχεια να αναπτύσσονται και να εκτελούνται εύκολα σε περιβάλλοντα serverless.

Πλαίσια όπως το OpenFaaS [24] παρέχουν μια πλατφόρμα serverless ανοικτού κώδικα (open source) που μπορεί και αυτή να αναπτυχθεί σε οποιοδήποτε cluster Kubernetes χρησιμοποιώντας το faas-netes

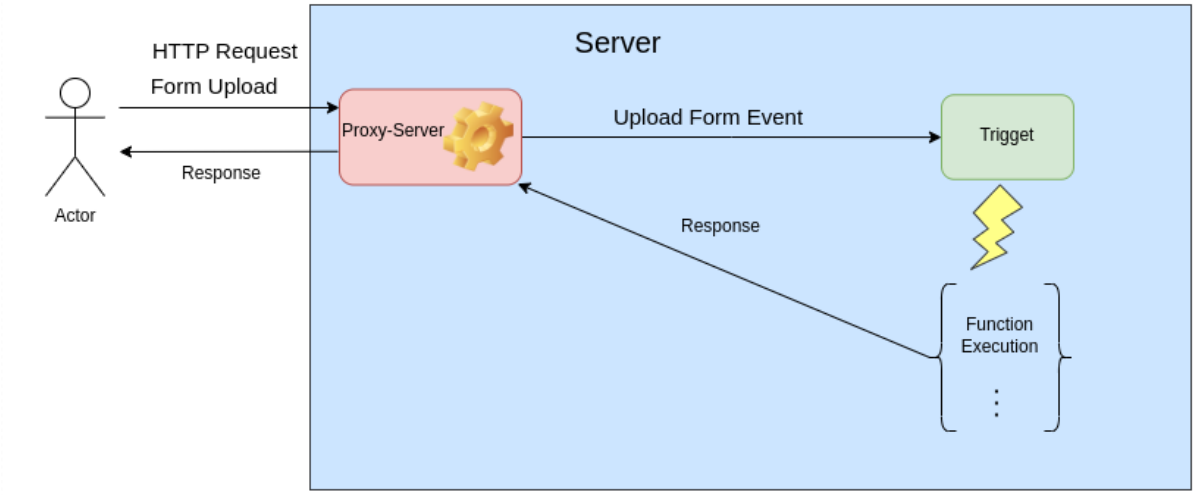


Figure 0.3.4: Παραδειγμα χρησης Serverless function, όπου ενας χρήστης κάνει upload σε έναν server μία φόρμα, μέσω ενός API, όπου τότε, μέσω ενός trigger, γίνεται triggered μία συνάρτηση η οποία επεξεργάζεται τη φόρμα και επιστρέφει στον χρήστη τα αποτελέσματα.

[25]. Το OpenFaaS απλοποιεί τη δημιουργία και διαχείριση των λειτουργιών serverless, κάνοντάς το μια αξιόλογη επιλογή για οργανισμούς που επιθυμούν να υιοθετήσουν την τεχνολογία του serverless computing στα δικά τους κέντρα δεδομένων.

Αυτές οι τεχνολογίες και πλαίσια συνθέτουν τα θεμέλια του serverless computing, επιτρέποντας στους προγραμματιστές να αναπτύσσουν εφαρμογές και λειτουργίες χωρίς την ανάγκη να διαχειρίζονται την υποκείμενη υποδομή. Αυτό εξασφαλίζει ευελιξία, αυξημένη αποδοτικότητα και απλοποιημένη διαχείριση, κάνοντας το Serverless ένα από τα πιο σημαντικά εργαλεία στον τομέα της σύγχρονης ανάπτυξης λογισμικού.

Edge Computing

Το Edge Computing αποτελεί έναν διαφορετικό τρόπο επεξεργασίας και διαχείρησης των δεδομένων. Αντίθετα από τον παραδοσιακό υπολογισμό στον "σύννεφο" (cloud computing), όπου τα δεδομένα επεξεργάζονται κεντρικά σε απομακρυσμένα κέντρα δεδομένων, το edge computing κατανέμει τους υπολογιστικούς πόρους πιο κοντά στην πηγή των δεδομένων, όπως έναν αισθητήρα, μια συσκευή ή ένα σημείο του Internet of Things (IoT). Αυτή η προσέγγιση της επεξεργασίας δεδομένων στον τόπο προέλευσής τους προσφέρει αρκετά σημαντικά πλεονεκτήματα. Η εικόνα 0.3.5 παρουσιάζει διαγραμματικά τις διαφορές μεταξύ Cloud και Edge Computing.

Ένα από τα κύρια πλεονεκτήματα του edge computing είναι η χαμηλή καθυστέρηση. Επεξεργάζοντας τα δεδομένα τοπικά, οι συσκευές στο edge (edge nodes) μπορούν να παρέχουν απαντήσεις σε πραγματικό χρόνο ή πολύ κοντά σε αυτόν, κάτι που το καθιστά ιδανικό για εφαρμογές όπως τα αυτόνομα οχήματα, η βιομηχανική αυτοματοποίηση και η επαυξημένη πραγματικότητα, όπου ακόμη και η παραμικρή καθυστέρηση μπορεί να έχει κρίσιμες, εώς και θανατηφόρες, συνέπειες. Το διάγραμμα 0.3.6 δείχνει λεπτομερώς την αρχιτεκτονική του Edge Computing και το πως τα Edge devices βρίσκονται πιο κοντά στις πηγές δεδομένων.

Επιπλέον, το edge computing βελτιώνει την ασφάλεια του απορρήτου και την ασφάλεια των δεδομένων. Αντί να μεταδίδονται ευαίσθητες πληροφορίες μεγάλες αποστάσεις προς το "σύννεφο" (cloud), τα δε-

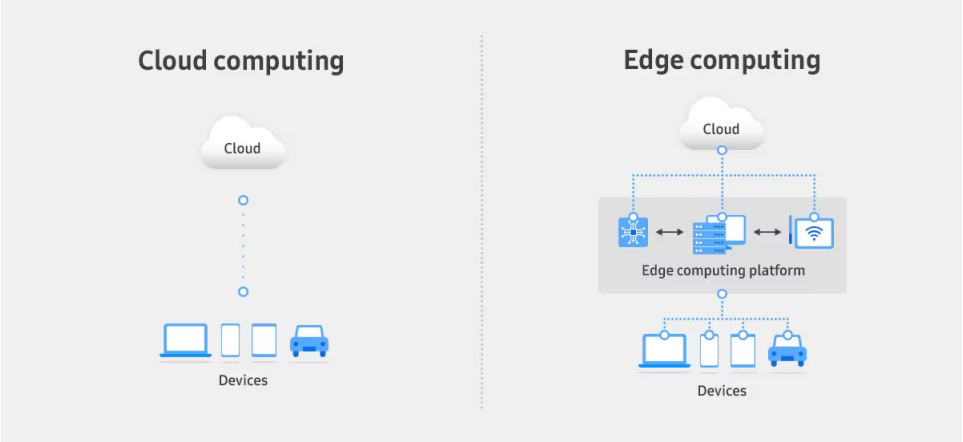


Figure 0.3.5: Edge vs Cloud Computing [26]

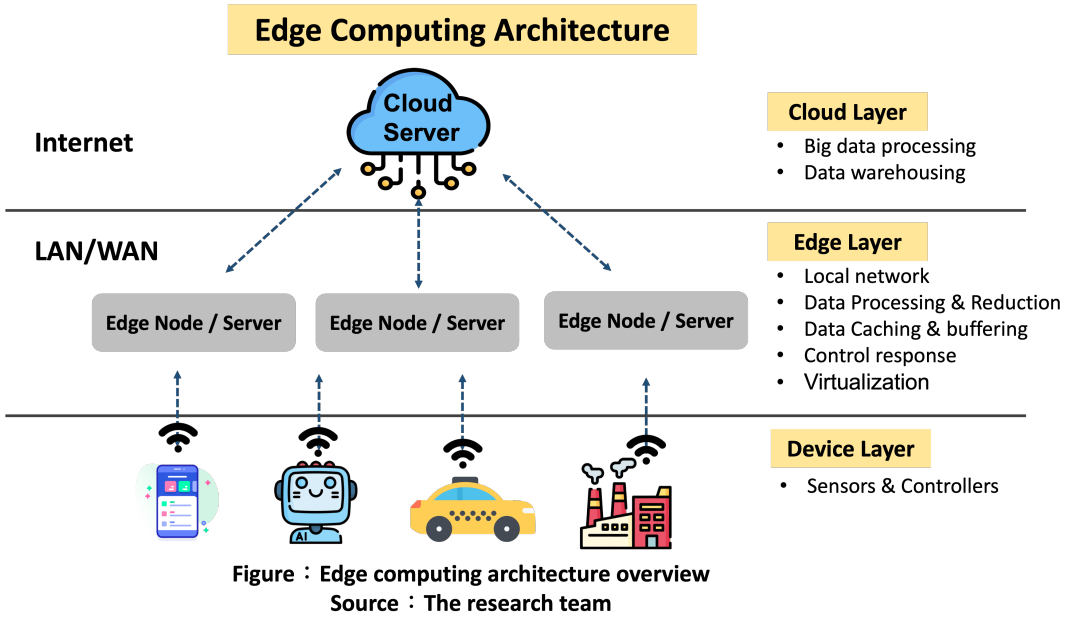


Figure 0.3.6: Edge Computing Architecture [27]

δομένα επεξεργάζονται τοπικά, μειώνοντας τον κίνδυνο διαρροής δεδομένων κατά τη διάρκεια της μετάδοσης. Αυτό είναι ιδιαίτερα σημαντικό σε βιομηχανίες όπως η υγεία και οι οικονομικές υπηρεσίες, όπου η εμπιστευτικότητα των δεδομένων έχει καίρια σημασία.

Ένα ακόμη πλεονέκτημα αφορά την βελτιστοποίηση του εύρους ζώνης. Οι συσκευές στο edge επεξεργάζονται προτού μεταδίδουν τα δεδομένα στο "σύννεφο" (cloud), μειώνοντας τον όγκο των δεδομένων που πρέπει να μεταφερθούν μέσω του δικτύου. Αυτό μπορεί να οδηγήσει σε σημαντικές εξοικονομήσεις στο κόστος, ιδίως σε περιπτώσεις με περιορισμένο εύρος ζώνης.

Για την αποτελεσματική υλοποίηση του edge computing, χρησιμοποιούνται διάφορες τεχνολογίες και πλαίσια. Οι συσκευές στο edge είναι εξοπλισμένες με υπολογιστικούς πόρους, όπως μικροεξυπηρετητές ή μονάδες επεξεργασίας γραφικών γενικού σκοπού (GPGPUs), για να ανταπεξέρχονται στα τοπικά αίτηματα (requests) επεξεργασίας. Επιπλέον, τεχνολογίες containerization όπως το Docker χρησιμοποιούνται για να ανακλείσουν εφαρμογές και τις αναγκαίες τους εξαρτήσεις, διασφαλίζοντας τη φορητότητα και την εύκολη ανάπτυξη σε διάφορα περιβάλλοντα του edge. Το Kubernetes μπορεί να διαχειριστεί αυτές τις containerized εφαρμογές σε edge clusters, διασφαλίζοντας αποδοτική χρήση των πόρων και δυνατότητα επεκτασιμότητας. Τέλος, το Knative επιτρέπει τη διαχείριση και την αυτόματη κλιμάκωση των εφαρμογών λειτουργιών (serverless) στον edge, προσφέροντας μια πλήρη λύση για την ανάπτυξη και την εκτέλεση εφαρμογών σε αυτό το περιβάλλον.

Στο Edge Computing χρησιμοποιείται μια ποικιλία συσκευών που καλύπτουν συγκεκριμένες ανάγκες και σενάρια. Ειδικότερα, συσκευές όπως το Raspberry Pi [28], η σειρά Jetson [29] από την NVIDIA και οι μικροελεγκτές STM32 [30] έχουν αποκτήσει σημαντική θέση στα οικοσυστήματα edge computing. Το Raspberry Pi, ένας συμπαγής και οικονομικός υπολογιστής, αποτελεί μια ευέλικτη επιλογή για πολλές εφαρμογές edge, από την αυτοματοποίηση του σπιτιού έως την τεχνητή νοημοσύνη στο edge. Η οικογένεια Jetson της NVIDIA, από την άλλη, ξεχωρίζει για τις ισχυρές δυνατότητες των integrated GPU της, καθιστώντας την ιδανική για απαιτητικές εργασίες στο edge όπως η υπολογιστική όραση και τα βαθιά νευρωνικά δίκτυα. Οι μικροελεγκτές STM32, γνωστοί για την χαμηλή κατανάλωση ενέργειας και τις δυνατότητες πραγματικού χρόνου επεξεργασίας, βρίσκουν εκτεταμένη χρήση στις εφαρμογές edge IoT, εξασφαλίζοντας αποδοτική συλλογή και επεξεργασία δεδομένων. Αυτές οι συσκευές, μαζί με διάφορες άλλες, συμβάλλουν στον πλούσιο χαρακτήρα της edge computing, ενισχύοντας μια ευρεία γκάμα εφαρμογών με τις εξατομικευμένες τους δυνατότητες και ιδιαιτερότητες.

Το Edge Computing βρίσκει εφαρμογές σε διάφορους τομείς, βελτιώνοντας την αποτελεσματικότητα, την αξιοπιστία και τη λήψη αποφάσεων σε πραγματικό χρόνο. Στον τομέα της υγείας, δίνει τη δυνατότητα για απομακρυσμένη παρακολούθηση ασθενών, όπου τα δεδομένα από ιατρικές συσκευές μπορούν να επεξεργαστούν τοπικά, εξασφαλίζοντας άμεσες ειδοποιήσεις και μειώνοντας την καθυστέρηση σε κρίσιμες καταστάσεις. Στα αυτόνομα οχήματα, edge συσκευές επεξεργάζονται τα δεδομένα αισθητήρων άμεσα, επιτρέποντας γρήγορες αντιδράσεις και βελτιωμένη ασφάλεια. Οι έξυπνες πόλεις επωφελούνται από το Edge Computing, βελτιστοποιώντας τη διαχείριση της κίνησης, αναλύοντας περιβαλλοντικά δεδομένα και επιτρέποντας έξυπνες δημόσιες υπηρεσίες. Στις βιομηχανικές εφαρμογές, διευκολύνει την προβλεπτική συντήρηση, μειώνοντας το χρόνο αδράνειας του εξοπλισμού και αυξάνοντας την παραγωγικότητα. Τέλος, το Edge Computing διαδραματίζει κεντρικό ρόλο στις εμπειρίες επαυξημένης και εικονικής πραγματικότητας (Augmented Reality), μειώνοντας την καθυστέρηση επεξεργάζοντας τα δεδομένα τοπικά. Αυτά τα ποικίλα παραδείγματα υπογραμμίζουν την ευελιξία και την μετασχηματιστική δυνατότητα του Edge Computing σε διάφορους τομείς.

0.4 Serverless Framework και Run-Time Scheduler

Όπως έχει ήδη αναφερθεί, αυτή η διπλωματική εργασία εξερευνά και παρουσιάζει ένα βελτιστοποιημένο framework που αξιοποιεί τεχνολογίες όπως το Kubernetes, το Knative, το Deep Learning, και πιο συγκεκριμένα το Reinforcement Learning, για να αντιμετωπίσει την περίπλοκη αλληλεπίδραση μεταξύ της αποτελεσματικής εκτέλεσης και του δυναμικού προγραμματισμού σε αποσπώμενες αναπτύξεις νευρωνικών δικτύων.

0.4.1 Serverless Framework για layered και full offloading των DNNs

Το framework χρησιμοποιεί το Kubernetes και το Knative για τη δημιουργία ενός ευέλικτου περιβάλλοντος για την εκτέλεση νευρωνικών δικτύων. Το Kubernetes, παρέχει την βάση πάνω στην οποία βασίζεται το framework. Επιτρέπει την ανάπτυξη και την κλιμάκωση των επιπέδων νευρωνικού δικτύου σε ένα κατανεμημένο σύνολο συσκευών. Το Knative προσθέτει στο Kubernetes ένα πλαίσιο serverless για την ανάπτυξη και τη διαχείριση εφαρμογών, διασφαλίζοντας αποτελεσματική διανομή πόρων. Μαζί, αυτές οι πλατφόρμες παρέχουν ένα ευέλικτο και κλιμάκωσιμο σύστημα για την εκτέλεση νευρωνικών δικτύων. Το framework απλοποιεί τη διαχείριση των πολυπλοκοτήτων της πολυσυσκευασμένης, πολυπλευρικής εκτέλεσης και παρέχει υποστήριξη για περιβάλλοντα CPU και GPU, επιτρέποντας τη βέλτιστη χρήση των πόρων και τη διαχείριση της ανάπτυξης νευρωνικών δικτύων σε διάφορα υπολογιστικά περιβάλλοντα.

0.4.1.1 Υπηρεσία Εκτέλεσης επιπέδων Νευρωνικών Δικτύων

Στο επικέντρο του framework βρίσκεται στο Service που εκτελεί τα επίπεδα (layers) των νευρωνικών δικτύων. Μια συνοπτική ανάλυση των βασικών ενεργειών που εκτελεί η Υπηρεσία Εκτέλεσης Επιπέδων:

- **Χειρισμός Συμβάντων (Events)**: Η υπηρεσία ακούει για εισερχόμενα CloudEvents μέσω αιτημάτων HTTP, χρησιμοποιώντας το Flask [31] για τη λήψη και την επεξεργασία των συμβάντων.
- **Εκτέλεση Πολλαπλών Διεργασιών**: Κάθε εισερχόμενο CloudEvent δημιουργεί μία αφιερωμένη διεργασία, συγκεκριμένα για αυτό το CloudEvent, επιτρέποντας την ταυτόχρονη επεξεργασία για τη βέλτιστη διαχείριση των πόρων.
- **Προεπεξεργασία Δεδομένων**: Τα ακατέργαστα δεδομένα εισόδου ανασχηματίζονται και μετατρέπονται χρησιμοποιώντας τη NumPy [32] βιβλιοθήκη για να ταιριάζουν με την καθορισμένη μορφή που απαιτεί το κάθε νευρωνικό δκτυο.
- **Εκτέλεση Επιπέδου**: Το PyTorch [33] χρησιμοποιείται για την αποτελεσματική εκτέλεση των επιπέδων του νευρωνικού δικτύου.
- **Προγραμματισμός του Επόμενου Επιπέδου**: Η υπηρεσία επικοινωνεί με την υπηρεσία προγραμματισμού (scheduling service) για να αποφασίσει πού πρέπει να εκτελεστεί το επόμενο επίπεδο, βάσει της διαθεσιμότητας πόρων και των στρατηγικών βελτιστοποίησης.
- **Διασυσκευαστική Επικοινωνία**: Εάν το επόμενο επίπεδο ανατεθεί σε διαφορετική συσκευή, η υπηρεσία σειριοποιεί τα δεδομένα εξόδου, δημιουργεί ένα νέο CloudEvent και το αποστέλλει στην καθορισμένη συσκευή, ακολουθώντας την ίδια ακολουθία ενεργειών.

Η εικόνα 0.4.1 δείχνει με διαγραμματικό τρόπο το πως εκτελούνται τα παραπάνω βήματα της Υπηρεσίας και με ποια σειρά.

Ενα κρίσιμο μέρος της λειτουργίας της υπηρεσίας αποτελεί **η Διαδικασία Προετοιμασίας (Warmup Process)**, που εξασφαλίζει αποτελεσματική αρχικοποίηση και διάθεση πόρων, αντιμετωπίζοντας τις χρονοκαθυστερήσεις από τα λεγόμενα *cold starts* που μπορεί να επηρεάσουν την αποκρισιμότητα της υπηρεσίας. Αυτή η διαδικασία περιλαμβάνει τη φόρτωση και εκτέλεση των απαιτούμενων επιπέδων του μοντέλου για να προετοιμάσει την υπηρεσία για βέλτιστη επιδόσεις μέσω πολλαπλών κύκλων εκτέλεσης.

0.4.1.2 Υπηρεσία Εκτέλεσης Νευρωνικών Δικτύων

Επιπλέον, έχει υλοποιηθεί ακόμα μία υπηρεσία στο framework μας που ακολουθεί την ίδια βασική λογική με αυτή που συζητήθηκε προηγουμένως, ωστόσο, η βασική διαφορά μεταξύ αυτών των δύο υπηρεσιών είναι ότι ενώ η πρώτη υπηρεσία ζητά από τον server να καθορίσει πού θα εκτελεστεί το επόμενο επίπεδο ενός νευρωνικού δικτύου, η δεύτερη υπηρεσία εκτελεί ολόκληρο το νευρωνικό δίκτυο. Με άλλα λόγια, η δεύτερη υπηρεσία αυτοματοποιεί τη διαδικασία εκτέλεσης του νευρωνικού δικτύου, εξαλείφοντας την ανάγκη επικοινωνίας μεταξύ της υπηρεσίας και του διακομιστή. Το διάγραμμα ροής αυτής της υπηρεσίας παρουσιάζεται στο σχήμα 0.4.2.

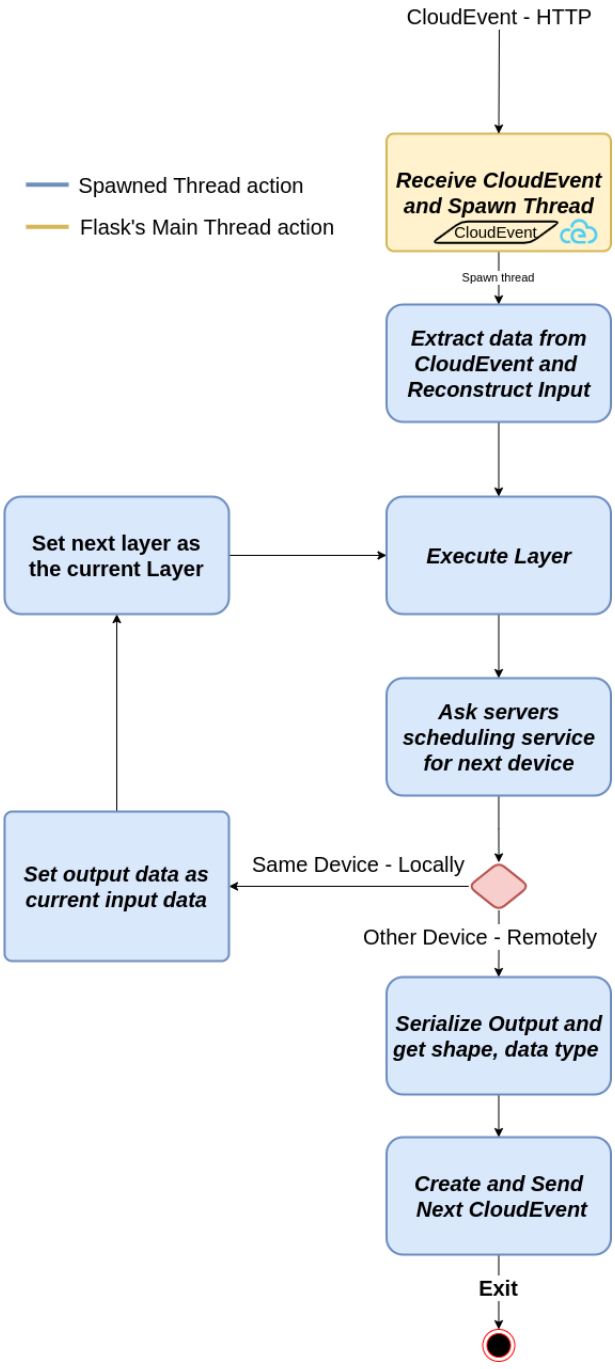


Figure 0.4.1: Τα βασικά στοιχεία της Υπηρεσίας Εκτέλεσης επιπέδων Νευρωνικών Δικτύων καθώς και η σειρά με την οποία εκτελούνται.

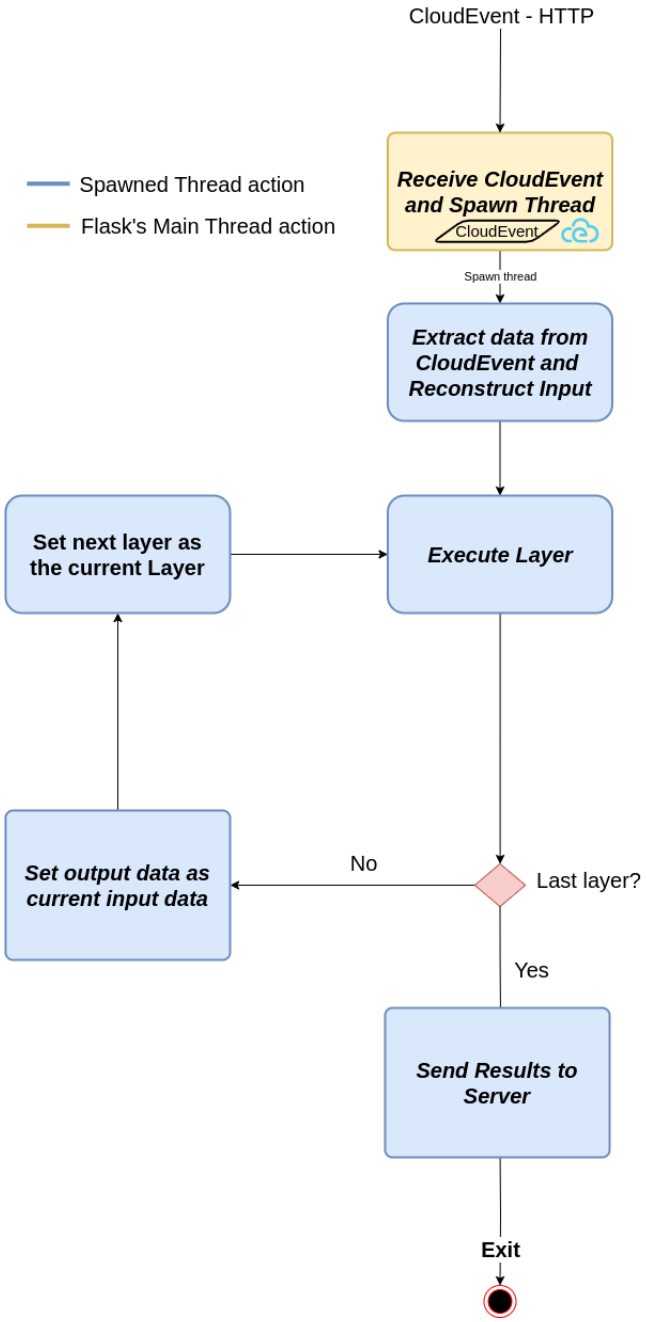


Figure 0.4.2: Whole Neural Network Execution Service Flowchart

0.4.1.3 Υπηρεσία Εκτέλεσης Νευρωνικών Δικτύων στο Cluster

Μετά την ανάπτυξη και τη βελτιστοποίηση του κώδικα της υπηρεσίας για την εκτέλεση νευρωνικών δικτύων, το επόμενο βήμα είναι η ανακατανομή αυτών των υπηρεσιών στο Kubernetes cluster. Η διαδικασία αυτή σχεδιάστηκε για να διασφαλίσει την ομαλή κλιμάκωση, την αποτελεσματική κατανομή πόρων και την ανταπόκριση σε διαφορετικά σενάρια φόρτου από requests, είτε χαμηλά είτε υψηλά. Τα βήματα που κάνουν deploy τη υπηρεσία αυτή στο cluster είναι:

- **Δημιουργία Συστήματος-Cluster**: Αρχικά, δημιουργούμε το cluster Kubernetes-Knative χρησιμοποιώντας το εργαλείο kubeadm CLI [34]. Αυτό περιλαμβάνει την προσθήκη συσκευών στο cluster και τη ρύθμιση στοιχείων όπως το Flannel για το δίκτυο και το Knative για τη διαχείριση εφαρμογών serverless. Επίσης, εγκαθιστούμε το NVIDIA device plugin για το Kubernetes για τη διαχείριση των GPUs.
- **Docker Εικόνες (Images) Υπηρεσίας**: Για να ανακατανείμουμε τις υπηρεσίες, πρέπει να τις ενθυλακώσουμε σε εικόνες Docker containers. Ο κώδικας της υπηρεσίας και οι εξαρτήσεις της ενσωματώνονται σε αυτές τις εικόνες, οι οποίες ανεβαίνουν στο DockerHub για ευκολότερη πρόσβαση και ανακατανομή στο cluster.
- **Αρχεία Ρύθμισης YAML**: Η ανακατανομή των υπηρεσιών γίνεται με τη βοήθεια αρχείων YAML [35]. Κάθε αρχείο ορίζει πώς πρέπει να δημιουργηθεί και να διατηρηθεί μια συγκεκριμένη υπηρεσία. Περιλαμβάνουν ρυθμίσεις περιβαλλοντικών μεταβλητών, περιορισμούς και παραμέτρους κλιμάκωσης, όπως το αριθμό των requests που μπορεί να εξυπηρετήσει παράλληλα ένα container/Kubernetes pod μίας υπηρεσίας. Ένα χαρακτηριστικό παράδειγμα είναι ότι σε αυτά τα αρχεία ορίζεται το *Node Affinity*, που δείχνει τη συσκευή που θα εκτελεστεί η κάθε συνάρτηση, όταν αυτή γίνει triggered.
- **Αυτόματη Κλιμάκωση με το KPA**: Χρησιμοποιούμε το Knative Pod Autoscaler (KPA) [36] για δυναμική κλιμάκωση. Ο KPA παρακολουθεί τις εργασίες του cluster και προσαρμόζει αυτόματα τον αριθμό των containers/kubernetes pods ανάλογα με τη ζήτηση.

Αυτά τα βήματα επιτρέπουν την αποτελεσματική ανακατανομή των υπηρεσιών στο cluster, δίνοντάς μας τη δυνατότητα να αναπτύξουμε αλγορίθμους δρομολόγησεις των εισερχόμενων αιτημάτων (requests) για την αποδοτική τους εξυπηρέτηση.

0.4.1.4 Γενική Αρχιτεκτονική με scheduler

Πάνω από αυτό το πλαίσιο (framework), ένας κεντρικός scheduler μπορεί να καθορίσει σε ποια συσκευή θα εκτελεστεί ένα νευρωνικό δίκτο ή ένα από τα επίπεδά του. Αυτή η ολοκληρωμένη αρχιτεκτονική απεικονίζεται στο σχήμα 0.4.3.

0.5 Reinforcement Learning based Scheduler

Η οργάνωση του framework επεκτείνεται και στο scheduling των αιτημάτων, επικεντρώνοντας την προσοχή στον προγραμματισμό των πόρων και στις τεχνικές ενισχυτικής μάθησης. Κατά τη διάρκεια αυτής της διαδικασίας, το framework προσαρμόζεται σε διάφορα σενάρια υπολογιστικού φόρτου. Η ενισχυτική μάθηση αποτελεί τη βάση που επιτρέπει στο framework να λαμβάνει αποφάσεις σχετικά με τη διανομή των επιπέδων του νευρωνικού δικτύου. Πρόκειται για περισσότερο από απλή διαχείριση πόρων, αφού αποτελεί έναν δυναμικό προγραμματιστή που αναλύει το περιβάλλον, αναθέτει πόρους και διασφαλίζει την αποτελεσματική εκτέλεση του νευρωνικού δικτύου.

Τα θεμέλια της ενισχυτικής Μάθησης (Reinforcement Learning - RL) εστιάζουν σε έναν πράκτορα (agent), ένα περιβάλλον (environment) και ένα σύστημα ανταμοιβών (rewards). Στο RL, ο πράκτορας μαθαίνει να λαμβάνει ακολουθία αποφάσεων με σκοπό να μεγιστοποιήσει τις συσσωρευτικές ανταμοιβές σε ένα περιβάλλον. Αυτό συμβαίνει με τον πράκτορα να αλληλεπιδρά με το περιβάλλον, λαμβάνοντας μέτρα και ανταλλάσσοντας ανατροφοδοτήσεις σε μορφή ανταμοιβών ή ποινών.

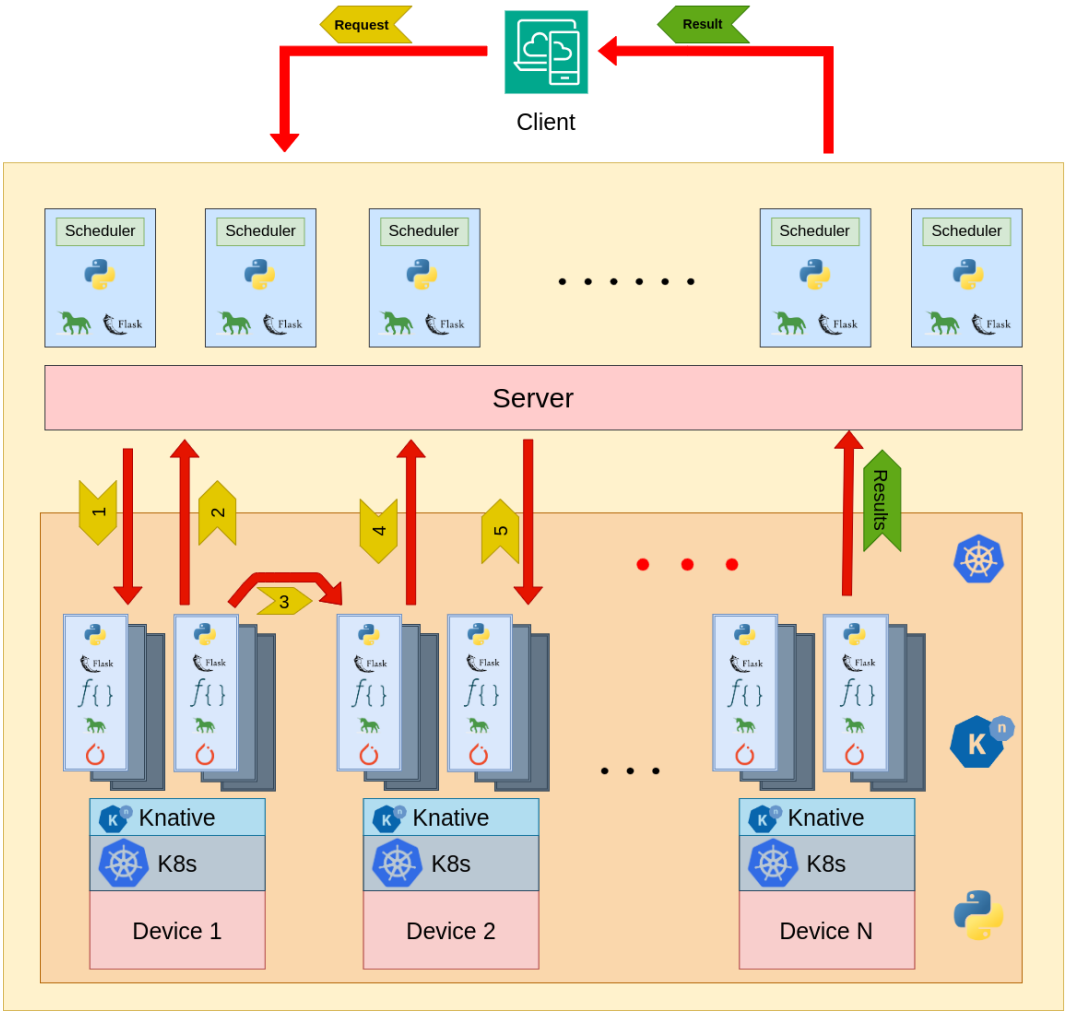


Figure 0.4.3: Αρχιτεκτονική του πλαισίου (framework) μαζί με κεντρικό scheduler

Σε αυτήν την εργασία, σκοπός είναι η ανάπτυξη ενός δυναμικού και προσαρμοστικού μηχανισμού προγραμματισμού πόρων με χρήση τεχνικών RL. Αυτός ο μηχανισμός αποσκοπεί στην έξυπνη διαχείριση των πόρων στα επίπεδα των νευρωνικών δικτύων σε πραγματικό χρόνο, με γνώμονα τη βελτιστοποίηση της απόδοσης και την ανταπόκριση σε μεταβαλλόμενα υπολογιστικά φορτία στο Edge.

Ορισμός Χώρου Παρατήρησης (Περιβάλλον)

Ο χώρος παρατήρησης έχει σημαντικό ρόλο στην αποτελεσματικότητα για την εκπαίδευση του RL αλγορίθμου. Μέσω του χώρου-κατάσταση παρατήρησης, ο πράκτορας RL αντιλαμβάνεται το περιβάλλον και λαμβάνει αποφάσεις. Η κατάσταση αυτή αποτελείται από ποικίλα δεδομένα και παραμέτρους και αποτελεί τη βάση για την έξυπνη κατανομή πόρων.

Κεντρικό στοιχείο για την αποτελεσματικότητα και την πληροφοριακή αξία της κατάστασης παρατήρησης αποτελεί μια περιεκτική προσέγγιση για την παρακολούθηση και τη συλλογή δεδομένων. Σε αυτό το πλαίσιο, χρησιμοποιούνται δύο διεπαφές γραμμής εντολών (CLIs), το tegrastats [37] και το perf [38], για την καταγραφή μετρήσεων από το υπολογιστικό περιβάλλον (δηλ. υπολογιστικές συσκευές). Αυτά τα εργαλεία αποτελούν σημαντικά στοιχεία της στρατηγικής παρακολούθησης, επιτρέποντας πραγματικού χρόνου ενημερώσεις σχετικά με τη χρησιμοποίηση πόρων, τη συμπεριφορά του υλικού και την αποτελεσματικότητα εκτέλεσης.

Επιπλέον, η προσέγγιση προς τον χώρο παρατήρησης επεκτείνεται πέρα από την απλή συλλογή δεδομένων κατά τη διάρκεια της εκτέλεσης. Αναγνωρίζεται η κρίσιμη σημασία του profiling των στρωμάτων νευρωνικών δικτύων για την κατανόηση των χαρακτηριστικών τους.

Σε αυτήν την κατεύθυνση, ο χώρος παρατήρησης διαιρείται σε δύο διακριτές περιοχές: την online φάση και την offline φάση.

Online Περίοδος - Παρακολούθηση κατά την Εκτέλεση (Runtime Monitoring)

- **Εργαλείο Perf CLI**: Κατά την online φάση, όπου λαμβάνονται δυναμικές αποφάσεις, χρησιμοποιείται το εργαλείο Perf CLI για τη συλλογή κρίσιμων μετρικών από κάθε συσκευή, συμπεριλαμβανομένων:
  - **IPC (Εντολές ανά κύκλο)**: Μέτρηση της αποδοτικότητας της εκτέλεσης εντολών CPU.
  - **Αστοχίες Προσπέλασης Μνήμης (Cache Misses)**: Αξιολόγηση των μοτίβων πρόσβασης στη μνήμη.
  - **Context Switches**: Παρακολούθηση των context switches για αποτελεσματική διαχείριση των πόρων της CPU.
  - **Page Faults**: Παρακολούθηση της διαχείρισης μνήμης.
- **Εργαλείο Tegrastats CLI (Ειδικό για το Jetson)**: Στις πλατφόρμες Jetson [29], χρησιμοποιείται επίσης το εργαλείο Tegrastats CLI για την καταγραφή μιας ποικιλίας δεικτών από κάθε συσκευή, περιλαμβανομένων:
  - **Κατανάλωση Ισχύος (Power Consumption)**: Μέτρηση της κατανάλωσης ισχύος για ενεργειακή αποδοτικότητα.
  - **Χρήση Μνήμης (RAM Utilization)**: Παρακολούθηση της χρήσης μνήμης για βέλτιστη διανομή πόρων.
  - **Χρήση CPU (Μέση τιμή και Τυπική Απόκλιση)**: Ανάλυση CPU φορτίου για ανίχνευση υπερφόρτισης μιας συσκευής.
  - **Συχνότητα CPU (Μέση και Τυπική Απόκλιση)**: Παρακολούθηση των ταχυτήτων ρολογιού CPU για δυναμική κατανομή.
  - **Χρήση GPU (GPU Utilization)**: Αξιολόγηση του φορτίου εργασίας της GPU για αποτελεσματική αντιστοίχιση εργασιών.
  - **Συχνότητα GPU (GPU Frequency)**: Παρακολούθηση των ταχυτήτων ρολογιού GPU για βέλτιστη απόδοση.
- **Τρέχουσες Καταστάσεις Ισχύος των Συσκευών (Devices Current Power Modes)**: Οι τρέχουσες καταστάσεις ισχύος των συσκευών συμπεριλαμβάνονται στα εισαγόμενα δεδομένα που παρέχονται στον πράκτορα RL.
- **Συσκευή Εκτέλεσης του Προηγούμενου Στρώματος (Previous Layer Execution Device)**: Επίσης, περιλαμβάνεται πληροφορία για τη συσκευή όπου εκτελέστηκε το προηγούμενο στρώμα. Αυτή η πληροφορία βοηθά στη μείωση της καθυστέρησης του δικτύου, επιτρέποντας στον πράκτορα RL να λαμβάνει ενημερωμένες αποφάσεις σχετικά με την τοποθέτηση του επόμενου στρώματος.

Offline Περίοδος - Profiling

- **Profiling των Στρωμάτων (Layer Profiling) και ολόκληρων των DNN**: Η περίοδος offline περιλαμβάνει το profiling των μεμονωμένων στρωμάτων αλλά και ολόκληρων των νευρωνικών δικτύων, όπου αξιολογείται η απόδοσή τους. Κατά τη διάρκεια του profiling, εξάγονται οι ίδιες μετρήσεις απόδοσης όπως και κατά τη διάρκεια της online φάσης.

- **Περιβάλλον CPU και GPU (CPU and GPU Context)**: Το profiling πραγματοποιείται τόσο στο πλαίσιο της CPU όσο και της GPU, παρέχοντας πληροφορίες για το πώς τα διάφορα στρώματα και νευρωνικά δίκτυα αποδίδουν σε διάφορο hardware.
- **Profiling Στρωμάτων και ολόκληρων των DNN σε Διάφορες Καταστάσεις Ισχύος (Layer Profiling Across Power Modes)**: Το profiling δεν περιορίζεται σε μια μόνο κατάσταση ισχύος, αλλά ανατείνεται σε όλες τις διαθέσιμες καταστάσεις ισχύος των συσκευών. Αυτή η συνολική προσέγγιση εξασφαλίζει ότι η απόδοση των στρωμάτων αξιολογείται υπό διάφορες λειτουργικές συνθήκες. Το profiling σε διάφορες καταστάσεις ισχύος παρέχει πληροφορίες για το πώς τα στρώματα και ολόκληρα τα DNN συμπεριφέρονται σε περιπτώσεις διαφορετικής κατανάλωσης ισχύος και ικανοτήτων απόδοσης, επιτρέποντας στο πλαίσιο να προσαρμόζεται έξυπνα στα ειδικά χαρακτηριστικά κάθε συσκευής.

  Αυτή η πολυδιάστατη αξιολόγηση (στο πλαίσιο εκτέλεσης {CPU, GPU} και σε διάφορες καταστάσεις ισχύος των συσκευών) αποτελεί σημαντικό στοιχείο στη στρατηγική για την αποτελεσματική και προσαρμοστική εκτέλεση νευρωνικών δικτύων.

Εξισώσεις

Σε μαθηματική μορφή, υποθέτοντας ότι ο αριθμός των συσκευών είναι ίσος με N, η κατάσταση παρατήρησης για ένα τυχαίο στρώμα ενός τυχαίου νευρωνικού δικτύου, το οποίο έγινε profiled σε κάθε συσκευή, σε κάθε κατάσταση ισχύος και σε κάθε πλαίσιο εκτέλεσης {CPU, GPU}, θα ήταν:

Για κάθε συσκευή D_i όπου $i \in 0...N$, το τρέχον διάνυσμα κατάστασης δίνεται από τη φάση online ως εξής:

$$D_i \leftarrow \begin{pmatrix} IPC_i \\ CacheMisses_i \\ ContextSwitches_i \\ PageFaults_i \\ Power_i \\ RAM_i \\ CPUs\_mean\_util_i \\ CPUs\_std\_util_i \\ CPUs\_mean\_freq_i \\ CPUs\_std\_freq_i \\ GPU\_util_i \\ GPU\_freq_i \\ Power\_Mode_i \end{pmatrix} \tag{0.5.1}$$

Για κάθε συσκευή D_i όπου $i \in 0...N$, το profiled διάνυσμα στρώματος ή του νευρωνικού δικτύου για το πλαίσιο εκτέλεσης c όπου $c \in \{CPU, GPU\}$ και την κατάσταση ισχύος pm δίνεται ως εξής:

$$D_{i\_c\_pm} \leftarrow \begin{pmatrix} IPC_{i\_c\_pm} \\ CacheMisses_{i\_c\_pm} \\ ContextSwitches_{i\_c\_pm} \\ PageFaults_{i\_c\_pm} \\ Power_{i\_c\_pm} \\ RAM_{i\_c\_pm} \\ CPUs\_mean\_util_{i\_c\_pm} \\ CPUs\_std\_util_{i\_c\_pm} \\ CPUs\_mean\_freq_{i\_c\_pm} \\ CPUs\_std\_freq_{i\_c\_pm} \\ GPU\_util_{i\_c\_pm} \\ GPU\_freq_{i\_c\_pm} \\ ExecutionTime_{i\_c\_pm} \end{pmatrix} \tag{0.5.2}$$

Αυτό το διάνυσμα υπολογίζεται για κάθε συσκευή D_i όπου $i \in 0...N$, κάθε κατάσταση ισχύος pm και κάθε πλαίσιο εκτέλεσης c όπου $c \in \{CPU, GPU\}$.

Συνδυάζοντας κατακόρυφα όλα τα διανύσματα από την offline φάση, τα N διανύσματα από την online φάση, καθώς και τη συσκευή εκτέλεσης του προηγούμενου στρώματος (με χρηση one-hot κωδικοποίησης), δημιουργείται η κατάσταση παρατήρησης.

Είναι σημαντικό να σημειωθεί ότι όλες οι τιμές κανονικοποιούνται με την κανονικοποίηση min-max προκειμένου να βρίσκονται στην ίδια κλίμακα.

Ορισμός Χώρου Δράσης

Ο χώρος δράσης (action space) στην ενισχυτική μάθηση (reinforcement learning - RL) αντιπροσωπεύει το σύνολο των αποφάσεων που μπορεί να λάβει το μοντέλο RL. Αυτές οι αποφάσεις μπορεί να είναι αρκετά λεπτομερείς και να περιλαμβάνουν επιλογές, όπως η επιλογή της συσκευής προορισμού για τον επόμενο επίπεδο, η καθορισμός της λειτουργικής κατάστασης για αυτήν τη συσκευή και ο καθορισμός του πλαισίου εκτέλεσης (CPU ή GPU). Αυτές οι λεπτομερείς αποφάσεις είναι σημαντικές για την ακριβή και προσαρμοστική κατανομή πόρων.

Ωστόσο, ο χώρος δράσης μπορεί να γίνει σημαντικά μεγάλος, απαιτώντας έναν τεράστιο αριθμό βημάτων εκπαίδευσης για το μοντέλο προκειμένου να μάθει. Για να αντιμετωπίσουμε αυτήν την πολυπλοκότητα και να διευκολύνουμε την πρακτική εκπαίδευση, μπορούμε να εξετάσουμε απλοποιημένες παραλλαγές του χώρου δράσης. Αυτοί οι απλοποιημένοι χώροι δράσης μπορεί να περιλαμβάνουν την επιλογή μόνο της επόμενης συσκευής (χωρίς να λαμβάνονται υπόψη η λειτουργική κατάσταση και το πλαίσιο εκτέλεσης), την επιλογή τόσο της συσκευής όσο και του πλαισίου εκτέλεσης, ή τον καθορισμό μόνο της συσκευής και της λειτουργικής κατάστασης.

Επιλογή Μόνο της Επόμενης Συσκευής:
Σε αυτήν την περίπτωση, με σταθερό πλαίσιο εκτέλεσης και λειτουργικές καταστάσεις των συσκευών, ο χώρος δράσης είναι ένα διάνυσμα AS με μήκος ίσο με τον αριθμό των συσκευών. Κάθε θέση $AS[i]$ αντιπροσωπεύει την πιθανότητα επιλογής της συσκευής i:

$$AS \leftarrow \begin{pmatrix} Device_0 \\ Device_1 \\ \vdots \end{pmatrix}$$

Επιλογή της Επόμενης Συσκευής και του Πλαισίου Εκτέλεσης:
Όταν επιλέγουμε τόσο την επόμενη συσκευή όσο και το πλαίσιο εκτέλεσης, ο χώρος δράσης είναι ένα διάνυσμα AS με μήκος ίσο με διπλάσιο αριθμό των συσκευών:

$$AS \leftarrow \begin{pmatrix} Device_{0_{cpu}} \\ Device_{0_{gpu}} \\ Device_{1_{cpu}} \\ Device_{1_{gpu}} \\ \vdots \end{pmatrix}$$

Εδώ, αν $i \mod 2 == 0$, η θέση $AS[i]$ αντιπροσωπεύει την πιθανότητα επιλογής της συσκευής $D = i//2$ στο πλαίσιο CPU. Αντίθετα, αν $i \mod 2 == 1$, η θέση $AS[i]$ αντιπροσωπεύει την πιθανότητα επιλογής της συσκευής $D = i//2$ στο πλαίσιο GPU.

Επιλογή της Επόμενης Συσκευής και της Λειτουργικής Κατάστασης της Συσκευής:
Σε αυτό το σενάριο, ο χώρος δράσης είναι ένα διάνυσμα AS με μήκος ίσο με τον αριθμό των συσκευών

πολλαπλασιασμένο με τον αριθμό των λειτουργικών καταστάσεων:

$$AS \leftarrow \begin{pmatrix} Device_{0_p m_0} \\ Device_{0_p m_1} \\ Device_{0_p m_2} \\ \vdots \\ Device_{1_p m_0} \\ Device_{1_p m_1} \\ Device_{1_p m_2} \\ \vdots \end{pmatrix}$$

Υποθέτοντας ότι για κάθε συσκευή υπάρχουν P λειτουργικές καταστάσεις, το $AS[i]$ αντιπροσωπεύει την πιθανότητα επιλογής της συσκευής $D = i//P$ στη λειτουργική κατάσταση $PM = i \mod P$.

Επιλογή και των Δύο Πλαισίων και των Λειτουργικών Καταστάσεων Μαζί με τις Συσκευές:
Στο πιο πολύπλοκο σενάριο, τόσο το πλαίσιο εκτέλεσης όσο και οι λειτουργικές καταστάσεις κωδικοποιούνται στον χώρο δράσης μαζί με τις συσκευές. Ο χώρος δράσης διαμορφώνεται απλώς συνδυάζοντας τα δύο προηγούμενα παραδείγματα:

$$AS \leftarrow \begin{pmatrix} Device_{0_p m_{0_c pu}} \\ Device_{0_p m_{0_g pu}} \\ Device_{0_p m_{1_c pu}} \\ Device_{0_p m_{1_g pu}} \\ \vdots \\ Device_{1_p m_{0_c pu}} \\ Device_{1_p m_{0_g pu}} \\ Device_{1_p m_{1_c pu}} \\ Device_{0_p m_{1_g pu}} \\ \vdots \end{pmatrix}$$

Εδώ, αν $i \mod 2 == 0$, η θέση $AS[i]$ αντιπροσωπεύει την πιθανότητα επιλογής της συσκευής $D = i//(2P)$, στη λειτουργική κατάσταση $P = (i//2) \mod P$ στο πλαίσιο CPU. Αντίθετα, αν $i \mod 2 == 1$, η θέση $AS[i]$ αντιπροσωπεύει την πιθανότητα επιλογής της συσκευής $D = i//(2P)$, στη λειτουργική κατάσταση $P = (i//2) \mod P$ στο πλαίσιο GPU.

0.6 Συνάρτηση Ανταμοιβής

Η συνάρτηση ανταμοιβής οδηγεί το μοντέλο ενισχυτικής μάθησης και σκοπό έχει να καθοδηγήσει τις ενέργειες προς την κατεύθυνση της βελτίωσης της απόδοσης της εκτέλεσης του νευρωνικού δικτύου. Η δημιουργία μιας αποτελεσματικής συνάρτησης ανταμοιβής είναι πολύπλοκη και απαιτεί βαθιά κατανόηση των στόχων του πλαισίου και των λεπτομερειών του edge computing. Στόχος της είναι να διασφαλίσει τη συνεχή εκπλήρωση ή υπέρβαση της Συμφωνίας Επίπεδου Υπηρεσίας (SLA). Για να επιτευχθεί αυτό, καθορίζουμε ένα κατώτατο κατώφλι απόδοσης κατά τη διάρκεια του προφιλαρίσματος. Ο πρωταρχικός στόχος είναι να βρίσκεται ο χρόνος εκτέλεσης των στρωμάτων κάτω από αυτό το κατώφλι. Η συνάρτηση ανταμοιβής υπολογίζει ανάλογα, ανακονώντας θετική ανταμοιβή για καλύτερη απόδοση και μειωμένη κατανάλωση ενέργειας, ενώ παράλληλα λαμβάνει υπόψη την απόκλιση από το κατώτατο κατώφλι όταν ο χρόνος εκτέλεσης υπερβαίνει αυτό το όριο. Το σύστημα ανταμοιβής αποτελεί βασικό στοιχείο της στρατηγικής εκχώρησης πόρων με βάση την ενισχυτική μάθηση και επιτυγχάνει ισορροπία ανάμεσα στη βελτιστοποίηση της απόδοσης και την αποδοτικότητα στην κατανάλωση ενέργειας.

Ένα παράδειγμα συστήματος ανταμοιβής είναι:

- **Σύστημα Ανταμοιβής με Διαίρεση**:
  - **Αρνητική Ανταμοιβή**: Όταν ο χρόνος εκτέλεσης υπερβαίνει το SLA:

$$neg\_rew \leftarrow layer\_execution\_time - SLA\_time \tag{0.6.1}$$

  - **Θετική Ανταμοιβή**: Όταν ο χρόνος εκτέλεσης είναι μικρότερος ή ίσος με το SLA:

$$pos\_rew \leftarrow 1/(10 \times power\_consumption \times execution\_time) \tag{0.6.2}$$

    Οι τιμές της ανταμοιβής βρίσκονται στο εύρος [-1, 1] επειδή οι χρόνοι και η κατανάλωση ενέργειας κανονικοποιούνται στο εύρος 0-1.

- **Σύστημα Ανταμοιβής με Αφαίρεση**:
  - **Αρνητική Ανταμοιβή**: Όταν ο χρόνος εκτέλεσης υπερβαίνει το SLA:

$$neg\_rew \leftarrow -(layer\_execution\_time/SLA\_time)$$

  - **Positive Reward**: Όταν ο χρόνος εκτέλεσης είναι μικρότερος ή ίσος με το SLA:

$$pos\_rew \leftarrow 1/(10 \times power\_consumption \times execution\_time)$$

    Αξίζει να σημειωθεί ότι κατά την αρχική εκπαίδευση, το μοντέλο εμφανίζει τυχαία συμπεριφορά, οδηγώντας σε μεγάλους χρόνους εκτέλεσης. Αυτό μπορεί να είναι προβληματικό καθώς η τιμή της αρνητικής ανταμοιβής στη αρχή μπορεί να φτάσει την τάξη χιλιάδων, ενώ μετά από λίγη εκπαίδευση την τάξη δεκάδων, θεωρώντας έτσι ότι κάνει πολύ καλή δουλεία ενώ αυτό δεν είναι απραίτητο. Αυτό επιλύεται με τη περικοπή των ανταμοιβών σε σταθερό εύρος, όπως [-10, +10].

Αρχιτεκτονική Μοντέλου Ενισχυτικής Μάθησης

Η ενισχυτική μάθηση περιλαμβάνει μια ποικιλία διαφορετικών αρχιτεκτονικών μοντέλων, καθένα σχεδιασμένο για να αντιμετωπίζει διάφορες προκλήσεις και σενάρια. Ωστόσο, αξίζει να σημειωθεί ότι η επιλογή των αρχιτεκτονικών της ενισχυτικής μάθησης περιορίζεται κάπως από τα εργαλεία που υπάρχουν. Συγκεκριμένα, από τη στιγμή που χρησιμποιείται το πακέτο Stable Baselines3, τα διαθέσιμα μοντέλα είναι συγκεκριμένα. Πιο αναλυτικά, είναι το Advantage Actor-Critic (A2C) [39], το Deep Deterministic Policy Gradient (DDPG) [40], το Deep Q Network (DQN) [41], το Hindsight Experience Replay (HER) [42], το Proximal Policy Optimization (PPO) [43], το Soft Actor-Critic (SAC) [44] και το Twin Delayed DDPG (TD3).

Στην εργασία αυτή, χρησιμοποιήθηκε ο αλγόριθμος Proximal Policy Optimization (PPO) [18]. Ο PPO ξεχωρίζει λόγω λόγω της σταθερότητάς του, της αποδοτικότητάς του στο δείγμα και της προσαρμοστικότητάς του σε υψηλής διάστασης χώρους καταστάσεων. Αυτός αποτελεί το κύριο στοιχείο της στρατηγικής μας για την κατανομή πόρων, επιτρέποντας πραγματική λήψη αποφάσεων σε δυναμικά περιβάλλοντα στην άκρη.

Προσομοίωση Φορτίου

Για προσομοίωση του φορτίου εργασίας που έρχεται κατά την πραγματική εκτέλεση, χρησιμοποιήθηκε ένα μοντέλο μη-ομογενούς διαδικασίας Poisson (Non-Homogeneous Poisson Process - NHPP) [45]. Αυτό επιτρέπει την αναπαραγωγή ποικίλων διαφορετικών σεναρίων φορτίων που μπορούν να εμφανιστούν σε πραγματικά περιβάλλοντα edge computing. Σε αντίθεση με την ομογενή κατανομή Poisson που υποθέτει σταθερό μέσο ρυθμό αιτημάτων (λ παράμετρος της κατανομής Poisson), η μη-ομογενής παραλλαγή εισάγει μεταβλητότητα στο ρυθμό αυτό, αντανακλώντας τη δυναμική φύση των περιβαλλόντων edge computing. Για την προσομοίωση αυτή, χρησιμοποιείται η τεχνική Thinning [46], που επιτρέπει τη δημιουργία μη-ομογενούς διαδικασίας Poisson που αντανακλά δυναμικές αλλαγές στις ρυθμίσεις άφιξης γεγονότων.

0.7 Αξιολόγηση

Σε αυτή την ενότητα παρουσιάζεται η διαδικασία αξιολόγησης που χρησιμοποιήσαμε για να αξιολογήσουμε το πλαίσιο και το μοντέλο προγραμματισμού που περιγράφηκε στις ενότητες 4.1 και 4.2, αντίστοιχα.

Πειραματική Διάταξη

Η πειραματική διάταξη περιλαμβάνει τρία Edge συγκευές και μία εικονική μηχανή (VM).

Συσκευές Edge

Ένα Jetson AGX Xavier, που μια πλατφόρμα υψηλής απόδοσης SoC (system-on-a-chip), παρέχει σημαντική υπολογιστική ισχύ και δύο συσκευές Jetson Xavier NX, που είναι αναγνωρισμένες για την ενεργειακή τους αποδοτικότητα και την καταλληλότητά τους για εφαρμογές στο edge. Αυτές οι συσκευές ακμής είναι υπεύθυνες για την εκτέλεση των διάφορων επιπέδων των νευρωνικών δικτύων με την εκτέλεση της υπηρεσίας (service) που περιγράφεται στην ενότητα 4.1.1. Δεν συμμετέχουν στις διαδικασίες λήψης αποφάσεων που σχετίζονται με τον προγραμματισμό και την κατανομή πόρων. Αντίθετα, εκτελούν τις υπολογιστικές εργασίες όπως καθοδηγούνται από το κεντρικό VM. Αυτή η διάταξη αντιστοιχεί στις πρακτικές συνθήκες του edge computing, όπου οι εργασίες κατανέμονται αποτελεσματικά σε διάφορες συσκευές του edge για τη μείωση της καθυστέρησης και την ενίσχυση της συνολικής απόδοσης του συστήματος.

Κεντρική Ελεγκτική Μονάδα (VM)

Μια εικονική μηχανή (VM) που φιλοξενείται σε ένα κεντρικό server, λειτουργώντας ως κεντρική μονάδα. Το VM διαδραματίζει κρίσιμο ρόλο στον συντονισμό και τη βελτιστοποίηση της διανομής πόρων σε ολόκληρο το δίκτυο. Επιβλέπει και διαχειρίζεται τις αποφάσεις κατανομής πόρων εκτελώντας τον αλγόριθμο ενισχυτικής μάθησης που περιγράφεται στην ενότητα 4.2.

Ο πίνακας 5.1 παρέχει μια λεπτομερή εικόνα των τεχνικών χαρακτηριστικών των συσκευών.

1lightgray

Συσκευή	Επεξεργαστής CPU	L2 Cache	L3 Cache	RAM	GPU
Xavier AGX	8 πυρήνες ARM v8.2 64-Bit	8MB	4MB	32GB LPDDR4X	512 πυρήνες NVIDIA Volta και 64 πυρήνες Tensor
Xavier NX	6 πυρήνες NVIDIA Carmel ARM® v8.2 64-bit	6MB	4MB	8GB LPDDR4x	384 πυρήνες NVIDIA Volta™ GPU και 48 πυρήνες Tensor
VM	16 πυρήνες Intel Xeon (Skylake, IBRS)	64MB	256MB	16GB	-

Table 1: Τεχνικά χαρακτηριστικά διαφορετικών κόμβων Edge και Cloud Server

0.7.1 Προκλήσεις Υλοποίησης

Κατά τη διάρκεια της φάσης υλοποίησης αυτής της έρευνας, αντιμετωπίσαμε μια πληθώρα πολύπλοκων προκλήσεων στην προσπάθειά μας να μεταφέρουμε θεωρητικές έννοιες σε πρακτικές λύσεις.

- **Υποστήριξη GPU - Kubernetes**: Για να υποστηρίζει το Kubernetes εκτέλεση κώδικα σε GPU, είναι απαραίτητη η προσθήκη του NVIDIA Device Plugin [47]. Αυτή η προσθήκη

ενσωματώνει υποστήριξη GPU στο cluster, απαιτώντας συμβατότητα με την έκδοση του λογισμικού Jetpack της NVIDIA [48]. Ειδικότερα, το NVIDIA Device Plugin είναι συμβατό με τις εκδόσεις Jetpack $\geq$ 5.0 (στη πραγματικότητα είναι συμβατό με την nvidia-docker [49] >= 2.0 και nvidia-container-toolkit [50] >= 1.7.0 που εγκαθίστανται στο Jetpack $\geq$ 5.0). Ως εκ τούτου, τα Jetsons πρέπει να αναβαθμιστούν σε μια συμβατή έκδοση Jetpack, όπως το 5.0.1. Επιπρόσθετα, το PyTorch Docker Image που χρησιμοποιείται ως base Image, που είναι το 4t-pytorch:r35.2.1-pth2.0-py3, προέρχεται από τον κατάλογο NGC και επιβάλλει σημαντικές απαιτήσεις χώρου αποθήκευσης (13GB). Δεδομένου του περιορισμένου αποθηκευτικού χώρου EMMC στις συσκευές Jetson, μια πρακτική λύση περιλαμβάνει τη χρήση εξωτερικών USB flash drives. Αυτές οι εξωτερικές μονάδες αποθήκευσης συμβάλλουν στην αύξηση της χωρητικότητας αποθήκευσης, διευκολύνοντας την εγκατάσταση αυτών των μεγάλων Images. Με την υλοποίηση αυτής της λύσης, επιτυγχάνεται αποτελεσματικά η αντιμετώπιση πιθανών περιορισμών αποθήκευσης.

- **Υποστήριξη GPU - Pytorch**: Η φόρτωση μοντέλων στη GPU με το PyTorch οδηγεί σε σημαντική χρήση RAM (περίπου 1-1.5GB), πράγμα που δεν είναι συμβατό με τους περιορισμένους πόρους του edge computing, ειδικότερα στις συσκευές Jetson Xavier NX. Για αυτό τον λόγο, η αξιολόγηση θα πραγματοποιηθεί μόνο στο πλαίσιο εκτέλεσης στην CPU. Μελλοντικές εργασίες θα επιλύσουν αυτό το ζήτημα προκειμένου να υποστηρίζεται, με λιγότερους πόρους, εκτέλεση μοντέλου στη GPU για καλύτερη αποδοτικότητα.

- **Χώρος Ενεργοποίησης Ενίσχυσης Μάθησης**: Στον τομέα της Ενισχυτικής Μάθησης (RL), τα μοντέλα συνήθως πρέπει να εξερευνήσουν το χώρο δράσης για να μάθουν μια καλή πολιτική. Εάν ο χώρος δράσης είναι πολύ μεγάλος, η αποτελεσματική εξερεύνηση μπορεί να πάρει περισσότερο χρόνο, με αποτέλεσμα πιθανώς να αυξηθεί ο χρόνος εκπαίδευσης. Οι πράκτορες μπορεί να χρειαστούν περισσότερες αλληλεπιδράσεις με το περιβάλλον για να εξερευνήσουν και να κατανοήσουν τις συνέπειες διαφορετικών ενεργειών. Για αυτόν τον λόγο, ο χώρος δράσης απλοποιείται με διάσταση ίση με 3, ένα για κάθε συσκευή που καθορίζει μόνο την επόμενη συσκευή όπου θα εκτελεστεί το επόμενο επίπεδο.

- **Tune της Ενισχυτικής Μάθησης**: Το Tuning στην Ενισχυτική Μάθησης είναι μια δύσκολη και χρονοβόρα διαδικασία. Αυτή η μελέτη θα επικεντρωθεί σε έναν περιορισμένο φάσμα εξερεύνησης, χρησιμοποιώντας τέσσερα διαφορετικά συστήματα ανταμοιβής που περιγράφονται στην ενότητα Scheduling. Επιπρόσθετα, η εξερεύνηση των υπερπαραμέτρων θα περιλαμβάνει κυρίως την εξερεύνηση της παραμέτρου Generalized Advantage Estimation Lambda, με συγκεκριμένες τιμές όπως [0.9, 0.7, 0.5, 0.3, 0.1].

0.7.2 Αποτελέσματα Πειραμάτων

Όπως αναφέρθηκε προηγουμένως, οι διενεργηθείσες πειραματικές δοκιμές κάλυψαν εκτενώς δύο συστήματα ανταμοιβής, το σύστημα ανταμοιβής με αφαίρεση και με διαίρεση, περιλαμβάνοντας ένα φάσμα τιμών λάμδα (λ), συγκεκριμένα [0,1, 0,3, 0,5, 0,7, 0,9], όλα εκτελούμενα μέσα στο πλαίσιο (framework) που δημιουργήθηκε ειδικά για αυτή τη μελέτη. Και οι δύο υπηρεσίες, περιλαμβανομένης της εκτέλεσης επίπεδο-προς-επίπεδο και της συνολικής, θα υποστούν λεπτομερή αξιολόγηση. Για να αξιολογήσουμε την απόδοση των αλγορίθμων ενίσχυσης μάθησης, θα χρησιμοποιήσουμε ένα σύνολο πιο απλών schedulers.

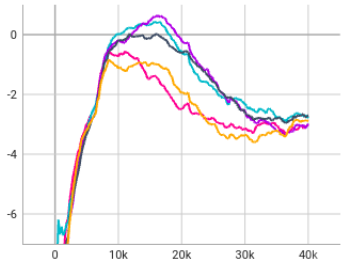
- **Τυχαίος Scheduler**: Δρομολογεί την εκτέλεση νευρωνικών δικτύων στις συσκευές με τυχαίο τρόπο.

- **Round Robin Scheduler**: Δρομολογεί την εκτέλεση νευρωνικών δικτύων στις συσκευές σύμφωνα με τον αλγόριθμο Round Robin.

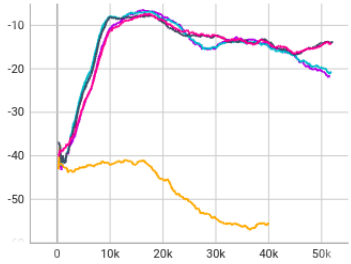
- **Ελάχιστη Χρήση CPU Scheduler**: Δρομολογεί την εκτέλεση νευρωνικών δικτύων στη συσκευή με την ελάχιστη χρήση CPU.

0.7.3 Ανταμοιβή κατά τη διάρκεια της εκπαίδευσης

Αυτή η ενότητα παρουσιάζει τον επεισοδιακό μέσο όρο των ανταμοιβών από σενάρια εκπαίδευσης, προσφέροντας μία εικόνα στις προκλήσεις και την πρόοδο των μοντέλων. Τα σχήματα 0.7.1a και 0.7.1b απεικονίζουν τον επεισοδιακό μέσο όρο των ανταμοιβών για τα συστήματα ανταμοιβής με αφαίρεση και διαίρεση αντίστοιχα, κάνοντας offload ολόκληρο το νευρωνικό δίκτυο για διάφορες τιμές lambda (λ) του Generalized Advantage Estimation. Αρχικά, οι ανταμοιβές βελτιώνονται, αλλά στη συνέχεια μειώνονται. Επιλέγουμε το μοντέλο όταν επιτυγχάνει τη υψηλότερη ανταμοιβή ανά επεισόδιο για να καταγράψουμε την κορυφαία απόδοση πριν αρχίσει οποιαδήποτε αποδιάνωση στη διαδικασία μάθησης. Οι ανταμοιβές στην layered εκτέλεση των νευρωνικών δικτύων ακολουθούν έναν παρόμοιο πρότυπο, και ξανά επιλέγουμε το μοντέλο όταν έχει την κορυφαία ανταμοιβή ανά επεισόδιο.



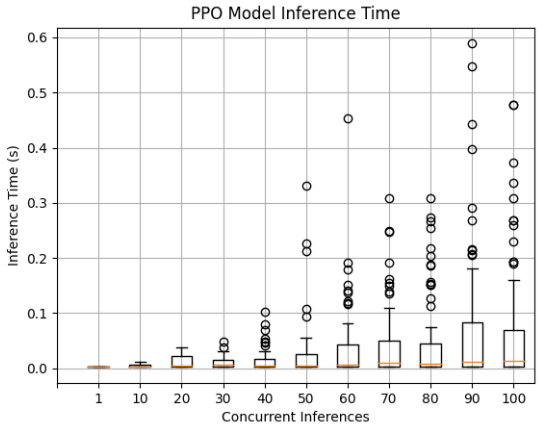
(a) Μέση επεισοδιακή ανταμοιβή με *αφαίρεση*, κάνοντας offload ολόκληρο το νευρωνικό δίκτυο



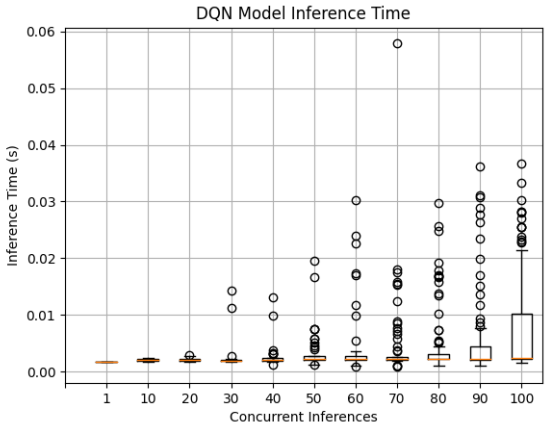
(b) Μέση επεισοδιακή ανταμοιβή με *διαίρεση*, κάνοντας offload ολόκληρο το νευρωνικό δίκτυο

0.7.4 Επιβάρυνση του Scheduler

Χρειάζεται να αξιολογήσουμε τον χρόνο που προστίθεται από τους schedulers ενίσχυσης μάθησης στο σύστημά μας, που περιλαμβάνει την ανίχνευση πολυεπίπεδων νευρωνικών δικτύων (MLP). Στα Σχήματα 0.7.2a και 0.7.2b, παρουσιάζουμε τον πρόσθετο χρόνο για τα μοντέλα PPO και DQN κατά την ανίχνευση. Ακόμη και με 100 ταυτόχρονες ανιχνεύσεις για κάθε μοντέλο, ο μέσος χρόνος ανίχνευσης παραμένει μερικά χιλιοστά δευτερολέπτου. Το μοντέλο PPO είναι πιο αργό από το DQN, με κάποιες περιπτώσεις να αγγίζουν ακόμα και τα 600 χιλιοστά δευτερολέπτου. Αντίθετα, το DQN φτάνει σε μόλις 6 χιλιοστά δευτερολέπτου. Λαμβάνοντας υπόψη τον αμελητέο μέσο χρόνο, δεν τον θεωρούμε σημαντικό για μελλοντικές δοκιμές.



(a) Επιβάρυνση PPO μοντέλου scheduler για διαφορετικό αριθμό από ταυτόχρονες εκτελέσεις



(b) Επιβάρυνση DQN μοντέλου scheduler για διαφορετικό αριθμό από ταυτόχρονες εκτελέσεις

0.7.5 Αξιολόγηση χρόνων και ενέργειας του συστήματος

Σε αυτήν την ενότητα, προβαίνουμε σε εκτενή αξιολόγηση του συστήματος προγραμματισμού μας. Παρουσιάζουμε πειραματικά αποτελέσματα για την απόδοσή του σε πραγματικό χρόνο και την ενεργειακή αποδοτικότητά του. Αυτή η αξιολόγηση είναι κρίσιμη για να αξιολογήσουμε την πρακτική εφαρμοσιμότητα του συστήματός μας σε σχέση με άλλους προγραμματιστές που συζητούνται στην ενότητα 5.3. Μέσα από ολοκληρωμένα πειράματα, στοχεύουμε στην παροχή μιας αντικειμενικής ανάλυσης των δυνατοτήτων του συστήματός μας, αναδεικνύοντας τα πλεονεκτήματα και τους περιορισμούς του. Αυτά τα πειράματα αποτελούν τη βάση για περαιτέρω βελτιστοποίηση του συστήματός μας.

Για την κατανόηση των αποτελεσμάτων, ας εξηγήσουμε την ονοματολογία που χρησιμοποιήθηκε κατά την αξιολόγηση. Σε κάθε διάγραμμα, ο οριζόντιος άξονας αντιστοιχεί στους schedulers. Το πρώτο γράμμα σε κάθε ετικέτα δηλώνει το σύστημα ανταμοιβής που χρησιμοποιήθηκε, ενώ οι επόμενοι χαρακτήρες περιγράφουν τη διαμόρφωση και τα χαρακτηριστικά του μοντέλου.

- S****_lambda_{LAMBDA VALUE}_{MODEL ITERATION}: PPO μοντέλο με σύστημα ανταμοιβής με αφαίρεση, όπου *LAMBDA VALUE* είναι το λ στο GAE και *MODEL ITERATION* είναι το iteration του μοντέλου που διαλέχτηκε
- D****_lambda_{LAMBDA VALUE}_{MODEL ITERATION}: PPO μοντέλο με σύστημα ανταμοιβής με διαίρεση, 'οπου *LAMBDA VALUE* είναι το λ στο GAE και *MODEL ITERATION* είναι το iteration του μοντέλου που διαλέχτηκε
- S****_DQN_lambda_{LAMBDA VALUE}_{MODEL ITERATION} (μόνο στο full offload των νευρωνικών δικτύων): DQN μοντέλο με σύστημα ανταμοιβής με αφαίρεση, όπου *LAMBDA VALUE* είναι το λ στο GAE και *MODEL ITERATION* είναι το iteration του μοντέλου που διαλέχτηκε
- D****_DQN_lambda_{LAMBDA VALUE}_{MODEL ITERATION} (μόνο στο full offload των νευρωνικών δικτύων): DQN μοντέλο με σύστημα ανταμοιβής με διαίρεση, όπου *LAMBDA VALUE* είναι το λ στο GAE και *MODEL ITERATION* είναι το iteration του μοντέλου που διαλέχτηκε
- {DUMMY SHEDULER}: Just the dummy scheduler.
- {DUMMY SCHEDULER}_load: The dummy scheduler with the services that send the devices states running. By default this state runs for the RL based schedulers. We evaluating the dummy schedulers with this load in order to examine the sensitivity of dummy schedulers to external load.

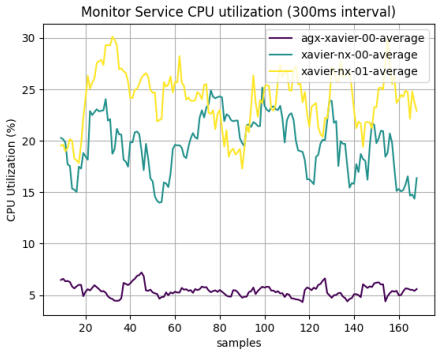
0.7.6 Επιβάρυνση του Monitor Service

Σε αυτήν την υποενότητα επικεντρωνόμαστε αποκλειστικά στην επιβάρυνση του monitor service. Η επίδειξη της επίδρασης του monitor service είναι κρίσιμη για μια σφαιρική αξιολόγηση των επιδράσεών της στην απόδοση του συστήματος, την χρήση πόρων και τη συνολική αποδοτικότητα.

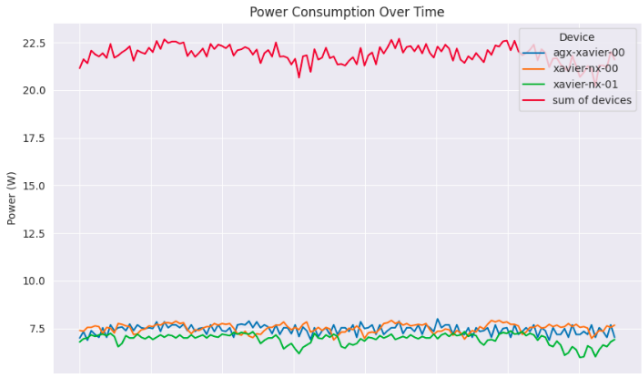
Τα Σχήματα 5.3.3 και 5.3.4 δείχνουν τη χρήση της CPU και την κατανάλωση ενέργειας του monitor service σε όλες τις συσκευές.

Παρατηρώντας τα αποτελέσματα, παρατηρούμε ότι στη συσκευή xavier-nx-00, η χρήση της CPU κυμαίνεται στο εύρος περίπου 20-28%, στη συσκευή xavier-nx-01 κυμαίνεται στο εύρος περίπου 15-25%, ενώ στη συσκευή agx-xavier-00, παραμένει χαμηλότερη περίπου στο 5-6%. Το monitor service έχει μια σχετικά σημαντική επίδραση στην απόδοση των συσκευών NX.

Όσον αφορά την κατανάλωση ενέργειας, σε όλες τις συσκευές, κατά την αποκλειστική λειτουργία του monitor service, η κατανάλωση ενέργειας παραμένει σταθερά στα περίπου 7,5 Watt.



(a) Monitor Service χρήση CPU



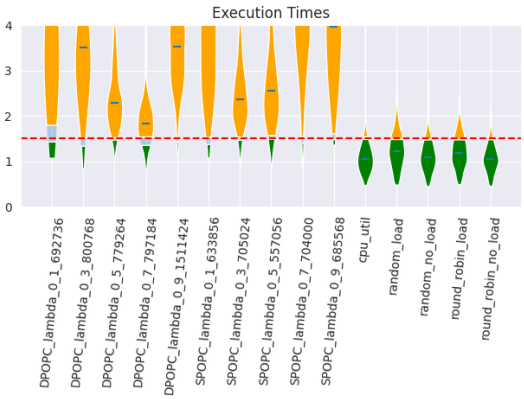
(b) Monitor Service κατανάλωση Ισχύος

0.7.7 Πείραμα με συγχρονισμένες συσκευές

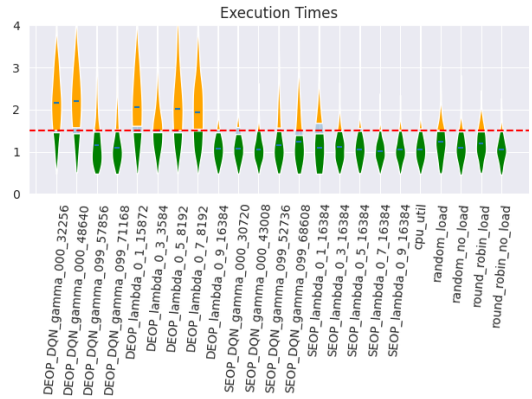
Στον πρώτο τύπο πειράματος επικεντρωνόμαστε στη συντονισμένη συμπεριφορά των συσκευών. Σε αυτό το σενάριο, ορισμένες συσκευές, δρώντας ως πελάτες, συγχρονίζονται για να ξεκινήσουν τις αιτήσεις τους περίπου την ίδια στιγμή ανά προκαθορισμένα χρονικά διαστήματα T (π.χ. 3 δευτερόλεπτα). Για παράδειγμα, μπορεί να έχουμε πολλές κάμερες που συγχρονίζονται για να ζητήσουν πληροφορίες εικόνων τις ίδες χρονικές στιγμές. Αυτό το πείραμα μας επιτρέπει να προσομοιώσουμε πραγματικές καταστάσεις όπου οι συσκευές συνεργάζονται ή συντονίζουν τις δραστηριότητές τους.

Αυτό το πείραμα προσομοιώνεται χρησιμοποιώντας τη Μη Ομογενή Διαδικασία Poisson, την ίδια διαδικασία που χρησιμοποιείται για την προσομοίωση του φορτίου κατά τη διάρκεια της εκπαίδευσης. Χρησιμοποιώντας τον αλγόριθμο 3, πραγματοποιούμε ένα πείραμα με $\lambda_{max} = 15$. Αυτός ο αλγόριθμος, σε κάθε χρονικό βήμα, καθορίζει τις συσκευές που αποστέλλουν ταυτόχρονες αιτήσεις. Το χρονικό διάστημα T ισούται με 3.

Για $\lambda_{max} = 15$, στα σχήματα 5.3.5a και 5.3.5b απεικονίζεται ο συνολικός χρόνος εκτέλεσης για layered και whole εκτελέσεις. Στα σχήματα 5.3.6a και 5.3.6b απεικονίζονται τα αποτελέσματα όσον αφορά την κατανάλωση ενέργειας. Επιπλέον, για καλύτερη κατανόηση των αποτελεσμάτων, στα σχήματα 5.3.7a και 5.3.8b παρέχουμε το ποσοστό των συσκευών που χρησιμοποιήθηκαν κατά τη διάρκεια κάθε πειράματος, και στα σχήματα 5.3.8a και 5.3.8b τα ποσοστά παραβίασης (το ποσοστό των αιτήσεων που υπερέβησαν το SLA). Τελικά, για την επίπεδη εκτέλεση, το σχήμα 5.3.9 δείχνει το ποσοστό χρόνου δικτύου και το ποσοστό χρόνου εκτέλεσης.

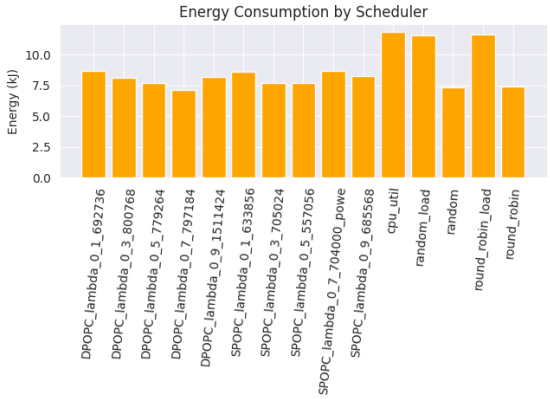


(a) Χρόνος αιτημάτων σε Layered εκτέλεση με $\lambda_{max} = 15$

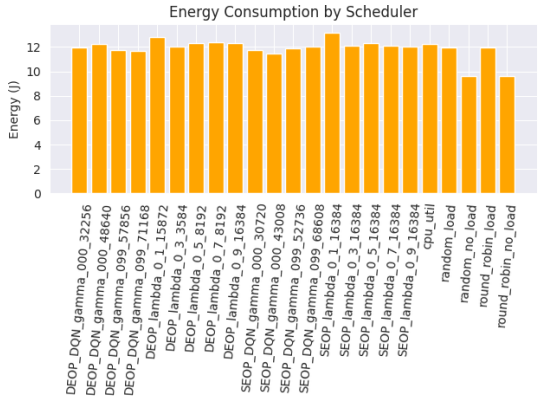


(b) Χρόνος αιτημάτων σε Whole εκτέλεση με $\lambda_{max} = 15$

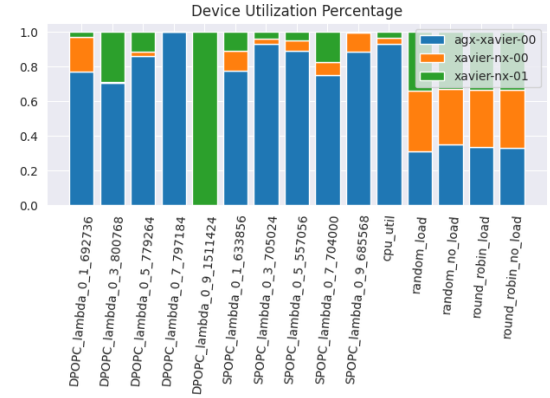
Layered Εκτέλεση: Οι αθροιστικοί χρόνοι εκτέλεσης και οι περιπτώσεις παραβιάσεων των SLAs είναι



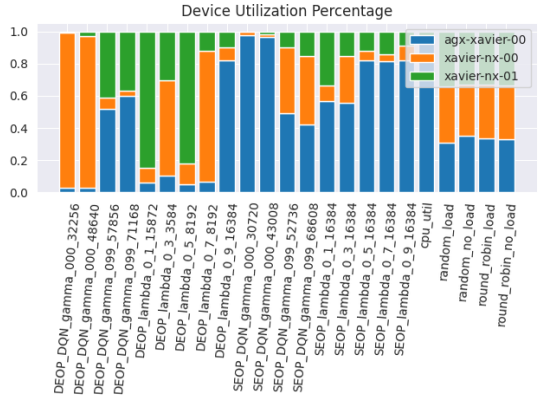
(a) Κατανάλωση ενέργειας σε Layered εκτέλεση με $\lambda_{max} = 15$



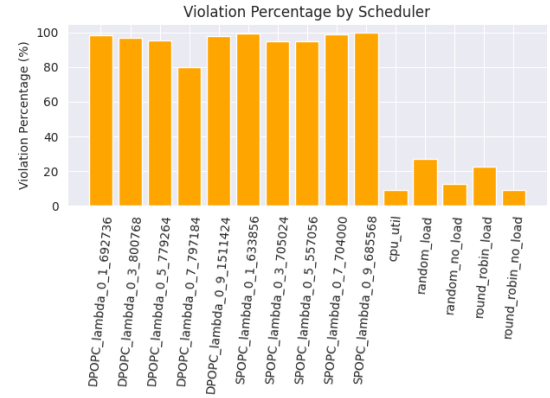
(b) Κατανάλωση ενέργειας σε Whole εκτέλεση με $\lambda_{max} = 15$



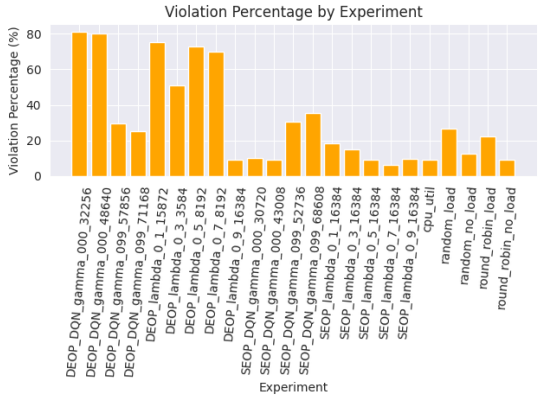
(a) Ποσοστό χρησιμοποίησης των συσκευών σε Layered εκτέλεση με $\lambda_{max} = 15$



(b) Ποσοστό χρησιμοποίησης των συσκευών σε Whole εκτέλεση με $\lambda_{max} = 15$



(a) Ποσοστό παράβασης σε Layered εκτέλεση με $\lambda_{max} = 15$



(b) Ποσοστό παράβασης σε Whole εκτέλεση με $\lambda_{max} = 15$

σημαντικά μεγαλύτεροι στο πλαίσιο της layered εκτέλεσης, σε σύγκριση με τη whole εκτέλεση με τους πιο απλούς schedulers (όπως απεικονίζεται στο σχήμα 5.3.5a και στο σχήμα 5.3.8a). Παρατηρώντας το σχήμα 5.3.9, γίνεται εμφανές ότι ένα σημαντικό ποσοστό της συνολικής δαπάνης χρόνου αφορά δραστηριότητες

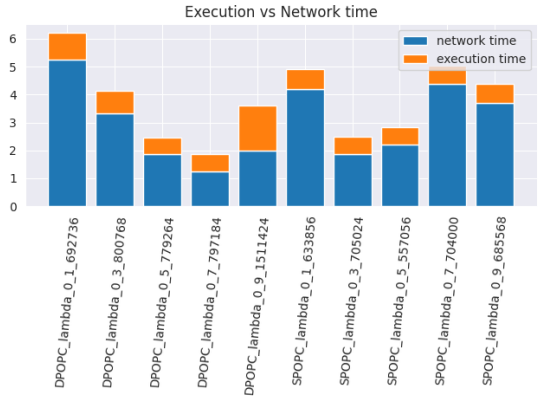


Figure 0.7.8: Χρόνος Εκτέλεσης vs Δικτύου σε Layered εκτέλεση

που σχετίζονται με το δίκτυο. Η ίδια η εκτέλεση αποτελεί ένα σχετικά μικρό μέρος του συνολικού χρόνου. Η αυξημένη ζήτηση σε πόρους δικτύου προέρχεται από τις πολυάριθμες αιτήσεις που δημιουργούνται. Αυτή η εκτεταμένη παραγωγή αιτημάτων έχει ως αποτέλεσμα τη συμφόρηση εντός του server. Ειδικότερα, ένα αυξημένο βάθος νευρωνικού δικτύου (περισσότερα στρώματα) αντιστοιχεί σε αυξημένο όγκο παραγόμενων αιτήσεων. Οι schedulers που παρουσιάζουν πιο ευνοϊκά χρονικά αποτελέσματα είναι εκείνοι που βασίζονται κυρίως σε μια μοναδική συσκευή για την επεξεργασία αιτημάτων. Αυτοί οι schedulers, με παράδειγμα το DPOPC_lambda_0_7_797184, καταγράφουν σημαντικά μειωμένες χρονικές επιβαρύνσεις που σχετίζονται με το δίκτυο, δεδομένης της απουσίας μεταφοράς δεδομένων μεταξύ συσκευών. Ωστόσο, είναι επιτακτική ανάγκη να αναγνωριστεί ότι οι εν λόγω schedulers οδηγούν αναπόφευκτα σε υπερχρήση της συσκευής. Για παράδειγμα, ο scheduler DPOPC_lambda_0_9_1511424, ο οποίος αναθέτει όλες τις εργασίες στο xavier-nx-01, έχει σημαντικά μεγαλύτερη συνολική διάρκεια εκτέλεσης. Αντίθετα, ο scheduler DPOPC_0_7_797184, ο οποίος στέλνει όλα τα αιτήματα στη συσκευή υψηλής απόδοσης agx-xavier-00, επιτυγχάνει σημαντικά μικρότερους συνολικούς χρόνους εκτέλεσης. Αξίζει να επισημανθεί η απόδοση των πιο απλών schedulers με πρόσθετες εξωτερικές διακυμάνσεις του φόρτου εργασίας (τρέχοντας απλά το monitor service σε κάθε συσκευή), που εκδηλώνεται ως αύξηση των ποσοστών παραβίασης κατά περίπου 16%. Όσον αφορά την κατανάλωση ενέργειας, οι schedulers παρουσιάζουν σταθερά χαμηλή ενεργειακή χρήση, κατά μέσο όρο περίπου 7,5 kilojoules. Ο schedulers DPOPC_lambda_0_7_797184 ξεχωρίζει με τη χαμηλότερη κατανάλωση ενέργειας, με 7 kilojoules, λόγω της αποκλειστικής εξάρτησής του από το agx-xavier-00 χωρίς μεταφορές δεδομένων μεταξύ συσκευών, όπως διευκρινίστηκε προηγουμένως. Οι άλλοι schedulers έχουν επίσης αξιοσημείωτες επιδόσεις όσον αφορά την ενεργειακή απόδοση. Ειδικότερα, οι απλοί schedulers και οι RL-based schedulers παρουσιάζουν συγκρίσιμα προφίλ ενεργειακής κατανάλωσης, αλλά στους απλούς schedulers η εισαγωγή ταυτόχρονων φορτίων εργασίας χαμηλής έντασης σε όλες τις συσκευές οδηγεί σε αξιοσημείωτη αύξηση της ενεργειακής κατανάλωσης, η οποία αυξάνεται από περίπου 7,5 kilojoules σε περίπου 12 kilojoules.

Whole Εκτέλεση: Οι συνολικοί χρόνοι εκτέλεσης είναι αρκετά παρόμοιοι, αλλά υπάρχουν ορισμένες εξαιρέσεις. Αυτές οι εξαιρέσεις περιλαμβάνουν τους schedulers DEOP_DQN_gamma_000_32256, DEOP_DQN_gamma_000_48640, DEOP_lambda_0_1_15872, DEOP_lambda_0_3_3584, DEOP_lambda_0_5_8192 και DEOP_lambda_0_9_8192. Αυτοί οι schedulers χρησιμοποιούν σε μεγάλο βαθμό συσκευές xavier-nx-0*, οι οποίες έχουν λιγότερες υπολογιστικές ικανότητες από την agx-xavier-00. Αυτή η υπερβολική χρήση οδηγεί σε μεγαλύτερους χρόνους εκτέλεσης, με ποσοστά παραβίασης που φτάνουν μέχρι και το 80%. Από την άλλη πλευρά, οι περισσότεροι άλλοι schedulers που βασίζονται σε RL χρησιμοποιούν κυρίως το agx-xavier-00, κατανέμοντας μόνο ένα μέρος των εργασιών στις συσκευές xavier-nx-0*. Αυτή η προσέγγιση οδηγεί σε συντομότερους χρόνους εκτέλεσης και ξεπερνά ακόμη και τους πιο απλούς schedulers. Συγκεκριμένα, ο schedulers round-robin με και χωρίς πρόσθετο φορτίο, έχει ποσοστά παραβίασης 22% και 9,5%, αντίστοιχα. Ο τυχαίος schedulers έχει ποσοστά παραβίασης 25% και 15%, ενώ ο schedulers που βασίζεται στη χρήση της CPU διατηρεί ποσοστό παραβίασης 10%. Όσον αφορά τους RL schedulers, πολλοί επιτυγχάνουν

καλές επιδόσεις με χαμηλά ποσοστά παραβίασης. Για παράδειγμα, οι DEOP_lambda_0_9_16384, SEOP_DQN_gamma_000_43008 και SEOP_lambda_0_5_16384 έχουν ποσοστά παραβίασης 9%. Αξίζει να σημειωθεί ότι το **SEOP_lambda_0_7_16384 επιτυγχάνει το χαμηλότερο ποσοστό παραβίασης με 5%**.Σύμφωνα με την κατανάλωση ενέργειας, όλοι οι schedulers επιτυγχάνουν παρόμοιες επιδόσεις με κατανάλωση γύρω στα 12 kilojoules. Η χαμηλότερη κατανάλωση ενέργειας επιτυγχάνεται από τους round-robin και random schedulers (5 kilojoules), χωρίς εξωτερικό φορτίο, ενώ οι ίδιοι με εξωτερικό φορτίο καταναλώνουν επίσης 12 kilojoules. Αυτό δείχνει ότι το monitor service είναι ο κύριος λόγος για την ενεργειακή διαφορά μεταξύ των schedulers χωρίς φορτίο και των άλλων schedulers.

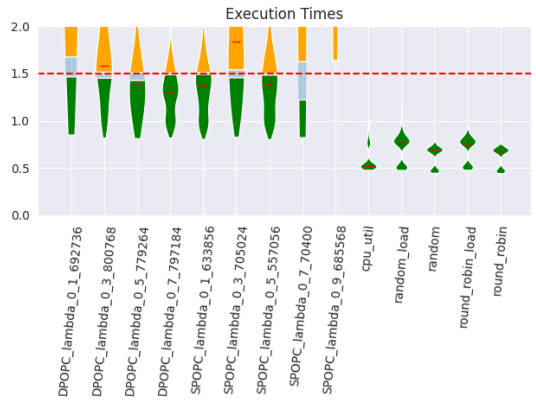
Αξίζει να αναφερθεί ότι αυτό το πείραμα αξιολόγησης αποτελεί πρόκληση για τους RL-based schedulers. Ο λόγος είναι ότι όταν ταυτόχρονα φτάνουν αιτήσεις στον server, ο οποίος φιλοξενεί τον scheduler, και ο scheduler επεξεργάζεται όλες αυτές τις αιτήσεις με σχεδόν ταυτόσημες καταστάσεις συσκευών ως είσοδο. Κατά συνέπεια, ο scheduler προγραμματίζει αυτές τις αιτήσεις χωρίς να λαμβάνει επαρκώς υπόψη τον πιθανό αντίκτυπό τους στις συσκευές. Αυτή η λειτουργική προσέγγιση μπορεί να οδηγήσει σε μη βέλτιστες και αναποτελεσματικές αποφάσεις, ιδίως στο πλαίσιο του συγκεκριμένου πειραματικού πλαισίου.

Πείραμα ανεξάρτητων πελατών

Σε αυτό το είδος πειράματος οι συσκευές στέλνουν ασύγχρονα αιτήματα στον server που φιλοξενεί τον scheduler. Τα πειράματα περιστρέφονται γύρω από δύο βασικές παραμέτρους: τον αριθμό των συμμετεχουσών συσκευών και τον ρυθμό με τον οποίο αυτές οι συσκευές στέλνουν αιτήματα. Μεταβάλλοντας αυτούς τους παράγοντες, αξιολογούμε τις επιδόσεις του συστήματος σε μια σειρά από συνθήκες φορτίου.

LAMBDA_MAX=25, NUM_CLIENTS=1:

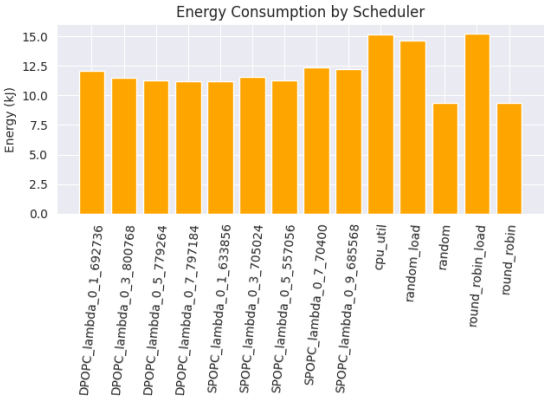
Σε αυτή την περίπτωση υπάρχει μόνο ένας πελάτης που ανά περιόδους στέλνει πολλά αιτήματα. Στα σχήματα 5.3.10a και 5.3.10b απεικονίζεται ο συνολικός χρόνος εκτέλεσης για whole και layered εκτελέσεις. Στα σχήματα 5.3.11a και 5.3.11b απεικονίζονται τα αποτελέσματα σύμφωνα με την κατανάλωση ενέργειας. Επιπλέον, για την καλύτερη κατανόηση των αποτελεσμάτων, στα σχήματα 5.3.12a και 5.3.11b παρέχουμε το ποσοστό των συσκευών που χρησιμοποιήθηκαν κατά τη διάρκεια κάθε πειράματος και στα σχήματα 5.3.13a και 5.3.13b τα ποσοστά παραβίασης (το ποσοστό των αιτήσεων που υπερέβησαν το SLA). Τέλος, για τη layered εκτέλεση, το σχήμα 5.3.14 δείχνει το ποσοστό του χρόνου δικτύου και το ποσοστό του χρόνου εκτέλεσης.



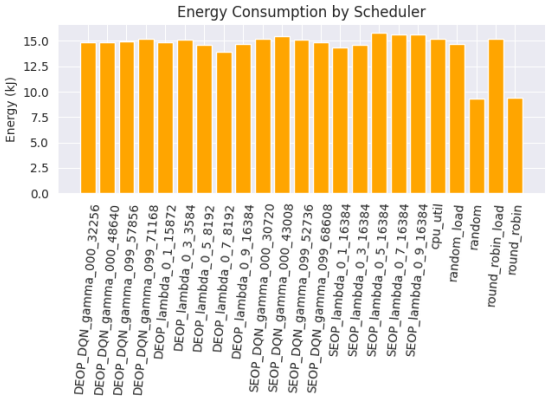
(a) Χρόνος αιτημάτων σε Layered εκτέλεση με $\lambda_{max} = 25$ και $num\_clients = 1$

(b) Χρόνος αιτημάτων σε Whole εκτέλεση με $\lambda_{max} = 25$ και $num\_clients = 1$

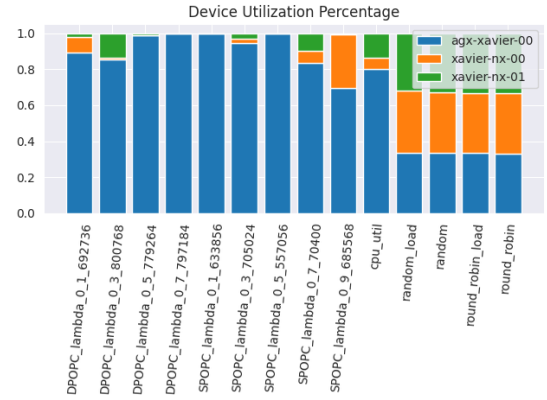
Layered εκτέλεση: Παρατηρώντας τις εικόνες 5.3.10a και 5.3.11a γίνεται φανερό ότι το σύστημα λειτουργεί υπό σχετικά μικρό φορτίο. Ειδικότερα, σε ορισμένες περιπτώσεις, η layered εκτέλεση επιδεικνύει σχετικά χαμηλό ποσοστό παραβιάσεων. Για παράδειγμα, ο scheduler DPOPC_lambda_0_7_791184



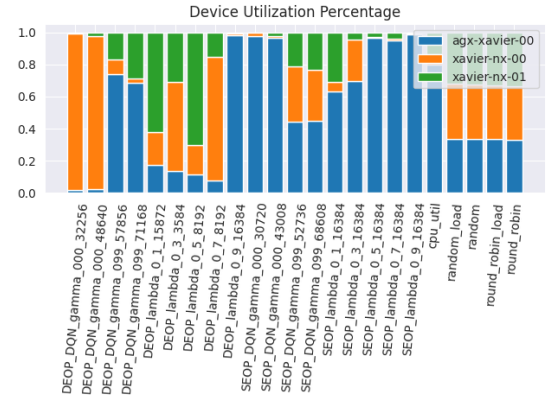
(a) Κατανάλωση ενέργειας σε Layered εκτέλεση με $\lambda_{max} = 25$ και $num\_clients = 1$



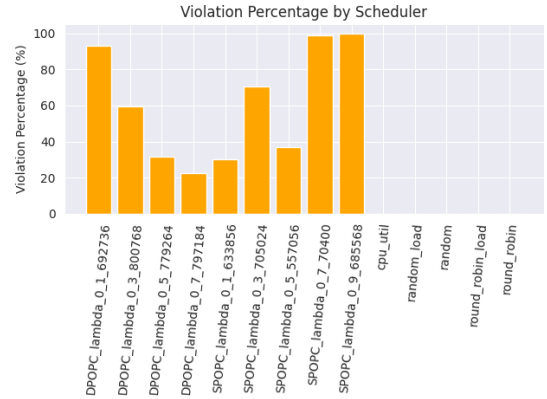
(b) Κατανάλωση ενέργειας σε Whole εκτέλεση με $\lambda_{max} = 25$ και $num\_clients = 1$



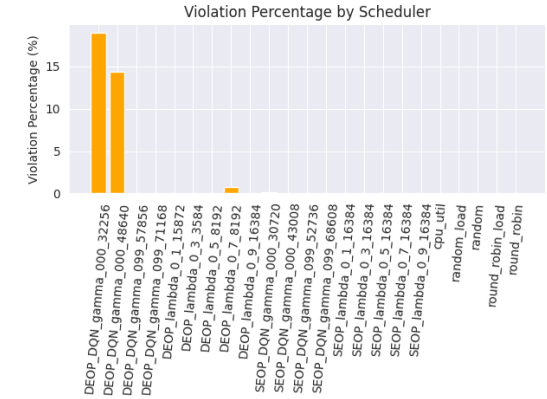
(a) Ποσοστό χρησιμοποίησης των συσκευών σε Layered εκτέλεση με $\lambda_{max} = 25$ και $num\_clients = 1$



(b) Ποσοστό χρησιμοποίησης των συσκευών σε Whole εκτέλεση με $\lambda_{max} = 25$ και $num\_clients = 1$



(a) Ποσοστό παράβασης σε Layered εκτέλεση με $\lambda_{max} = 25$ και $num\_clients = 1$



(b) Ποσοστό παράβασης σε Whole εκτέλεση με $\lambda_{max} = 25$ και $num\_clients = 1$

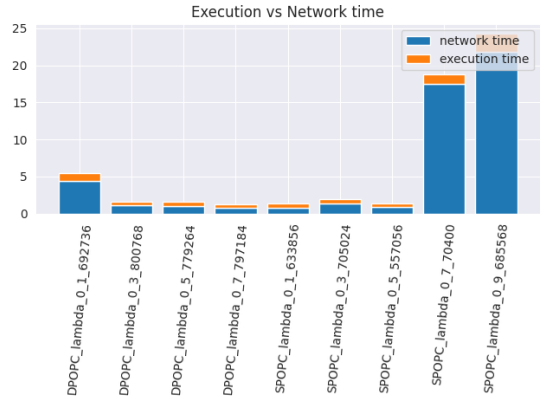


Figure 0.7.13: Χρόνος Εκτέλεσης vs Δικτύου σε Layered εκτέλεση $\lambda_{max} = 25$ και $num\_clients = 1$

και ο scheduler SPOPC_lambda_0_1_633856 παρουσιάζουν ποσοστά παραβίασης 23% και 30% αντίστοιχα, μια σημαντική βελτίωση από το προηγούμενο πείραμα, όπου αυτοί οι scheduler είχαν 80% 95% αντίστοιχα. Επιπλέον, το σχήμα 5.3.14 παρέχει πληροφορίες σχετικά με το χρόνο δικτύου, αποκαλύπτοντας ότι η πλειονότητα των scheduler παρουσιάζει σημαντικά μειωμένο χρόνο δικτύου σε σύγκριση με το προηγούμενο πείραμα. Αυτό μπορεί να αποδοθεί στην απουσία συμφόρησης του δικτύου στο διακομιστή, που οδηγεί σε αυξημένη αποδοτικότητα του συστήματος.

Ιδιαίτερο ενδιαφέρον παρουσιάζει η απόδοση των απλών schedulers, οι οποίοι, ελλείψει εξωτερικού φορτίου, επιτυγχάνουν μηδενικά ποσοστά παραβίασης. Ωστόσο, όταν υποβάλλονται σε εξωτερικό φορτίο, φαίνεται να καταναλώνουν περισσότερη ενέργεια

Whole Εκτέλεση: Όσον αφορά τη συνολική εκτέλεση, όπως φαίνεται στο Σχήμα 5.3.13b, σχεδόν όλοι οι schedulers επιτυγχάνουν ποσοστό παραβίασης 0%, γεγονός που υποδηλώνει τον αποτελεσματικό χειρισμό του σχετικά ελαφρού φόρτου εργασίας. Οι μόνες εξαιρέσεις σε αυτό το μοτίβο είναι οι schedulers DEOP_DQN_gamma_000_32256 και DEOP_DQN_gamma_000_48640, οι οποίοι δυσκολεύονται λόγω της δρομολόγησης όλων των αιτήσεων στη συσκευή xavier-nx-00.

Όσον αφορά την κατανάλωση ενέργειας, το σχήμα 5.3.11b δείχνει ότι τα αποτελέσματα παραμένουν παρόμοια με το προηγούμενο πείραμά μας. Τα περισσότερα μοντέλα παρουσιάζουν τις ίδιες επιδόσεις, εκτός από τους απλούς schedulers χωρίς φορτίο, γεγονός που υποδεικνύει για άλλη μια φορά ότι η υπηρεσία παρακολούθησης είναι υπεύθυνη για ένα σημαντικό ποσό κατανάλωσης ενέργειας.

Στα επόμενα πειράματα, έχουμε παραλείψει την παρουσίαση των αποτελεσμάτων της layered εκτέλεσης. Η παράλειψη αυτή οφείλεται στην ιδιαίτερα κακή επίδοση στους συνολικούς χρόνους εκτέλεσης, γεγονός που καθιστά την εμφάνισή τους περιττή.

LAMBDA_MAX=15, NUM_CLIENTS=5:

Σε αυτή την περίπτωση υπάρχουν πέντε πελάτες που ανά περιόδους στέλνουν πολλά αιτήματα. Ο ρυθμός που τα στέλνουν είναι μικρότερος από το προηγούμενο πείραμα ($lambda_{max} = 15$ αντί για 25).

Σε αυτό το συγκεκριμένο σενάριο, η απόδοση των περισσότερων schedulers παρουσιάζει ένα αξιέπαινο επίπεδο αποδοτικότητας. Ειδικότερα, οι απλοί schedulers, όταν δεν εφαρμόζεται κανένα εξωτερικό φορτίο, επιτυγχάνουν σχεδόν 0% ποσοστό παραβίασης, αποδεικνύοντας την αξιοπιστία τους στη διαχείριση των φόρτων εργασίας. Ο scheduler CPU_UTIL ξεχωρίζει επίσης, με εξαιρετικά αποτελεσματική απόδοση, επιτυγχάνοντας ποσοστό παραβίασης μικρότερο από 1%. Επιπλέον, ο scheduler SEOP_lambda_0_3_16384 παρέχει εξαιρετικές επιδόσεις, με σχεδόν μηδενικά ποσοστά παραβίασης. Αυτό είναι εμφανές από τα δεδομένα που παρουσιάζονται στο Σχήμα 5.3.15, όπου αυτός ο scheduler, μαζί με τους schedulers SEOP_lambda_0_1_16384 και CPU_UTIL, καταγράφει τους χαμηλότερους μέσους χρόνους εκτέλεσης. Από το Σχήμα 5.3.19, μπορούμε να παρατηρήσουμε ότι ο scheduler

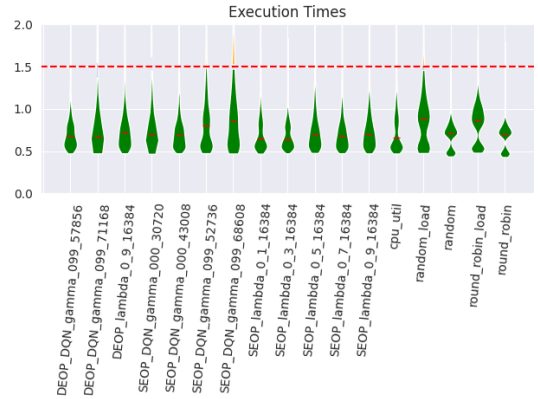


Figure 0.7.14: Χρόνος αιτημάτων σε Whole εκτέλεση με $\lambda_{max} = 15$ και $num\_clients = 5$

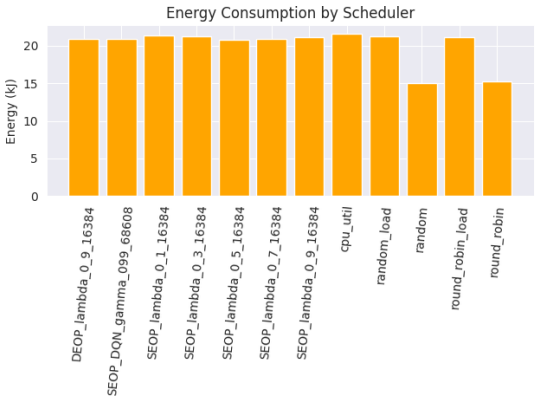


Figure 0.7.15: Κατανάλωση ενέργειας σε Whole εκτέλεση με $\lambda_{max} = 15$ και $num\_clients = 5$

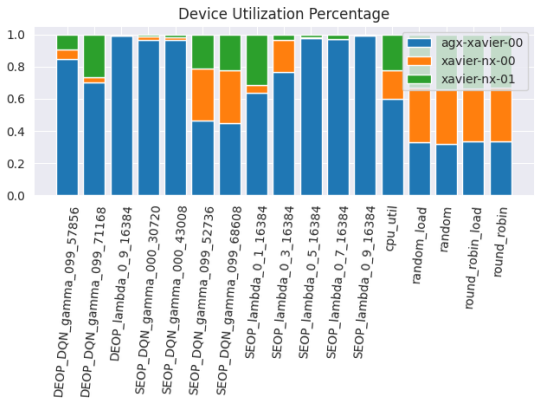


Figure 0.7.16: Ποσοστό χρησιμοποίησης των συσκευών σε Whole εκτέλεση με $\lambda_{max} = 15$ και $num\_clients = 5$

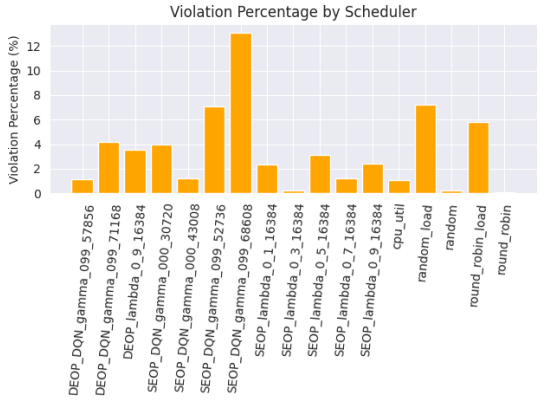


Figure 0.7.17: Ποσοστό παράβασης σε Whole εκτέλεση με $\lambda_{max} = 15$ και $num\_clients = 5$

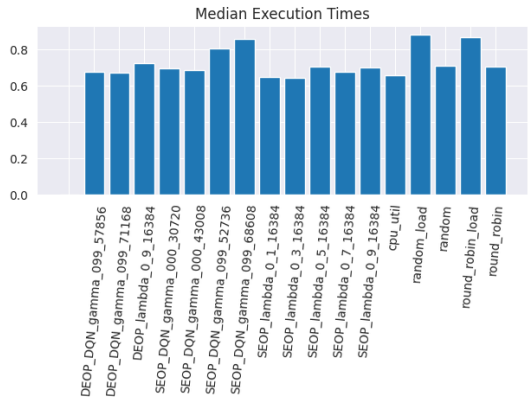


Figure 0.7.18: Μέσος χρόνος αιτημάτων σε Whole εκτέλεση με $\lambda_{max} = 15$ και $num\_clients = 5$

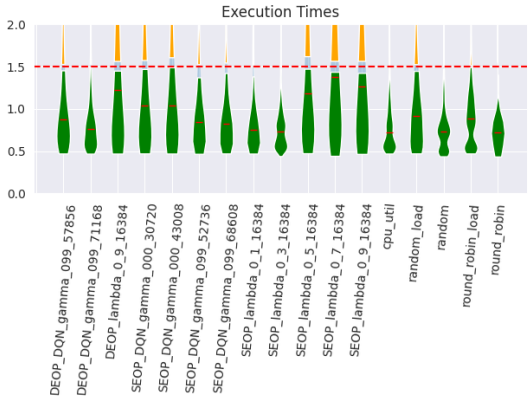


Figure 0.7.19: Χρόνος αιτημάτων σε Whole εκτέλεση με $\lambda_{max} = 10$ και $num\_clients = 8$

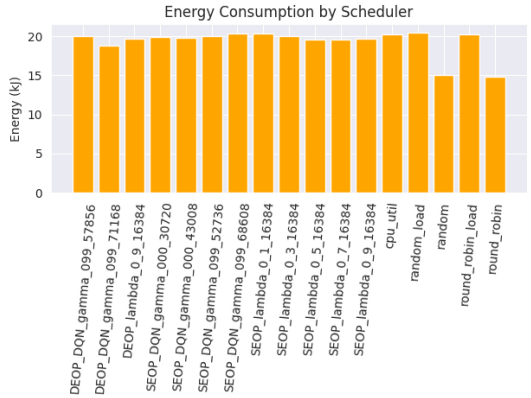


Figure 0.7.20: Κατανάλωση ενέργειας σε Whole εκτέλεση με $\lambda_{max} = 10$ και $num\_clients = 8$

SEOP_lambda_0_3_16384 αξιοποιεί την ισχυρή συσκευή agx-xavier-00, ενώ μεταφέρει με σύνεση ένα σημαντικό μέρος του φόρτου εργασίας στις συσκευές xavier-nx-0*. Αυτή η στρατηγική προσέγγιση έχει ως αποτέλεσμα ένα εντυπωσιακό ποσοστό παραβίασης σχεδόν 0% και τον χαμηλότερο μέσο χρόνο εκτέλεσης. Αντίθετα, άλλοι schedulers που επιβαρύνουν υπερβολικά τις συσκευές xavier-nx-0* ή βασίζονται υπερβολικά στη συσκευή agx-xavier-00 για την εκτέλεση εργασιών παρουσιάζουν σημαντικά χειρότερα ποσοστά παραβίασης. Όσον αφορά την κατανάλωση ενέργειας, τα αποτελέσματα παραμένουν συνεπή με εκείνα προηγούμενων πειραμάτων.

LAMBDA_MAX=10, NUM_CLIENTS=8:

Σε αυτή την περίπτωση υπάρχουν οκτώ πελάτες που ανά περιόδους στέλνουν πολλά αιτήματα.Ο ρυθμός που τα στέλνουν είναι λίγο μικρότερος από το προηγούμενο πείραμα ($\lambda_{max} = 10$ αντί για 15).

Παρατηρώντας το σχήμα 5.3.24, γίνεται εύκολα αντιληπτό ότι ο scheduler **SEOP_lambda_0_3_16384 επιδεικνύει εξαιρετική απόδοση, διατηρώντας ποσοστό παραβίασης μικρότερο από 1%**. Αυτή η απόδοση αποδίδεται στη δυναμική κατανομή του 75% των φόρτων εργασίας στη συσκευή agx-xavier-00 και του υπόλοιπου 25% στις συσκευές xavier-nx-0*, όπως φαίνεται στο σχήμα 5.3.22, όταν το agx-xavier-00 υπόκειται σε μεγάλα φορτία. Αντίθετα, οι απλοί schedulers, όταν λειτουργούν χωρίς εξωτερικό φορτίο, παρουσιάζουν σημαντικά μικρότερη ευρωστία, αποδίδοντας ποσοστά παραβίασης που έχουν αυξηθεί από σχεδόν 0% στο προηγούμενο πείραμα σε 3-4%. Επιπλέον, ο scheduler SEOP_lambda_0_3_16384 υπερτερεί έναντι των απλών schedulers με εξωτερικό φορτίο (20% και 6,5% ποσοστά παραβίασης του random scheduler και

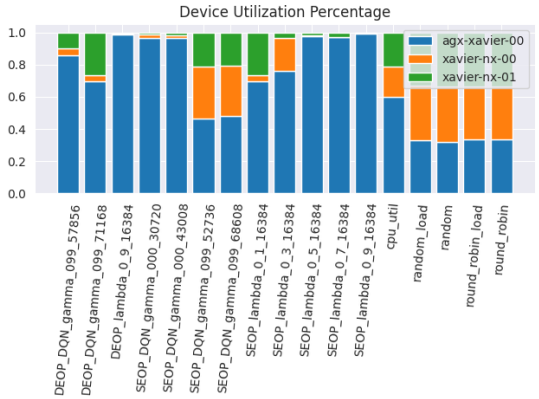


Figure 0.7.21: Ποσοστό χρησιμοποίησης των συσκευών σε Whole εκτέλεση με $\lambda_{max} = 10$ και $num\_clients = 8$

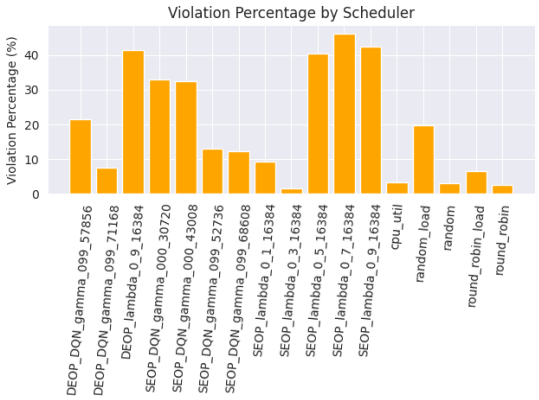


Figure 0.7.22: Ποσοστό παράβασης σε Whole εκτέλεση με $\lambda_{max} = 10$ και $num\_clients = 8$

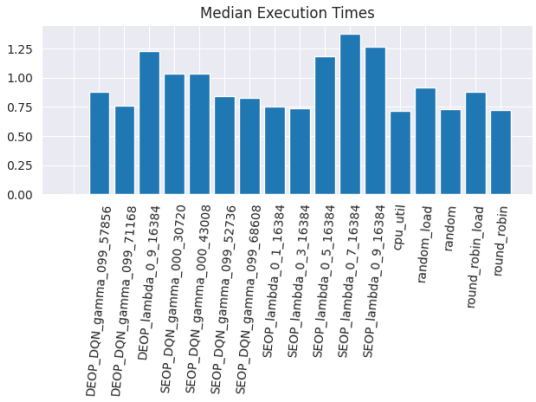


Figure 0.7.23: Μέσος χρόνος αιτημάτων σε Whole εκτέλεση με $\lambda_{max} = 10$ και $num\_clients = 8$

του round robin scheduler αντίστοιχα). Άλλοι schedulers που βασίζονται στην ενισχυτική μάθηση (RL), όπως ο DEOP_DQN_gamma_099_71168, παρέχουν επίσης αξιοσημείωτες επιδόσεις. Αυτό επιτυγχάνεται με τη συνετή μεταφόρτωση ενός μέρους των αιτήσεων στις συσκευές xavier-nx-0*, αποτρέποντας έτσι την υπερχρήση του agx-xavier-00 και οδηγώντας σε χαμηλότερα ποσοστά παραβίασης. Αντίθετα, οι schedulers που βασίζονται δυσανάλογα στο agx-xavier-00 αποτυγχάνουν να χειριστούν τις αιτήσεις. Για παράδειγμα, οι DEOP_lambda_0_9_16384, SEOP_lambda_0_5_16384, SEOP_lambda_0_7_16384 και SEOP_lambda_0_9_16384 παρουσιάζουν ποσοστά παραβίασης 40%. Αξίζει να σημειωθεί ότι ο scheduler cpu_util διατηρεί σταθερά επιδόσεις παρόμοιες με αυτές του προηγούμενου πειράματος, υπογραμμίζοντας την ανθεκτικότητά του. Τέλος, όπως απεικονίζεται στο Σχήμα 5.3.24, παρατηρούμε ότι οι SEOP_lambda_0_3_16384, DEOP_DQN_gamma_099_71168, cpu_util και οι υπόλοιποι απλοί schedulers χωρίς εξωτερικούς φόρτους εργασίας επιτυγχάνουν τους χαμηλότερους μέσους χρόνους εκτέλεσης. Αξίζει να σημειωθεί ότι τα αποτελέσματα της κατανάλωσης ενέργειας παραμένουν συνεπή με εκείνα του προηγούμενου πειράματος.

0.8 Συμπεράσματα

Στην παρούσα διπλωματική εργασία, δημιουργήθηκε ένα ολοκληρωμένο πλαίσιο (framework) για να καταστεί δυνατή η εκτέλεση βαθιών νευρωνικών δικτύων σε ένα Serverless Edge Cluster. Επιπλέον, έχει αναπτυχθεί ένας scheduler, ο οποίος χρησιμοποιεί τεχνικές Ενισχυτικής Μάθησης για την έξυπνη διαχείριση του scheduling της εκτέλεσης των DNN για τα εισερχόμενα αιτήματα στις Edge Devices. Ο πρωταρχικός στόχος αυτής της ερευνητικής προσπάθειας είναι να επιτευχθεί η βέλτιστη κατανομή των πόρων εντός των Edge Devices, τηρώντας παράλληλα σταθερά τις Συμφωνίες Επιπέδου Υπηρεσιών (SLAs).

0.8.1 Συζήτηση

Στο κεφάλαιο 0.4.1, αναπτύσσουμε ένα robust πλαίσιο (framework) για την εκτέλεση βαθιών νευρωνικών δικτύων. Αυτό το πλαίσιο αξιοποιεί το Kubernetes και το Knative για την ανάπτυξη, αξιοποιώντας τα πλεονεκτήματα και των δύο τεχνολογιών. Το Kubernetes, γνωστό για την ευρωστία του, αποτελεί το θεμέλιο του πλαισίου μας, παρέχοντας την αξιοπιστία που απαιτείται για την αποτελεσματική εκτέλεση DNN. Το Knative προσφέρει μια σειρά από πολύτιμα εργαλεία, συμπεριλαμβανομένου του Knative Pod Autoscaler, το οποίο προσαρμόζει δυναμικά τον αριθμό των ενεργών pods σε απόκριση των μεταβαλλόμενων φόρτων εργασίας για βέλτιστη αξιοποίηση των πόρων.

Επιπλέον, το πλαίσιό μας προσφέρει την ευελιξία να υποστηρίζει τόσο την layered όσο και την whole εκτέλεση βαθιών νευρωνικών δικτύων. Αυτή η προσαρμοστικότητα δίνει τη δυνατότητα στους χρήστες να επιλέξουν τον καταλληλότερο τρόπο εκτέλεσης, είτε πρόκειται για μια προσέγγιση σε επίπεδο για πιο λεπτομερή έλεγχο είτε για ολόκληρη εκτέλεση για αποδοτικότητα. Η ευελιξία του πλαισίου μας το εξοπλίζει για την αντιμετώπιση ποικίλων περιπτώσεων χρήσης και το ευθυγραμμίζει με τις εξελισσόμενες απαιτήσεις των edge περιβαλλόντων.

Στο κεφάλαιο 0.5, εμβαθύνουμε στην ανάπτυξη ενός scheduler Ενισχυτικής Μάθησης που έχει σχεδιαστεί για να διαχειρίζεται αποτελεσματικά την κατανομή των εισερχόμενων αιτήσεων στις συσκευές. Οι πρωταρχικοί στόχοι αυτού του scheduler είναι δύο: να διασφαλίσει ότι δεν παραβιάζονται οι Συμφωνίες Επιπέδου Υπηρεσιών (SLAs) και να ελαχιστοποιήσει την κατανάλωση ενέργειας μέσα στο σύστημα.

Για την αντιμετώπιση ενός φάσματος σεναρίων, έχουμε σχεδιάσει schedulers τόσο για layered όσο και για whole εκτέλεση βαθιών νευρωνικών δικτύων. Αυτή η προσαρμοστικότητα είναι καίριας σημασίας, καθώς μας επιτρέπει να προσαρμόζουμε την προσέγγιση του scheduling στις συγκεκριμένες ανάγκες διαφορετικών περιπτώσεων χρήσης, παρέχοντάς μας τη δυνατότητα βελτιστοποίησης της κατανομής πόρων και της κατανάλωσης ενέργειας σε ένα ευρύ φάσμα περιβαλλόντων υπολογισμού ακραίων περιοχών.

0.8.2 Μελλοντικές Επεκτάσεις

Στο μέλλον ενδέχεται να προταθούν διάφορες επεκτάσεις και παραλλαγές της εργασίας αυτής.

- Η layered εκτέλεση πάσχει από το χρόνο δικτύου και η whole εκτέλεση δεν δίνει αρκετό έλεγχο στο που θα εκτελεστούν τα layers. Για να επιτύχουμε μια ισορροπία μεταξύ αυτών των δύο μεθόδων, προτείνουμε μια υβριδική προσέγγιση. Η προσέγγιση αυτή περιλαμβάνει τη στρατηγική τοποθέτηση καθορισμένων σημείων εξόδου μέσα σε κάθε βαθύ νευρωνικό δίκτυο, επιτρέποντας αποφάσεις scheduling μόνο σε αυτά τα προκαθορισμένα σημεία. Με την εφαρμογή αυτής της στρατηγικής, ελαχιστοποιούμε αποτελεσματικά τη σωρευτική καθυστέρηση του δικτύου, καθώς λιγότερα σημεία εξόδου οδηγούν σε μειωμένη επιβάρυνση του δικτύου. Ταυτόχρονα, διατηρούμε ένα λογικό επίπεδο ελέγχου, διασφαλίζοντας ότι η υπολογιστική διαδικασία παραμένει προσαρμόσιμη στις ειδικές ανάγκες της εκάστοτε εργασίας.
- Μία άλλη μελλοντική επέκταση είναι η δημιουργία μιας κατανεμημένης έκδοσης του RL scheduler. Το υπάρχον μοντέλο κεντρικού διακομιστή αποδεικνύεται λιγότερο κλιμακούμενο στο πλαίσιο μιας συστάδας Edge Cluster πολλαπλών συσκευών. Για την αντιμετώπιση αυτής της πρόκλησης κλιμάκωσης, μια κατανεμημένη εκδοχή του μηχανισμού scheduling μας προσφέρει μια πολλά υποσχόμενη λύση. Αυτή η κατανεμημένη προσέγγιση συνεπάγεται την ανάπτυξη μεμονωμένων πρακτόρων RL σε κάθε συσκευή, οι οποίοι συμμετέχουν συλλογικά σε μια μορφή δημοπρασίας για να καθορίσουν τη συσκευή που είναι υπεύθυνη για την εκτέλεση μιας δεδομένης αίτησης. Αυτό μπορεί να βελτιώσει σημαντικά την επεκτασιμότητα και την προσαρμοστικότητα του συστήματος scheduling σε πολύπλοκα και δυναμικά περιβάλλοντα Edge Cluster.

Chapter 1

Introduction

In today's digital era, the confluence of two major trends has dramatically transformed the landscape of modern computing. On one hand, the explosive growth of data has ushered in the era of big data, revolutionizing the way we capture, process, and harness information. On the other hand, artificial intelligence (AI), and more specifically, neural networks, have risen to prominence as the driving force behind a multitude of applications, ranging from image recognition to natural language processing. As these versatile AI models take center stage, their efficient execution becomes increasingly critical in shaping the future of computing.

Traditionally, neural networks have found their home in Cloud computing environments, renowned for their extensive computational resources housed within sprawling data centers. These environments offer immense computational power, but they also introduce challenges related to latency and network availability. Such challenges can be particularly limiting for applications that hinge on real-time responsiveness, where every moment counts.

In response to these challenges, our work explores a shift in the deployment of neural networks, moving beyond the traditional confines of the Cloud to embrace the paradigm of Edge computing. Edge computing represents an alternative approach, seeking to overcome the limitations of traditional cloud environments by bringing computation closer to data sources. This proximity enables real-time data processing and decision-making, effectively reducing the execution latency of neural networks and enhancing the responsiveness of applications.

However, the transition to the edge environment is not without its own set of complexities. Devices operating at the edge exhibit a wide spectrum of computational capacity, ranging from high-performance servers to resource-constrained Internet of Things (IoT) devices. Effectively managing this heterogeneity and efficiently allocating resources to ensure optimal neural network execution is a multifaceted challenge.

1.1 Contributions

In response to these challenges, We have built a robust and reliable framework for DNN execution on the edge, harnessing the power of Serverless computing. Serverless computing, known for its capacity to abstract the intricacies of infrastructure management, streamline resource scaling, alleviate operational burdens, and optimize resource allocation, emerges as a pivotal solution. This approach seamlessly aligns with the dynamic and heterogeneous nature of edge environments, making it a fundamental enabler in our pursuit to address the challenges and optimize the deployment of neural networks.

The deployment of neural networks, while essential, is just the implementation part. Efficiency hinges on the presence of a sophisticated and effective scheduling mechanism. It's the dynamic art of scheduling that ensures each task seamlessly aligns with the right resource at precisely the most opportune

moment. The intricacies of this scheduling process are central to enhancing the overall efficiency and responsiveness of our system. Here, we introduce a Reinforcement Learning (RL) based scheduling algorithm. This RL algorithm plays a central role in intelligently orchestrating the scheduling process, striving to meet defined Service Level Agreements (SLAs) and response time targets. Moreover, it seeks to optimize energy consumption by judiciously allocating tasks to energy-efficient devices, whenever and wherever feasible.

1.2 Thesis Structure

The thesis is divided into 5 chapters. Chapter 2 delivers a rich and comprehensive exploration of the related work, contributing significantly to the body of knowledge in our research area. Chapter offers a detailed look into the background work and the technologies used in our study, helping us better understand the basis of our research. Chapter 4 provides an in-depth explanation of the implementation for both the deployment framework and the scheduling algorithm, offering an in-depth exploration of our workings and functionalities. Chapter 5 delivers an extensive evaluation of our approach, offering a deep analysis and comprehensive insights into its performance and effectiveness. Finally, Chapter 6 summarizes the thesis's findings and presents our future directions.

Chapter 2

Related Work

This chapter provides an overview of prior research efforts that inform the context of our study. These works cover various aspects of edge computing, including edge-based deep learning, collaborative DNN partitioning, edge swarm coordination, and serverless computing at the edge. By examining these contributions, we gain insights into the evolving field of edge computing, setting the stage for our own research.

Hyuk-Jin et al. [1] propose a partitioning-based Deep Neural Network (DNN) offloading technique for edge computing. The proposed technique, IONN, partitions the DNN layers and incrementally uploads the partitions to allow collaborative execution by the client and the edge server even before the entire DNN model is uploaded. The experimental results demonstrate the effectiveness of IONN in improving both query performance and energy consumption during DNN model uploading, compared to a simple all-at-once approach. The proposed technique has the potential to enable DNN-centric edge servers where a mobile client can offload its custom DNN computations without a long waiting time.

Laskaridis et al. [2] propose a novel distributed inference system that aims to address the limitations of existing approaches by employing a progressive inference method. The proposed system, SPINN introduces a scheme of distributing progressive early-exit models across device and server, in which one exit is always present on-device, guaranteeing the availability of a result at all times. Moreover, SPINN casts the acceptable prediction confidence as a tunable parameter to adapt its accuracy-speed trade-off. Alongside, the system proposes a novel run-time scheduler that jointly tunes the split point and early-exit policy of the progressive model, yielding a deployment tailored to the application performance requirements under dynamic conditions. A comprehensive evaluation of the system's performance shows that SPINN outperforms its state-of-the-art collaborative inference counterparts by up to 2x in achieved throughput under varying network conditions, reduces the server cost by up to 6.8x, and improves accuracy by 20.7% under latency constraints, while providing robust operation under uncertain connectivity conditions and significant energy savings compared to cloud-centric execution.

Xueyu Hou et al. [3] propose Dystri, an innovative framework devised to facilitate dynamic inference on distributed edge infrastructure, thereby accommodating multiple heterogeneous users. The architecture comprises distributed controllers and a global coordinator, allowing per-request, per-user adjustments of quality-of-service, ensuring instantaneous, flexible, and discrete control. The framework is evaluated on three multi-user, heterogeneous DNN inference service platforms deployed on distributed edge infrastructure, encompassing seven DNN applications. The results show that Dystri achieves near-zero deadline misses and excels in adapting to varying user numbers and request intensities. Furthermore, Dystri outperforms baselines with accuracy improvement up to 95%.

Yanan Yang et al. [4] propose INFless. INFless provides a unified, heterogeneous resource abstraction between CPU and accelerators, which enables efficient resource allocation based on the workload

demands. The system achieves high throughput using built-in batching and non-uniform scaling mechanisms, while also supporting low latency through coordinated management of batch queuing time, execution time, and cold start rate. The architecture of INFless is designed to address the challenges of managing hybrid CPU/accelerator systems, selecting appropriate batch sizes and resources, and minimizing running overhead. A detailed overview of the INFless architecture includes the gateway, scheduler, prediction model, operator profiles, dispatcher, batching workload, CPU/GPU nodes, cold start manager, auto-scaling engine, and more.

Lockhar et al. [5] propose Scission, a tool for automated benchmarking of deep neural networks (DNNs) on a given set of target device, edge, and cloud resources for determining the optimal partition for maximizing DNN performance. It is underpinned by a benchmarking approach that determines the combination of potential target hardware resources and the sequence of layers that should be distributed for maximizing distributed DNN performance while accounting for user-defined objectives. Scission relies on empirical data and does not estimate performance by making assumptions of the target hardware or the DNN layers. Experimental studies were carried out on 18 different DNNs to demonstrate that Scission is a valuable tool for obtaining context-aware and performance-efficient distributed DNNs. Scission can also make decisions that cannot be manually made by a human due to the complexity and number of dimensions affecting the search space.

Kakolyris et al. [6] propose RoaD-RuNNer, a collaborative framework for Deep Neural Network (DNN) partitioning and offloading on heterogeneous edge systems. This innovative approach addresses the resource management challenges of DNNs deployed on edge computing systems and offers a promising solution for efficient DNN partitioning and offloading. The authors propose a dynamic offloading mechanism that uses collaborative filtering to predict the execution time and energy consumption of individual layers over different CPU/GPU architectures. The framework also includes a dynamic partitioning mechanism that efficiently splits and offloads DNN layers. The authors conducted an extensive experimental evaluation of their proposed framework, comparing it with baseline algorithms and state-of-the-art DNN offloading approaches over real hardware and networking. The results show that RoaD-RuNNer outperforms existing approaches by achieving up to 9.58× speedup on average and up to 88.73% less energy consumption on average. This work contributes to the field of edge computing and DNN offloading by providing a collaborative solution for efficient resource management.

Patterson et al. [7] propose a hardware-software system stack, HiveMind, that enables programmable execution of complex task workflows between cloud and edge resources in a performant and scalable manner. This architecture addresses the challenges of partitioning work manually and changing where computation runs, which can affect the software infrastructure needed. The program synthesis tool automatically explores task placement strategies, simplifying programmability of cloud-edge applications. The reconfigurable hardware acceleration fabric for network and remote memory accesses contributes to the platform's performance and scalability. Overall, this architecture presents a comprehensive solution to the challenges of autonomous devices, making it a valuable contribution to the field.

Bin Wang et al. [8] propose the LaSS platform, a novel architecture to address the challenges of serverless computing at the edge. The platform uses model-driven approaches to accurately predict the resources needed for serverless functions in the presence of highly dynamic workloads. LaSS uses principled queuing-based methods to determine an appropriate allocation for each hosted function and auto-scales the allocated resources in response to workload dynamics. The platform also utilizes resource reclamation methods based on container deflation and termination to reassign resources from over-provisioned functions to under-provisioned ones. The authors implement a prototype of their approach on OpenWhisk [9] and conduct a detailed experimental evaluation, demonstrating the ability of LaSS to meet service level objectives and operate with fair-share allocation guarantees in overload scenarios. The LaSS platform provides valuable insights into the design and implementation of serverless computing platforms for edge environments.

Chapter 3

Background on Deep Learning, Serverless and Edge computing

This chapter provides a comprehensive exploration of the theoretical underpinnings of this study, including (deep) neural networks, their fundamental architectures, serverless computing, edge computing, and their underlying technologies.

3.1 Artificial Neural Networks - Deep Learning

Neural networks are a class of machine learning algorithms inspired by the structure and function of biological neural networks in the human brain. They are composed of interconnected computational units, often referred to as "neurons," which work collaboratively to process data, extract patterns, and make predictions. Neural networks possess the remarkable ability to learn from examples, adjust their internal parameters, and generalize their knowledge to new, unseen data.

The concept of neural networks dates back to the 1940s. The term "neural network" itself was introduced in the works of Walter Pitts and Warren McCulloch [51]. The first was a logician and the second a neurophysiologist and their research progressed within the confines of the University of Chicago. This paper laid the foundation for the mathematical representation of artificial neurons, providing a theoretical framework for simulating computation in a manner inspired by the human brain.

However, the full potential of neural networks was not fully realized until much later due to limitations in computational resources and data availability. The resurgence of interest in neural networks began in the 1980s and gained momentum in the 2000s, driven by several factors. Advances in computational power, coupled with the availability of large datasets, allowed researchers to explore more complex neural network architectures.

The emergence of deep learning, characterized by the use of neural networks with multiple layers (deep neural networks), revolutionized the field. Techniques such as convolutional neural networks (CNNs) for image analysis and recurrent neural networks (RNNs) for sequential data transformed areas like computer vision, natural language processing, and speech recognition.

In recent years, the deep learning revolution has been fueled by advancements in hardware, including Graphics Processing Units (GPUs) optimized for parallel processing, and specialized hardware like Tensor Processing Units (TPUs). These technologies have accelerated the training and deployment of large-scale neural networks, enabling them to achieve impressive results in tasks that were previously considered challenging.

3.1.1 Neuron: The Core of Neural Networks

In artificial neural networks, a network consists of a collection of neurons. These neurons serve as the building block of the network, interconnected through synapses, allowing them to communicate and collaborate. The level at which neurons interact varies, driven by the synaptic weights assigned to their connections. These synaptic weights are not fixed; they dynamically adapt as the neural network receives input from the environment and learns from the data. This dynamic adaptation leads to the strengthening or weakening of connections, akin to enhancing or diminishing the significance of certain features. Essentially, synaptic weights encode empirical knowledge and equip the network with the capability to learn and adapt to the environment.

Figure 3.1.1 shows the structure of a neuron:

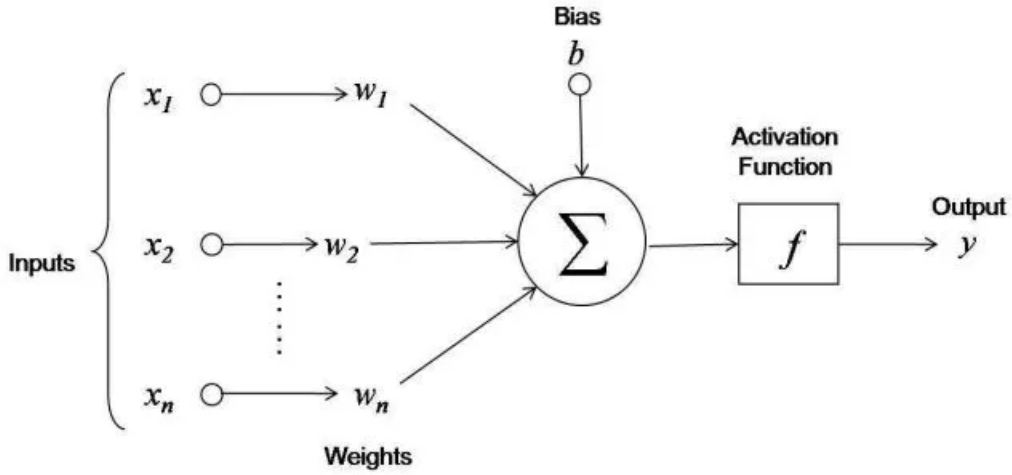


Figure 3.1.1: The structure of an artificial neuron [52]

The core elements are:

1. A collection of synapses, distinguished by individual weights. The input signal x_j at each artificial synapse j linked to artificial neuron k experiences multiplication by the synaptic weight w_kj.
2. A summation unit (adder) for adding input signals, each multiplied by the weight of the relevant connection.
3. A constant term b added to the weighted input sum before activation, to improve the models performance, known as bias.
4. A mathematical transformation applied to the weighted sum of inputs in a neuron, determining its output, known as activation function. They introduce non-linearity, enabling neural networks to capture intricate patterns and relationships within data.

The core elements can be described in a mathematical manner as follows:

$$u = W * x + b \tag{3.1.1}$$

$$where: \tag{3.1.2}$$

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \quad \text{and} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

The W is the vector containing the weights, the x is the input vector and u is the sum of bias and the adders output $(W * x)$.

Finally, the output of the neuron is:

$$y = \phi(u) \tag{3.1.3}$$

Where $\phi(.)$ is the activation function.

According to the activation functions, the most popular are:

1. Step Function
   We have

   $$\phi(u) = \begin{cases} 1, & \text{if } x \geq 1 \\ 0, & \text{otherwise} \end{cases} \tag{3.1.4}$$

   So, the output of the neuron, with the above activation function, is of the form:

   $$y = \begin{cases} 1, & \text{if } u \geq 1 \\ 0, & \text{otherwise} \end{cases} \tag{3.1.5}$$

   Where u is the output of the adder. It is not frequently utilized because is non-differentiable in $x = 0$ and the derivative is 0 elsewhere. This leads to challenges during the training process.

2. Sigmoid Function
   We have

   $$\phi(u) = \frac{1}{(1 + \exp^{-\ u})} \tag{3.1.6}$$

   It's one of the most popular activation functions in artificial neural networks. There are certain problems with sigmoid function like inferior performance and vanishing gradient [53].

3. tanh Function
   We have

   $$\phi(u) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.1.7}$$

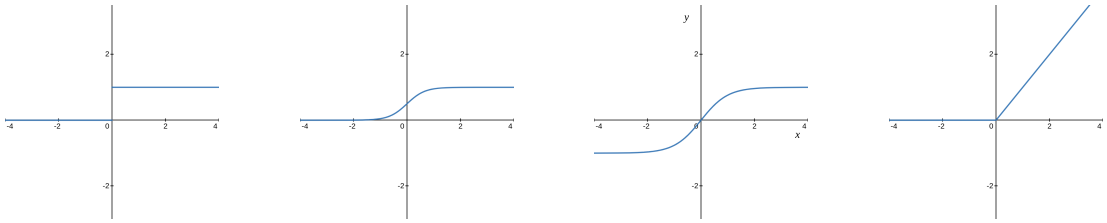
   Like sigmoid function, its one of the most popular activation function in artifical neural networks, but struggles too with inferior performance and vanishing gradient.

4. Rectified Linear Unit (ReLU)
   We have

   $$\phi(u) = \begin{cases} u, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.1.8}$$

   The most common activation function in the field of deep learning, overcoming the tanh and sigmoid.

Figure 3.1.2 illustrates the differences of the stated functions:



(a) Step function (b) Sigmoid function (c) tanh function (d) ReLU function

Figure 3.1.2: Most popular activation functions

3.1.2 Architectures of Neural Networks

Neural Network architectures are the fundamental structures that define the layout, connections and organization of artificial neural networks. These architectures play a crucial role in determining the network's ability to learn, generalize and perform specific tasks. Here are some common types of neural network architectures:

1. *Feedforward Neural Networks (FNNs)*: FNNs are the simplest form of neural networks. They are widely employed in machine learning tasks that require pattern recognition, classification, and feature extraction from static data, finding utility in image recognition, sentiment analysis, and medical diagnosis.
2. *Recurrent Neural Networks (RNNs)*: RNNs find extensive application in various domains, leveraging their ability to model sequences and time-dependent patterns, making them suitable for tasks such as natural language processing, speech synthesis, and predicting financial trends in time series data.
3. *Convolutional Neural Networks (CNNs)*: CNNs have gained renown for their proficiency in processing structured grid data and particularly images. These networks find widespread application in tasks such as image classification, object detection, and image segmentation, elevating their significance in the realm of computer vision and pattern recognition.
4. *Generative Adversarial Networks (GANs)*: GANs have emerged as a significant innovation due to their ability to synthesize realistic data samples. Their utility extends across domains like image generation, style translation, and anomaly detection, revolutionizing diverse applications in the realm of artificial intelligence.
5. *Transformers*: Transformers is a relatively new and pivotal architectural innovation, have gained significant acknowledgement for their adeptness in handling sequential data, enabling contextual comprehension and capturing intricate relationships. Their impact spans applications such as natural language processing, machine translation, and image generation, reshaping the landscape of contemporary artificial intelligence.

3.1.2.1 Feedforward Neural Networks

Feedforward neural network (FNNs), also known as Multi-Layer Perceptron (MLP), a fundamental architecture in deep learning, consist of layers of interconnected neurons (also referred to as perceptrons), described in 3.1.1, that process information in a one-way flow, from input to output, as shown in 3.1.3. The architecture lacks cycles or loops, distinguishing it as "feedforward". All neurons in one layer transmit their outputs as inputs to all neurons of the subsequent layer without feedback connections. Input data propagates through the network, undergoing transformations and computations through

weighted connections and activation functions. Each neuron's output influences the subsequent layer's neurons, culminating in the final output layer that generates predictions or classifications.

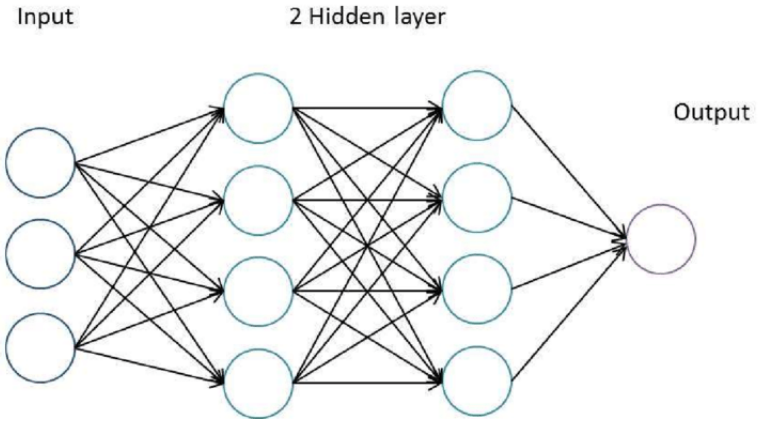


Figure 3.1.3: Example of a Feedforward Network [11]

Training feedforward neural networks often involves supervised learning [54] and optimization techniques to adjust the weights, optimizing their ability to capture complex patterns in data. The technique most commonly embraced for updating the weights of feedforward networks encompasses the utilization of a supervised algorithm known as Back Propagation [55], combined with an optimizer, e.g. Gradient Descent [56].

3.1.2.2 Reccurent Neural Networks

Recurrent Neural Networks (RNNs) constitute a dynamic architecture designed to process sequences of data by incorporating feedback loops, allowing them to maintain memory of previous inputs. Unlike feedforward networks, RNNs possess an internal state (known as hidden state) that captures information from earlier steps in the sequence, enabling them to comprehend temporal dependencies and patterns. This mechanism makes RNNs particularly adept at tasks involving sequences, such as language modeling and time series analysis. During processing, each input in the sequence is fed into the RNN one by one. The network not only processes the current input but also considers the context of preceding inputs stored in the hidden state. This iterative process allows RNNs to learn and recognize patterns across sequential data, making them valuable tools for tasks that involve sequential relationships and dynamics. Illustrated in 3.1.4, the RNN initiates by taking x0 from the input sequence and deriving h0 (hidden state). Subsequently, h0, combined with x1, forms the input for the subsequent step, initiating a similar pattern through subsequent steps.

In a mathematical manner, the RNNs are described as:

$$h_t = f(W_{hh}h_{t-1} + W_{hx}x_t) \tag{3.1.9}$$

$$y_t = W_s h_t \tag{3.1.10}$$

where:

- h_t: the hidden state at time t
- $x_t \in \mathbb{R}^d$: input vector at time-step t
- $W_{hx} \in \mathbb{R}^{d_h \times d}$: weights matrix used to condition the input vector x_t
- $W_{hh} \in \mathbb{R}^{d_h \times d_h}$: weights matrix used to condition the output of the previous time-step h_{t-1}.
- $f()$: non-linear activation function (e.g. tanh)

- $y_t \in \mathbb{R}^l$: the output at time-step t.
- $W_s \in \mathbb{R}^{d_h \times l}$: weights matrix used to take an output y_t

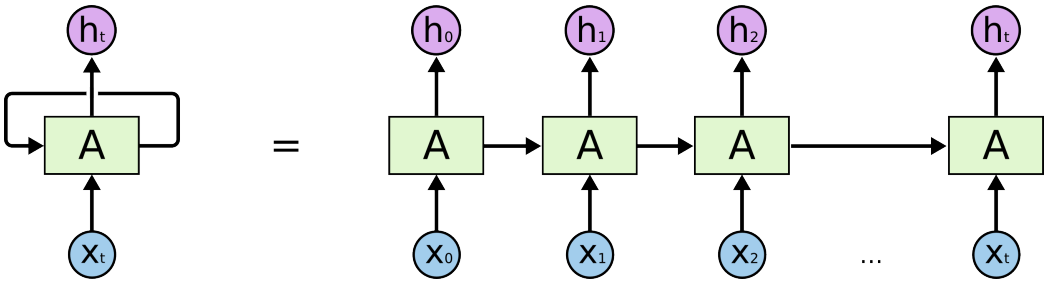


Figure 3.1.4: Recurrent Neural Network [12]

However, traditional RNNs can struggle with maintaining long-range dependencies due to vanishing gradient problem, leading to the development of more advanced RNN variants that mitigate these issues and enhance their ability to capture intricate patterns within sequences.

Long Short-Term Memory Neural Networks

To overcome the limitations of capturing long-range dependencies, in 1997, Sepp Hochreiter et al. [13] proposed an architecture called Long Short-Term Memory (LSTM). LSTMs address the vanishing gradient problem by introducing specialized memory cells that can store and retrieve information over extended sequences. These memory cells maintain their states over time, allowing LSTMs to effectively learn and retain information across distant time steps. The iterative process and internal unit structure of LSTMs is illustrated in 3.1.5. Each LSTM unit consists of the *cell state* and three gating mechanisms: *input gate, forget gate,* and *output gate*. These gates regulate the flow of information into, out of, and within the memory cell. By controlling the information flow, LSTMs can selectively retain or discard information, facilitating the capture of intricate patterns and temporal relationships in sequences. This capacity makes LSTMs highly effective for tasks like language modeling, speech recognition, and time series prediction, where capturing long-term dependencies is crucial for accurate predictions and modeling complex dynamics.

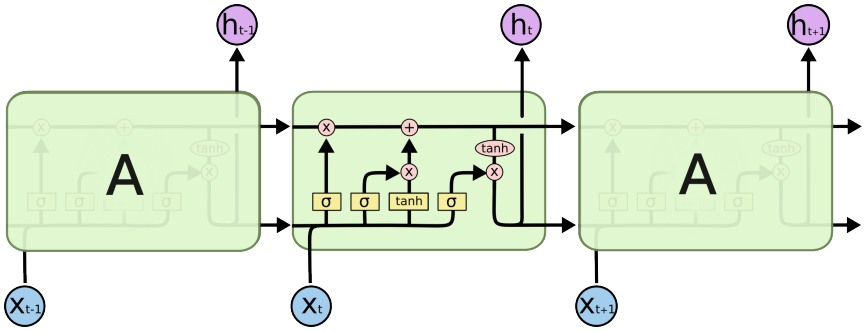
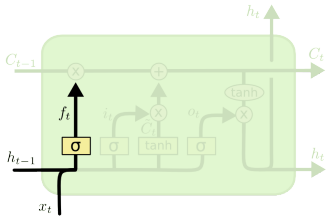


Figure 3.1.5: LSTM iterative process and internal structure. [57]

LSTM Cell State and Gate Mechanisms:

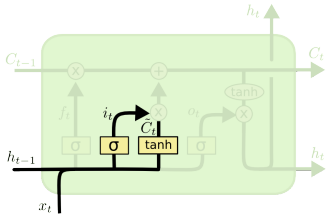
- **Forget gate**: The forget gate determines which information from the previous cell state (long-term memory) should be discarded. It evaluates the current input and hidden state to produce a forget factor, which ranges from 0 (discard) to 1 (retain). This forget factor is then applied to the previous cell state C_{t-1}, allowing the LSTM to decide what information is no longer relevant.
- **Input gate**: The input gate regulates the flow of new information into the cell state. It computes candidate values (the *tanh* part in figure 3.1.10) based on the current input and hidden state.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

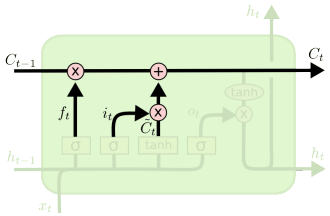
Figure 3.1.6: Forget Gate of LSTM [57]

The input gate then determines the amount of these candidate values that should be added to the cell state (the *sigmoid* part in figure 3.1.10) in order to calculate the new cell state C_t, allowing the LSTM to adapt its representation based on the new information.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

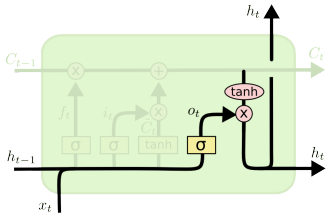
Figure 3.1.7: Input Gate of LSTM [57]



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 3.1.8: Update of cell state [57]

- **Output gate**: The output gate produces the final output. First, it processes the current input and hidden state to generate an output factor, which is then combined with the modified cell state to determine the information that will be passed on to the next time step.



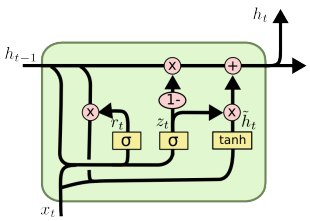
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 3.1.9: Output Gate of LSTM [57]

Gated Recurrent Unit

Gated Recurrent Unit (GRU) is simplified version of LSTM, proposed by Cho et al. [14]. Is featured by two main gates, an *update gate* (a combination of the forget and input gate of LSTM) and a *reset gate*. Also, the hidden state emerges by merging the hidden and the cell state of LSTM. The reset gate determines how much of the previous hidden state should be forgotten or reset, allowing the model to focus on new input information. The update gate decides how much of the new input

to incorporate into the hidden state. These gating mechanisms enable GRUs to effectively capture temporal dependencies and patterns in sequential data while simplifying the architecture compared to traditional LSTMs. The internal unit structure of GRU architecture is illustrated in following figure:



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3.1.10: GRU internal structure. [57]

3.1.2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) represent a pivotal advancement in deep learning, prominently shaped by the innovative contributions of Yann LeCun et al. proposing a CNN for document recognition [15], in 1998. These networks, inspired by the human visual system, demonstrate exceptional capabilities in processing visual data. LeCun's architectural breakthrough introduced a layered framework, empowering CNNs to adeptly learn and extract intricate features from input images, facilitating the discernment of spatial hierarchies and patterns. This transformation has propelled CNNs into diverse applications, encompassing domains such as medical imaging and autonomous vehicles, firmly establishing their pivotal role within the realm of modern artificial intelligence.

CNNs operate through a distinctive layering system tailored to automatically learn and extract essential features from input, primarily images. These layers include:

- **Convolutional layer**: Convolutional layers detect patterns by sliding small filters across input data, generating *feature maps* that capture relevant information such as edges, textures, and shapes. These layers excel in capturing spatial hierarchies and local patterns, making them essential for tasks like image recognition and object detection. The following figure illustrates an example of a convolutional layer.

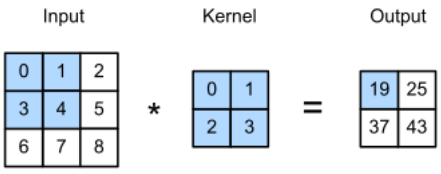


Figure 3.1.11: Example of convolutional layer [58]

- **Pooling layer**: Pooling layer downsample feature maps, reducing dimensionality while retaining crucial information. There are three main types of pooling: max pooling selects the maximum value from a local region, min pooling selects the minimum value from a local region, and average pooling calculates the average value. These layers enhance the network's efficiency and effectiveness in tasks like image recognition and object detection. The following figure illustrates an example of each a aforementioned pooling layer.

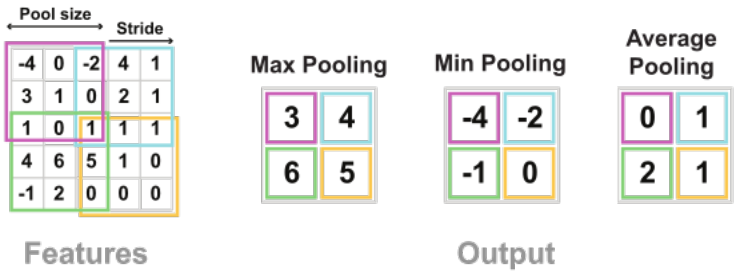


Figure 3.1.12: Example of pooling layers [59]

- **Fully Connected layer**: Fully connected layer connects each neuron to every neuron in the previous and subsequent layers, enabling comprehensive feature integration and mapping. This layer is vital for making final predictions in tasks like classification and regression. The following figure illustrates an example of a fully connected layer.

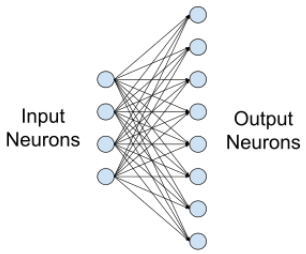


Figure 3.1.13: Example of Fully Connected layer [60]

3.1.2.4 Overfitting and Underfitting

Overfitting occurs when a machine learning model learns the training data too well, capturing not only the underlying patterns but also the noise and randomness present in the data. A model's complexity can contribute to this phenomenon. If a model is excessively complex, it may fit the training data so closely that it loses the ability to generalize effectively to new, unseen data. This leads to a model that performs exceptionally well on the training set but fails to generalize to real-world scenarios. In other words, it memorizes the training examples rather than learning the underlying concepts.

On the contrary, underfitting occurs when a machine learning model is too simplistic to capture the underlying patterns in the training data. This results in poor performance not only on the training data but also on new, unseen data. An underfit model lacks the capacity to grasp the complexities present in the dataset, essentially oversimplifying the relationships between input features and target outputs. It often stems from using overly basic algorithms or models with too few parameters. To address underfitting, one can explore more complex model architectures and increase the number of model parameters. Achieving a balance between model simplicity and its ability to capture intricate patterns is essential for avoiding underfitting.Figure 3.1.14 illustrates an example of underfitting, an ideal fit and overfitting.

When addressing the challenge of overfitting, a variety of approaches are involved. To be more precise, regularization constitutes a collection of methods that can hinder overfitting in neural networks, enhancing the precision of a Deep Learning model when encountering entirely novel data within the problem domain. A subset of these methods encompasses L1 regularization, L2 regularization, Dropout and data augmentation.

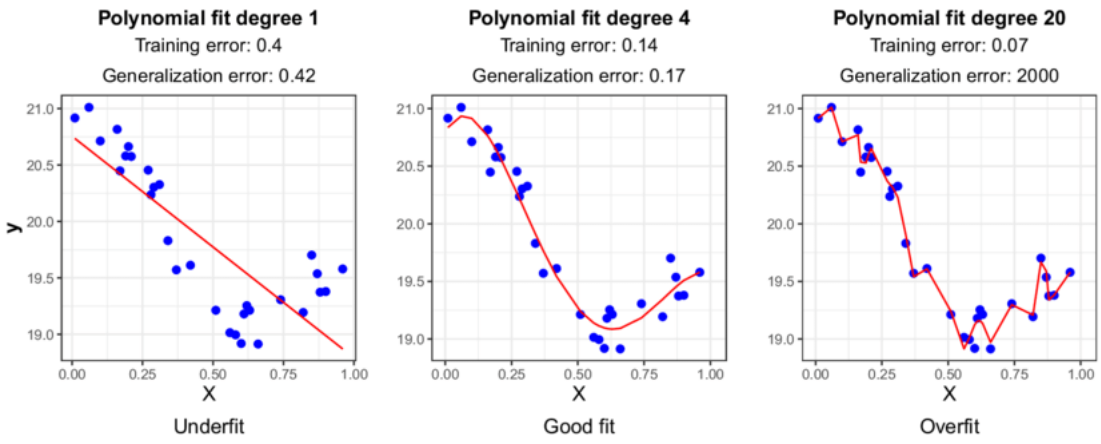


Figure 3.1.14: **Left**: Underfitting, the model is incapable of learning the data completely. **Center**: Ideal, the model acquires knowledge of the underlying data structure. **Right**: Overfitting, The model is overly complex and acquires noise as it learns from the training data. [61]

L1 and L2 regularization

L1 regularization, also known as Lasso regularization, introduces a penalty term to the model's loss function, based on the absolute values of the model's coefficients. By doing so, L1 regularization encourages the model to minimize the magnitudes of certain coefficients to zero, effectively leading to sparse feature selection. This process has a pruning effect, reducing the impact of less relevant features on the model's predictions. L1 regularization aids in simplifying the model and preventing it from memorizing noise in the training data, resulting in improved generalization to unseen data. Its ability to induce sparsity in the feature space makes L1 regularization particularly useful when dealing with high-dimensional datasets where feature selection is essential.

L2 regularization, often referred to as Ridge regularization, like L1 regularization, adds a penalty term to the loss function, but this time based on the squared magnitudes of the model's coefficients. Unlike L1, L2 regularization doesn't force coefficients to zero; instead, it encourages all coefficients to be small. This results in a smoother weight distribution. L2 regularization is effective in preventing large weight values and controlling the overall complexity of the model, contributing to better generalization performance.

In mathematical terms, the equations of L1 and L2 regularizations are:

$$\text{L1: } Loss = Error(Y - \widehat{Y}) + \lambda \sum_{1}^{n} |w_i| \tag{3.1.11}$$

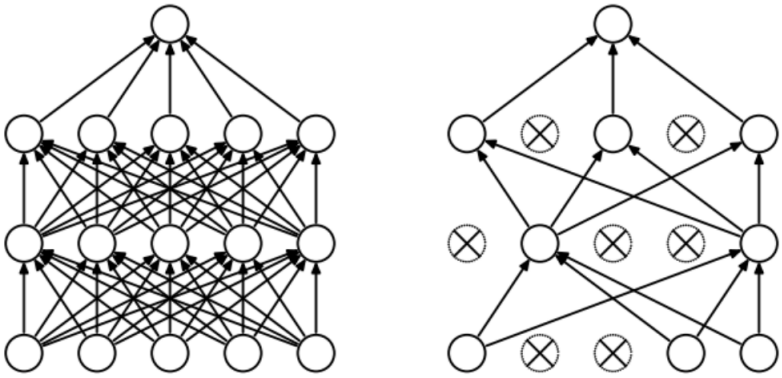
$$\text{L2: } Loss = Error(Y - \widehat{Y}) + \lambda \sum_{1}^{n} w_i^2 \tag{3.1.12}$$

where $Error$ is the Loss function of the model

Dropout

In contrast to L1 and L2 regularization, which modify the model's loss function, Dropout operates within the architecture of the neural network itself. During training, Dropout randomly deactivates a fraction of neurons at each iteration as shown in figure 3.1.15. This technique effectively creates an ensemble of subnetworks that share weights, preventing the network from relying too heavily on specific neurons for predictions. By introducing controlled randomness, Dropout serves as a potent

regularizer, reducing the risk of overfitting and enhancing the model's generalization ability. The process of deactivating neurons simulates a form of model averaging, leading to more robust and resilient networks. Dropout has become a standard practice in building deep neural networks, contributing significantly to their improved performance and adaptability across a range of complex tasks.

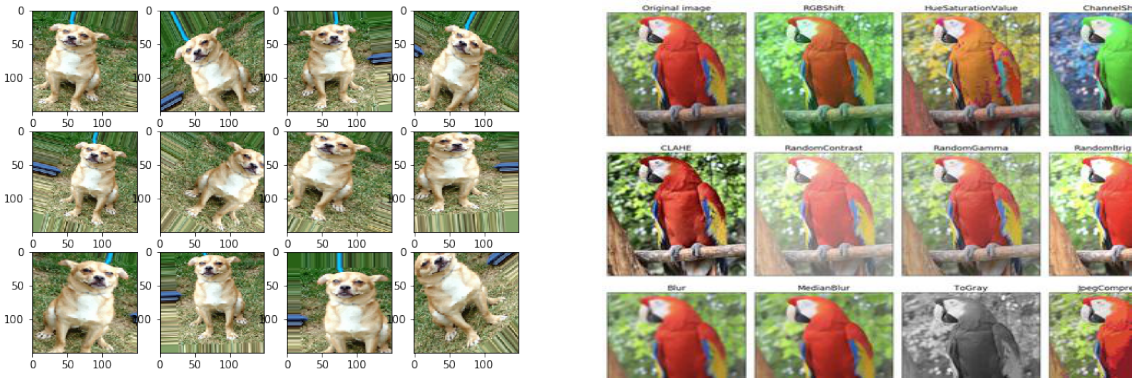


(a) Standard Neural Net (b) After applying dropout.

Figure 3.1.15: **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying Dropout to the network on the left. [62]

Data Augmentation

Unlike previous methods that focus on modifying model architecture or loss functions, data augmentation operates on the training data itself. It involves applying various transformations, such as rotation, scaling, cropping, and flipping, to create new instances of existing data points, as shown in figure 3.1.16. By diversifying the dataset with these altered versions, data augmentation helps the model learn to be more invariant to variations in the input data. This effectively enhances the model's ability to generalize by exposing it to a wider range of scenarios that it might encounter during inference. Data augmentation is particularly useful when working with limited training data, as it artificially expands the dataset and reduces the risk of overfitting. This technique has proven valuable across various domains, from computer vision to natural language processing, contributing to improved model robustness and accuracy on unseen data.



(a) Rotation Data Augmentation [63]

(b) Changing colors Data Augmentation [64]

Figure 3.1.16: Data augmentation examples

3.1.3 Reinforcement Learning

3.1.3.1 Introduction to Reinforcement Learning

Artificial Intelligence (AI) has undeniably transformed the way we live, work, and interact with technology. From natural language understanding to image recognition, AI systems have achieved remarkable feats, largely driven by the power of machine learning. Yet, there exists a class of AI challenges that extends beyond passive pattern recognition and mere data analysis. These are challenges where AI must learn to interact with dynamic environments, make sequences of decisions, and adapt its behavior through trial and error. This is where Reinforcement Learning (RL) emerges as a pivotal force, pushing the boundaries of AI capabilities and paving the way for autonomous systems that can navigate the complexities of the real world.

Reinforcement Learning [65] is a subfield of machine learning that takes inspiration from behavioral psychology and the principles of reward-driven learning. At its core, RL is concerned with the development of intelligent *agents* that learn to make a series of decisions in an *environment* to maximize a cumulative *reward* signal. Unlike supervised learning [54], where algorithms are trained on labeled datasets, and unsupervised learning [66], where algorithms uncover hidden patterns, reinforcement learning focuses on an agent's interaction with an environment. This interaction is marked by a continuous loop of observation, action, and feedback. Through this loop, RL agents learn to not just passively recognize patterns but to actively seek actions that lead to desired outcomes.

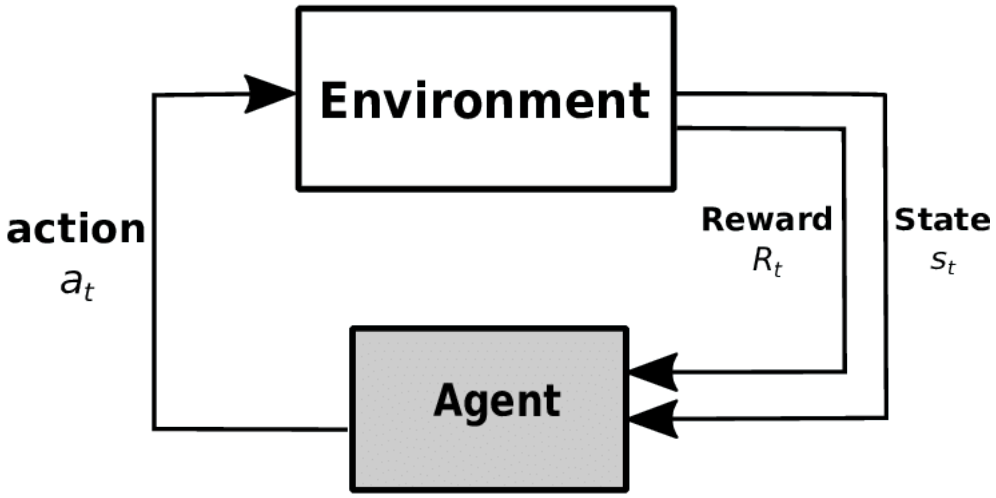


Figure 3.1.17: Reinfrocement learning loop [16]

One of the defining characteristics of reinforcement learning is its ability to handle sequential decision-making problems. In many real-world scenarios, decisions made at one moment can have a profound impact on future outcomes. For instance, in robotics, a robot must decide how to move its limbs or adjust its sensors to perform tasks effectively. In finance, an algorithmic trader must make decisions on buying or selling financial instruments to maximize profits over time. In healthcare, personalized treatment plans can be seen as sequences of medical decisions made to optimize a patient's well-being. Reinforcement learning equips AI with the cognitive capability to navigate these complex sequences of actions and consequences.

Reinforcement Learning has garnered significant attention in recent years due to its potential to solve complex problems that were previously deemed insurmountable for AI systems. It has enabled autonomous vehicles to navigate streets, defeating world champions in complex board games, and optimizing energy consumption in data centers. Moreover, it has shown promise in domains as diverse as healthcare, finance, robotics, gaming, and natural language processing.

3.1.3.2 Basics of Reinforcement Learning

Markov Decision Processes (MDPs)

Markov Decision Processes (MDPs) are a mathematical framework used in the field of reinforcement learning (RL) and decision-making under uncertainty. MDPs provide a structured way to model and solve problems where an agent interacts with an environment and makes decisions to maximize some notion of cumulative reward over time.

As Michael L. et al.[67] explain, the components of an MDP are:

- **States (S)**: States represent different situations or configurations of the environment. The agent and the environment jointly exist in one of these states at any given time. States can be discrete or continuous, depending on the problem. The set of all possible states forms the *state space*.
- **Actions (A)**: Actions are the choices or decisions available to the agent within each state. The set of all possible actions forms the *action space*. Actions can include physical actions, like moving a robot, or abstract decisions, like choosing a stock portfolio.
- **Transitions (T)**: Transitions describe how the environment changes from one state to another based on the agent's actions. They define the probabilities of moving from one state to another given a specific action. In simple terms, transitions answer questions like "If I take action A in state S, what is the probability of ending up in state S' ?"
- **Rewards (R)**: Rewards are numerical values associated with state-action pairs. They represent the immediate benefit or cost of taking a particular action in a given state. Rewards provide feedback to the agent about the quality of its decisions. The agent's goal is to maximize the cumulative reward over time.
- **Policy (π)**: A policy is a strategy that specifies which action the agent should take in each state. It maps states to actions and guides the agent's decision-making. Policies can be deterministic (i.e., they always choose the same action in a given state) or stochastic (i.e., they specify probabilities for taking different actions).

MDPs are based on the *Markov property*, which means that the future state depends only on the current state and the action taken, not on the sequence of states and actions that preceded it. This property simplifies modeling and computation, as the history of interactions is condensed into the current state.

Objective: The primary objective in MDPs is to find an optimal policy, denoted as π^*, that maximizes the expected cumulative reward over time. This optimal policy represents the best way for the agent to make decisions in the environment.

MDPs can be solved using various methods, including Bellman Equations (Dynamic Programming), value iteration, policy iteration, and reinforcement learning algorithms like Q-learning and Deep Q-Networks (DQN). These methods aim to find the optimal policy or approximate it to make good decisions in complex, real-world scenarios.

In summary, MDPs provide a mathematical framework to model decision-making problems where an agent interacts with an uncertain environment. They are fundamental to reinforcement learning and have applications in robotics, finance, game playing, and more, where agents must learn to make sequential decisions to achieve long-term goals.

3.1.3.3 Policy, Value Functions, and Bellman Equations

In the context of Markov Decision Processes (MDPs) and reinforcement learning, policy, value functions, and Bellman equations are essential concepts that help us understand and solve decision-making problems.

Policy (π): As already mentioned, a policy, denoted as π, is a strategy that defines the agent's behavior in an MDP. It specifies which action the agent should take in each state or state-action pair. Policies

can be deterministic, meaning they always select the same action for a given state, or stochastic, where they specify probabilities for taking various actions. The goal of reinforcement learning is often to find the optimal policy, denoted as π*. This optimal policy maximizes the expected cumulative reward over time.

Value Functions: Value functions are essential tools used to evaluate and compare different policies and states within an MDP. There are two primary types of value functions: *the state-value function (V)* [68] and the *action-value function (Q)* [69]:

- **State-Value Function (Vπ)**: This function estimates the expected cumulative reward an agent can achieve, starting from a particular state and following a given policy π. In other words, it quantifies how good it is to be in a particular state while following the policy π.
- **Action-Value or Q Function (Qπ)**: This function estimates the expected cumulative reward an agent can achieve when starting in a particular state, taking a specific action, and then following the policy π. It quantifies the expected goodness of taking a particular action in a particular state while following the policy π.

Bellman Equations: The Bellman equations (also known as Dynamic Programming eqution) are fundamental recursive relationships that connect the value functions of different states or state-action pairs.

- **Bellman Expectation Equation for Vπ**: This equation expresses the state-value function (Vπ) in terms of the expected reward (immediate reward) and the expected value of the next state, taking into account the current state and the action chosen according to the policy π. Mathematically, it can be represented as:

$$V_\pi(s) = \sum_{s'}[P_{\pi(s)}(s'|s, \pi(s)) * (R(s, \pi(s), s') + \gamma * V\pi(s'))] \tag{3.1.13}$$

  Here, *s* represents the current state, *s'* represents the next state, *π(s)* represents the action chosen in state *s* according to policy *π*, *P(s' | s, π(s))* represents the transition probability from *s* to *s'* under policy *π*, *R(s, π(s), s')* is the immediate reward, *γ* (gamma) is the discount factor (which values future rewards less than immediate rewards), and *Vπ(s')* is the value function of the next state *s'*.

- **Bellman Expectation Equation for Qπ**: This equation expresses the action-value function (Qπ) in terms of the expected reward (immediate reward) and the expected value of the next state, taking into account the current state, the action chosen, and the policy π. Mathematically, it can be represented as:

$$Q_\pi(s, a) = \sum_{s'}[P_\pi(s'|s, a) * (R(s, a, s') + \gamma * \sum_{a'} \pi(a'|s') * Q_\pi(s', a')] \tag{3.1.14}$$

  Here, *s* represents the current state, *a* represents the current action, *s'* represents the next state, *P(s' | s, a)* represents the transition probability from *s* to *s'* under action *a*, *R(s, a, s')* is the immediate reward, *γ* (gamma) is the discount factor, *π(a' | s')* represents the probability of taking action *a'* in state *s'* according to policy *π*, and *Qπ(s', a')* is the Action-Value function of the next state *s'* and action *a'*.

These Bellman equations play a crucial role in solving MDPs and finding optimal policies or value functions. They allow us to update and refine value estimates iteratively, ultimately leading to improved decision-making in uncertain environments.

3.1.3.4 Reinforcement Learning Algorithms

This section explores some of the most well-know Reinforcement Learning (RL) algorithms, ranging from foundational techniques like Value Iteration and Policy Iteration to modern innovations such

as Deep Q-Networks (DQNs), Q-Learning, Policy Gradient Methods, Proximal Policy Optimization (PPO), and Actor-Critic Methods. These algorithms play a pivotal role in the field of RL, significantly impacting the landscape of artificial intelligence. They offer adaptable solutions with applications in various contexts, making this exploration valuable for both experienced RL practitioners and those new to the field.

- **Value Iteration**: Value Iteration [70] is a dynamic programming algorithm widely used in Reinforcement Learning to compute the optimal value function and policy for a Markov Decision Process (MDP). The algorithm iteratively refines the value estimates for each state in the environment until convergence to the optimal solution. It's based on the principle of Bellman's Optimality Equation, which states that the optimal value of a state is the maximum expected return achievable from that state by following the optimal policy. The algorithm begins with initializing the value function *V (s)* for all states s in the MDP. It then iteratively updates the value estimates using the Bellman equation for $V\pi$ 3.1.13. An alternate method to write the algorithm is to use the idea of Q-values which is closer to a code-based implementation. For this, the loop is:

Algorithm 1 Value Iteration with Q-Values

Input: MDP $M = \langle S, A, P_a\left(s' \mid s\right), r\left(s, a, s'\right)\rangle$
Output: Value function V

Initiazlize V to arbitrary value function; e.g., $V(s) = 0$ for all s

while V changes (no convergence) **do**
  for each $s \in S$ **do**
    for each $a \in A(s)$ **do**
      $Q(s, a) \leftarrow \sum_{s' \in S} P_a\left(s' \mid s\right)\left[r\left(s, a, s'\right) + \gamma V\left(s'\right)\right]$
    end for
    $V(s) \leftarrow \max_{a \in A(s)} Q(s, a)$
  end for
end while=0

Value Iteration is guaranteed to find the optimal policy for finite MDPs. However, it can be computationally expensive for large state spaces due to its iterative nature. Nevertheless, it serves as a foundational concept for understanding other advanced reinforcement learning algorithms.

- **Policy Iteration**: Policy Iteration [71] is an iterative reinforcement learning method for finding the optimal policy in a Markov Decision Process (MDP). It consists of two main steps: *policy evaluation* and *policy improvement*. In the policy evaluation step, it estimates the value function Vπ(s) for a given policy π which represents the expected cumulative reward from a starting state s. The estimated values are calculated using the 3.1.13. In the policy improvement step, the goal is to improve the policy by updating the actions it recommends based on that we receive from the policy evaluation. Let Qπ(s, α) be the expected reward from s when doing a first and then following the policy π. Qπ can be defined as:

$$Q(s, a) \leftarrow \sum_{s' \in S} P_a\left(s' \mid s\right)\left[r\left(s, a, s'\right) + \gamma V\left(s'\right)\right] \tag{3.1.15}$$

This is the same equation used in Algorithm 1, but here the $V(s')$ term is the value function from the policy evaluation. If there is an action α such that $Q\pi(s,\alpha) > Q\pi(s,\alpha)$ then the policy π can be strictly improved by setting $\pi(s) \leftarrow \alpha$. Algorithm 2 demonstrates an example implementation of this concept.

The policy iteration algorithm finishes with an optimal π (π^*), after a finite number of iterations, because the number of policies is finite.

Algorithm 2 Policy Evaluation

Input: MDP $M = \langle S, A, P_\pi\left(s' \mid s\right), r\left(s, \pi(s), s'\right)\rangle$
Output: Policy function V

Initiazlize π to arbitrary value function; e.g., $\pi(s) = \alpha$ for all s, where $a \in A$ is an arbitray action

while π changes (not convergence) : **do**
  for each $s \in S$ **do**
    $Q_\pi(s, a) \leftarrow \sum_{s' \in S} P_a\left(s' \mid s\right)\left[r\left(s, a, s'\right) + \gamma V\left(s'\right)\right]$ {Policy Evaluation}
  end for
  for each $s \in S$ **do**
    $\pi(s) \leftarrow argmax_{\alpha \in A(s)} Q(s, \alpha)$
  end for
end while=0

Value and Policy iteration are the fundamental algorithms in the field of Reinforcement Learning. However, as RL continued to evolve and diversify, numerous additional algorithms emerged to address various complexities and challenges.

- **Q-Learning**: Q-Learning [17] is a pivotal algorithm in the realm of reinforcement learning due to its practical effectiveness and robustness in handling complex problems. One of its notable characteristics is its model-free nature, meaning it doesn't require prior knowledge of the environment's dynamics. Instead, it learns through a trial-and-error process by interacting with the environment. It is worth mention that Q-Learning operates on the principle of temporal difference [72]. At its core, Q-Learning revolves around estimating the value of taking a specific action in a given state. This estimation is often referred to as the Q-value. The algorithm iteratively updates these Q-values based on observed rewards and state transitions. The Bellman equation, once again, plays a central role in these updates, aiding in the calculation of the expected return for a given state-action pair. The core update equation for Q-Learning is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (R(s_t, a_t) + \gamma * max_Q Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{3.1.16}$$

  In this equation 3.1.16, $Q(s_t, a_t)$ represents the Q-value for state s_t and action a_t, α denotes the learning rate, controlling the step size of the updates, R stands for the immediate reward received after taking action a_t in state s_t, γ represents the discount factor, which balances the importance of immediate and future rewards and s_{t+1} represents the resulting state after taking action a_{t+1} in state s_t. In Q-Learning, for the update of the current $Q(s_t, a_t)$ is used the state-action pair $< s_{t+1}, a_{t+1} >$ that maximizes the Q-value. On the other hand, to pick the next action/state (from the current state) the epsilon greedy policy [73] derived from the current Q table is used. For this reason Q-Learning is called off policy algorithm, which means that uses different policy to explore and learn than the target policy. In this way, Q-Learning entails a delicate balance between exploring new actions (due to the randomness of epsilon greedy) to gather information about the environment and exploiting existing knowledge to maximize rewards.

  It is important to notice that a major limitation of Q-learning is that it is only works in environments with discrete and finite state and action spaces.

- **Deep Q-Networks**: Deep Q-Networks (DQNs) [74] represent a groundbreaking advancement in the field of reinforcement learning (RL). At their core, DQNs combine the power of neural networks with the Q-learning algorithm as shown in , enabling the handling of high-dimensional state spaces encountered in complex environments like video games and robotics. In a DQN, a neural network approximates the Q-function, with the input being the state and the output representing the Q-values for each possible action. This approach drastically improves the ability to generalize across states, making it possible for an agent to learn effective policies even when it

hasn't explored every state-action pair. Moreover, DQNs introduce the concept of experience replay, where past experiences are stored in a replay buffer and randomly sampled during training. This technique breaks the temporal correlations of sequential experiences, making learning more stable and efficient. DQNs have achieved remarkable success, outperforming humans in various Atari 2600 games and demonstrating their potential in real-world applications like autonomous driving. Despite their strengths, DQNs are not without challenges, including hyperparameter tuning, instability in training, and the need for substantial computational resources. Nevertheless, their ability to learn from raw sensory data and navigate complex environments marks a significant milestone in the realm of reinforcement learning.

Variations of the original DQN algorithm have also emerged, each with unique enhancements tailored to specific challenges. These variants include Double DQN (DDQN) [75], which mitigates the overestimation bias present in traditional DQNs, and Dueling DQN [76], which disentangles the value and advantage functions to improve learning efficiency. Additionally, Rainbow [77], an integration of multiple DQN extensions, further enhances the algorithm's performance, showcasing the ongoing innovation in this field. These adaptations highlight the dynamic nature of DQN algorithms, continuously evolving to address complex real-world problems effectively.

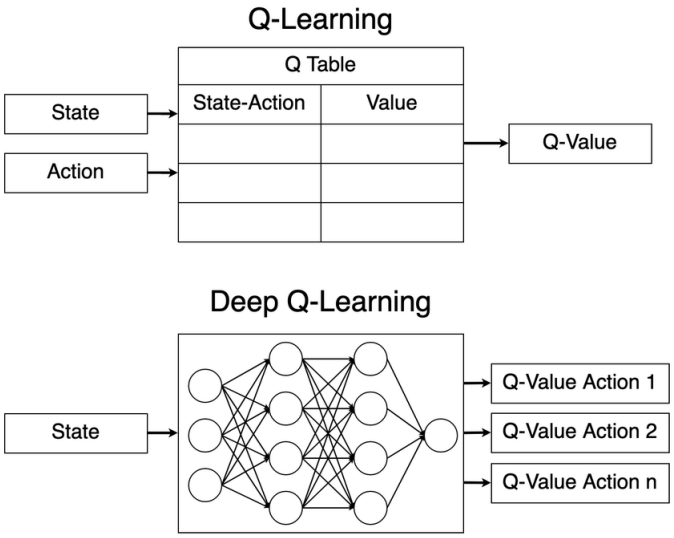


Figure 3.1.18: Q-Learning vs DQN [78]

- **Policy Gradient methods**: Policy gradient algorithms [79] represent a class of reinforcement learning (RL) techniques that directly learn the optimal policy for an agent. Instead of estimating the value function, as done in methods like Q-learning, policy gradient methods focus on finding the policy that maximizes the expected cumulative reward. At the heart of policy gradient algorithms is the parameterization of the policy itself, often using neural networks. The policy network takes the current state as input and outputs the probabilities of taking different actions. During training, these networks are optimized using gradient descent to maximize the expected reward. One of the fundamental policy gradient algorithms is the REINFORCE algorithm, which uses the likelihood ratio gradient estimator to update the policy parameters. While effective, REINFORCE can suffer from high variance in its updates, which can slow down learning. To address this issue, several variations and improvements of policy gradient methods have been introduced. Proximal Policy Optimization (PPO) [18] PPO is one algorithm designed to address this issue. It introduces a clever modification to the policy update step by incorporating a clipping mechanism. This clipping constrains the policy update to be within a certain range, preventing overly large changes in policy. By doing so, PPO achieves more stable learning and faster convergence. PPO also employs multiple epochs of minibatch updates on collected data, further

enhancing its robustness. It's known for its simplicity and effectiveness, making it a popular choice for various RL tasks. PPO's ability to balance exploration and exploitation and its ease of use have contributed to its widespread adoption in both research and practical applications. Trust Region Policy Optimization (TRPO) [80] is another policy gradient method that aims to ensure stable policy updates. It introduces a key concept known as a "trust region." In simple terms, a trust region is a region around the current policy where the algorithm is allowed to make updates. However, the updates should not venture too far from the current policy to avoid destabilizing learning. TRPO rigorously maintains a bound on how much the new policy can deviate from the old one, effectively controlling the magnitude of policy updates. This ensures that the changes made during training are gradual and do not lead to catastrophic policy collapses. TRPO's trust region approach provides strong theoretical guarantees on policy improvement, making it an attractive choice for safety-critical applications. While TRPO's stability and guarantees are appealing, it can be computationally demanding due to the need for constrained optimization. Researchers have explored approximations and extensions to TRPO to strike a balance between stability and efficiency. Additionally, Actor-Critic methods [81] combine elements of both value-based and policy-based approaches. In these methods, an actor network parameterizes the policy, while a critic network estimates the value function. The actor is responsible for selecting actions. It learns a policy, which is essentially a strategy for choosing actions in different states. In detail, the actor takes the current state as input and outputs the probability distribution over possible actions. Over time, through training, the actor's policy becomes more refined, aiming to maximize expected rewards. The critic, on the other hand, evaluates the actions taken by the actor. It learns the value function, which estimates the expected cumulative reward an agent can achieve from a given state by following the actor's policy. This value function provides feedback to the actor, helping it adjust its policy to select better actions. The Actor-Critic method combines the advantages of both policy-based and value-based methods. It can learn both the optimal policy and estimate the state's value simultaneously, making it more stable and efficient. Figure 3.1.19 illustrates the Actor-Critic architecture. This approach is particularly useful in scenarios where dealing with high-dimensional state spaces and complex environments is crucial. Actor-Critic methods come in various forms, and they are often employed in tasks such as robotic control, game playing, and autonomous navigation. Variants of Actor-Critic methods, such as A3C (Asynchronous Advantage Actor-Critic) and A2C (Advantage Actor-Critic) [82], have demonstrated remarkable success in various RL tasks.

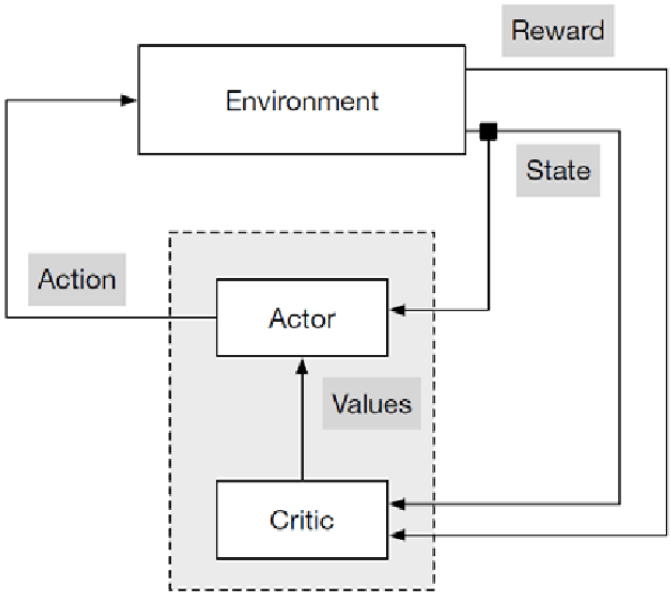


Figure 3.1.19: Actor-Critic RL Architecture [83]

Policy gradient algorithms and their variations offer a powerful framework for tackling RL problems, especially in scenarios with high-dimensional action spaces or continuous action domains. These methods, with their ability to directly optimize policies, have been instrumental in solving complex tasks in robotics, natural language processing, and game playing, among others. While they can be computationally intensive, the versatility and effectiveness of policy gradient algorithms make them a vital component of the RL toolkit.

3.1.3.5 Applications of Reinforcement Learning

Reinforcement Learning (RL) has found a wide range of applications across various domains due to its ability to make autonomous decisions through trial and error. Here's an extensive look at its applications:

- **Game Playing and AI**: RL has made significant strides in game playing. AlphaGo [84], developed by DeepMind, achieved a historic milestone by defeating Lee Sedol, one of the world's top Go players, in 2016. Go is an ancient board game with an incredibly vast and complex state space, making it a benchmark for AI. AlphaGo's success demonstrated RL's capacity to handle intricate and strategic games. In video games, agents trained using RL techniques have mastered games like Dota 2 [85] and StarCraft II [86], both of which demand high-level strategic planning, coordination, and real-time decision-making. These accomplishments are not just limited to scripted gameplay but also involve adapting to evolving environments and opponents.

  It is worth mentioning that RL's success in game playing has led to its adoption in simulating environments such as flight simulation. Simulated games can replicate real-world scenarios and allow AI agents to learn from various situations without the associated risks. This is especially valuable for the advancement of RL in fields like robotics.

- **Robotics**: Reinforcement Learning (RL) has emerged as a cornerstone in the field of robotics, revolutionizing the way robots learn and adapt to complex, unstructured environments. RL empowers robots with the ability to autonomously acquire and refine their skills, making them increasingly versatile and capable of performing a wide array of tasks. This technology is instrumental in the development of autonomous vehicles [87], drones [88], and industrial robots [89]. In the realm of autonomous vehicles, RL is leveraged to optimize control strategies, enabling self-driving cars to navigate safely through dynamic traffic scenarios. Drones benefit from RL by learning how to fly more efficiently, perform precise maneuvers, and adapt to changing environmental conditions. In industrial settings, RL is used to train robots for tasks like pick-and-place operations, assembly line tasks etc. RL-driven robots can adapt in real-time, making them indispensable in environments where unforeseen challenges often arise.

- **Healthcare**: Reinforcement learning (RL) is making significant inroads into the healthcare industry, revolutionizing patient care, diagnostics, drug discovery, and treatment planning. In personalized medicine, RL algorithms can analyze vast patient data sets to tailor treatments and therapies for individuals, optimizing outcomes. For instance, RL-driven predictive models have been used to identify at-risk patients for chronic diseases like diabetes or to optimize insulin dosage for diabetic patients [90]. In diagnostics, RL-based systems can assist medical professionals by providing more accurate and rapid diagnoses, such as detecting anomalies in medical images or interpreting medical texts [91]. Moreover, in drug discovery, RL accelerates the search for potential drug candidates [92]. Additionally, RL plays a pivotal role in treatment planning, suggesting personalized rehabilitation plans for patients [93]. The ability of RL to analyze and adapt to complex medical data makes it an indispensable tool in healthcare, with the potential to enhance patient care, improve outcomes, and expedite medical discoveries.

- **Finance**: In the realm of finance, RL plays a pivotal role in algorithmic trading, trading strategy optimization, portfolio management, fraud detection, cybersecurity, and dynamic pricing. According to algorithmic trading, RL-driven trading systems can adapt and optimize trading strategies based on market dynamics, enhancing profitability and risk management [94]. In portfolio management, RL aids in the allocation of assets to maximize returns while minimizing risk

[95]. In the realm of fraud detection and cybersecurity, RL models can continuously learn to identify and respond to evolving threats in real time [96]. Furthermore, RL plays a role in dynamic pricing strategies [97] for businesses, optimizing pricing decisions based on market conditions and consumer behavior. With its capacity to make data-driven decisions in a dynamic and uncertain environment, RL is poised to make a significant impact on financial institutions, providing them with powerful tools to navigate the ever-evolving landscape of finance.

In conclusion, the applications of reinforcement learning are vast and continually expanding, revolutionizing numerous domains. From mastering complex games to enabling autonomous robotics, optimizing healthcare treatments, and enhancing decision-making in finance, RL has demonstrated its versatility and transformative potential. As researchers push the boundaries of what's possible in artificial intelligence, the future promises even more innovative applications and breakthroughs. With ongoing advancements in algorithms and increasing computational capabilities, the impact of reinforcement learning on society, technology, and industry is bound to grow, fostering a new era of intelligent systems that adapt and learn from their environments.

3.1.3.6 Challenges of Reinforcement Learning

Despite the fact that is a powerful paradigm with a growing list of applications spanning numerous domains, harnessing the potential of Reinforcement Learning comes with its own set of formidable challenges that researchers and practitioners must navigate. These challenges represent critical aspects of RL's development and deployment. This section presents some of the primary challenges that confront RL, including exploration, sample efficiency, generalization and transfer learning, as well as safety and ethical considerations. By addressing these challenges head-on, we aim to illuminate the path forward for RL in diverse applications, ensuring responsible and effective use across various domains.

- **Exploration vs Exploitation**: Achieving the delicate balance between exploration and exploitation [98] is a cornerstone challenge in reinforcement learning (RL). In RL, agents are required to actively explore their environment by trying out different actions and closely observing the outcomes. Striking the right balance between exploring new actions to gather valuable information and exploiting known actions to maximize immediate rewards is particularly intricate in complex and dynamic environments. This challenge becomes even more pronounced when dealing with environments characterized by high uncertainty, as agents must decide when and how to explore new possibilities while not sacrificing the performance gained through exploitation of previously learned actions.
- **Sample Efficiency**: Sample efficiency is a critical challenge in reinforcement learning (RL) [99], where algorithms aim to learn and improve policies with minimal data. In this context, sample efficiency refers to an algorithm's ability to make the most of the data it collects, allowing it to learn and improve policies quickly. RL algorithms that are sample-efficient can achieve better performance with the same number of training samples compared to less sample-efficient methods. For instance, a human player can master a game like Pong with just a few dozen trials, while some RL algorithms may require tens of thousands of samples to learn effective policies. This challenge is particularly significant due to the costs associated with real-world interactions and simulations, as well as considerations of time, energy, and equipment wear and tear. Developing more sample-efficient RL algorithms is crucial for practical applications in various domains, where efficiency and speed of learning are paramount.
- **Generalization and Transfer Learning**: Achieving robust generalization and transfer of knowledge across diverse environments remains a significant challenge in RL [100]. RL models often struggle to apply learned policies effectively in contexts different from their training environment, limiting their adaptability and scalability. This challenge necessitates the development of innovative techniques for enabling RL agents to generalize their knowledge and transfer it seamlessly to varied scenarios.
- **Safety Considerations**: The deployment of reinforcement learning in real-world applications,

such as autonomous driving and robotics, has brought forward significant safety concerns [101]. These concerns are primarily rooted in the potential risks associated with RL systems making critical decisions, which could lead to accidents or undesirable outcomes. As a result, there's a growing demand for the development of safe RL algorithms that can operate reliably and adhere to safety constraints. While the field of control theory has a long history of addressing safety in systems, the study and development of safe RL algorithms are still relatively nascent. Researchers are actively working on creating algorithms that not only optimize performance but also prioritize safety, ensuring responsible and secure RL deployment in various domains.

3.1.3.7 Reinforcement Learning in Practice

Transitioning from theoretical concepts to real-world applications involves addressing several practical considerations. One crucial aspect is the availability of robust tools, libraries, and reinforcement learning frameworks designed to facilitate RL research and development. In the earlier days of RL, implementing algorithms from scratch was a formidable challenge, often requiring extensive mathematical and programming expertise. However, the advent of specialized RL frameworks has significantly simplified the process, allowing researchers and practitioners to focus on the core components of their applications rather than getting bogged down in algorithmic details.

Prominent platforms like OpenAI Gym [102], TensorFlow [103], and PyTorch [33] provide essential resources, environments, and abstractions for training and evaluating RL agents, making it more accessible for researchers and practitioners. These frameworks offer standardized interfaces, optimized performance, and pre-built environments that can simulate a wide range of scenarios. This not only saves time but also ensures the reliability and reproducibility of RL experiments.

Moreover, specialized reinforcement learning frameworks like KerasRL [104], Stable Baselines [105], Stable Baselines 3 [106], PyQlearning [107], TensorForce [108], RLCoach [109], TF-Agents [110], and RLLib [111] have gained traction for their tailored support and optimized implementations of RL algorithms. These frameworks offer a wide range of pre-implemented algorithms, making it easier for developers to experiment with different approaches, fine-tune hyperparameters, and adapt RL methods to their specific applications. Importantly, many of these frameworks are designed to seamlessly integrate with popular environments like the aforementioned OpenAI Gym, allowing users to harness the power of RL while working within familiar and standardized simulation environments. This democratization of RL through user-friendly frameworks is reponsible for the accelerated innovation and the practical adoption of RL in various domains.

3.2 Serverless Computing

This chapter introduces the concept of serverless computing, a revolutionary paradigm in cloud computing. This approach transforms traditional application development by abstracting the complexities of infrastructure management, allowing developers to concentrate on code logic.

3.2.1 The Need of Serverless Computing

Cloud computing facilitates the delivery of computational services via the Internet. As defined by NIST [112], traditional cloud computing encompasses three service categories:

- **Infrastructure as a Service (IaaS)**: Infrastructure as a service is a form of cloud computing that provides virtualized computing resources over the internet. The cloud provider manages IT infrastructures such as storage, server and networking resources, and delivers them to subscriber organizations via virtual machines accessible through the internet. AWS, for instance, offers services like Elastic Compute Cloud (AWS EC2) [113] and Simple Storage Service (AWS S3) [114]. IaaS is a cost-efficient solution to operate a workload without having to buy, manage and support the underlying infrastructure. However, IaaS retains the operational complexities of applications, leaving developers responsible for tasks such as resource provisioning and application code management.

- **Software as a Service (SaaS)**: SaaS, on the other hand, enables developers to directly access cloud provider applications like Google's Gmail [115] and Docs [116]. Operating under the SaaS umbrella, the intricacies of operations are gracefully concealed, sparing developers the burdens of intricate management. However, this simplicity doesn't come without trade-offs. While SaaS hides operational intricacies, it imposes limitations and may entail relinquishing control over the application.
- **Platform as a Service (PaaS)**: PaaS provides developers with the means to develop, run, and manage applications using cloud provider-supported execution environments, exemplified by Google's App Engine [117] and Azure's App Service [118]. PaaS strikes a balance between IaaS and SaaS, offering a middle ground where certain operational complexities are abstracted away. However, it doesn't entirely absolve developers from all management duties, requiring them to still handle a portion of management and configuration tasks, potentially complicating development [119].

To alleviate the cloud management burden on software developers, cloud providers introduced a novel paradigm called *serverless computing*. Similar to PaaS [[120], [121]], serverless computing minimizes complex management tasks concerning underlying servers ("server-less") while retaining application control. Moreover, it offers automatic scaling based on demand, distinguishing it from PaaS [[122], [119]]. However, unlike PaaS, serverless computing is unsuitable for long-running processes and stateful applications [[123]].

Serverless applications adhere to the design principles of the microservice software style, which involves dissecting applications into discrete, independent tasks. In practice, serverless computing and microservices share notable commonalities. Both paradigms share the objective of disassembling monolithic applications into manageable units that can be developed, deployed, and maintained individually. The execution units within serverless applications, termed serverless functions, can be perceived as a means of hosting microservices. When it comes to monitoring and management, as the number of components within an application increases, the complexity of overseeing them rises, necessitating robust monitoring and log management tools. However, distinctions exist between serverless computing and microservices. For instance, serverless functions operate at a finer granularity than traditional microservices, potentially fulfilling multiple functions within a single unit. Moreover, in the context of constructing microservices-based applications, developers find themselves tasked with additional responsibilities, encompassing crucial aspects like scalability, fault tolerance, and load balancing. These responsibilities require strategic planning, implementation, and ongoing management to ensure the seamless performance of the distributed architecture. In contrast, the provisioning of infrastructure for serverless applications follows a different trajectory. This responsibility is borne by the serverless providers themselves, freeing developers from the intricacies of infrastructure management.

3.2.2 Architecture of Serverless Computing

Serverless architecture, a cornerstone of modern cloud computing, empowers developers to construct intricate applications without the burdens of traditional infrastructure management. At its core, this architecture revolves around the concept of serverless functions orchestrated within a cloud-based environment. Developers harness cloud providers' specialized platforms to craft applications composed of multiple serverless functions, each designed to execute specific tasks, such as running a Deep Learning model (also called model *inference* [124]). These functions are intricately tied to predefined events, such as HTTP requests or data updates within cloud storage, e.g. databases, acting as triggers for their execution. When an event occurs, the serverless platform automatically initializes the runtime environment, be it containers or virtual machines, needed to execute the designated function. Once executed, the resources are recycled. This intricate orchestration of events, functions, and resource allocation lies at the heart of the serverless architecture, streamlining application development and ushering in a new era of dynamic, event-driven computing.

To better understand the event-driven architecture of serverless computing, consider a practical use case that illuminates its seamless integration into user interactions. Imagine a scenario where a user

initiates search and purchase requests directly from their web browser. This scenario is illustrated in figure 3.2.1. These requests are guided through an API Gateway, the entrance to the serverless ecosystem. The Gateway efficiently directs the search request to a dedicated search function and the purchase request to a specialized purchase function. These functions, akin to skilled workers, diligently carry out their designated tasks within the dynamic serverless environment.

In parallel with function execution, the serverless platform instantaneously springs into action, constructing the necessary runtime environments. These environments empower the functions to perform their tasks.

Upon task completion, the functions generate responses containing data or status updates. These responses navigate back through the API Gateway, making their way to the user's web browser. This well-coordinated procedure showcases how serverless architecture adeptly manages tasks and responses, ultimately improving the user experience.

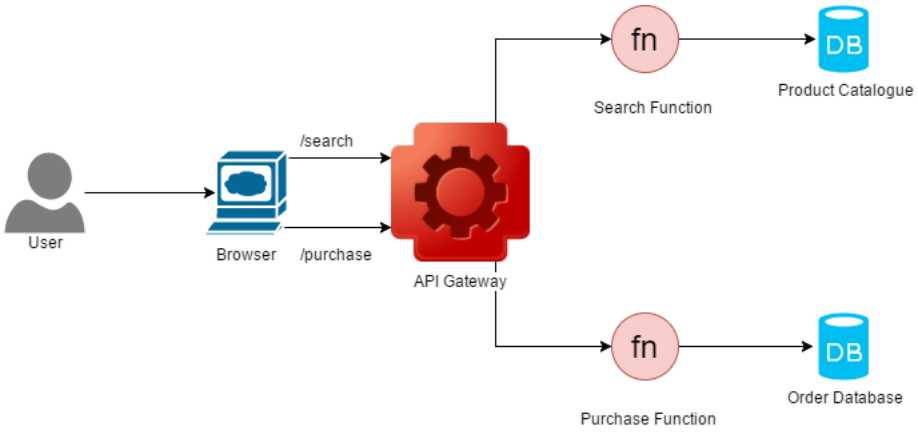


Figure 3.2.1: Step-by-step flow of serverless computing through a user-initiated search and purchase request [125]

3.2.3 Key Characteristics of Serverless Computing

The following outlines the key characteristics that define serverless computing, setting it apart from traditional application architectures. These characteristics collectively redefine how applications are developed, executed, and managed in the cloud environment.

- **Event-Driven Execution**: At the core of serverless architecture lies an event-driven model that orchestrates the execution of functions based on specific triggers. These triggers encompass a range of events, from HTTP requests and database updates to file uploads and beyond. This dynamic mechanism ensures that functions spring into action precisely when required, enabling resource utilization to be maximized.
- **Automatic Scaling**: Serverless platforms exhibit automatic horizontal and vertical scaling in response to fluctuations in application workloads. Horizontal scaling involves launching new function instances or recycling existing ones, while vertical scaling entails adjusting the computation resources allocated to function instances. After handling requests, function instances and resources remain in memory briefly for potential reuse by subsequent requests. In the absence of such requests, these resources are automatically recycled, potentially leading to a *scaling to zero* scenario. In this scenario, the serverless platform intelligently de-allocates all resources associated with the function instance, making it resource-efficient. However, this efficient approach can give rise to the "*cold start problem*, where a new function instance experiences a delay in its response time due to the need for time-consuming environment setup before executing the task at hand. This trade-off between resource efficiency and initial response time is a characteristic consideration in serverless design.

- **Ephemeral Statelessness**: Within the realm of serverless computing, functions are purposefully crafted to embrace a stateless and ephemeral nature. This foundational design principle ensures that each function invocation is devoid of any remnants from prior executions. By maintaining a stateless disposition, these functions avoid the complexities of managing persistent data across invocations, streamlining both development and maintenance processes. Moreover, the ephemeral nature of serverless functions aligns harmoniously with the dynamic nature of cloud-native architectures. This design facilitates rapid and efficient scaling, enabling functions to be instantiated or recycled swiftly in response to varying workloads. The marriage of statelessness and ephemerality not only simplifies function management but also contributes to enhanced fault tolerance. The lack of persistent state prevents failures in one instance from affecting others, bolstering the overall reliability and resilience of serverless applications.
- **Microservices Composition**: Serverless encourages a microservices-oriented approach to application development. Functions, acting as microservices, perform specific tasks, enabling developers to compose complex applications by integrating these functions through APIs. This microservices-driven strategy not only enhances modularity and maintainability but also promotes reusability and scalability of individual components, fostering an environment of rapid innovation in application design.
- **Pay-as-You-Go Billing**: Serverless operates on a pay-as-you-go pricing model, which represents a significant departure from traditional cloud billing. Users are exclusively billed for the exact resources consumed during function execution. This approach eliminates the need to pay for idle resources, aligning costs directly with usage. By embracing an event-driven philosophy, where functions remain dormant until triggered by specific events, serverless minimizes wastage and ensures cost efficiency. This approach empowers users to optimize their expenditures while taking full advantage of cloud resources, revolutionizing the way computing costs are managed.
- **Infrastructure Abstraction & Faster Development**: Serverless computing introduces a paradigm where developers are shielded from the intricacies of underlying infrastructure. Instead, cloud providers assume the role of managing server provisioning, maintenance, and scaling. This abstraction empowers developers to concentrate exclusively on crafting code that delivers functionality and meets business requirements. By delegating infrastructure management to experts, developers can streamline their workflows, accelerate development cycles, and focus on creating value through innovative applications.
- **Simplified Deployment Process**: Serverless architecture simplifies function deployment. Developers can easily upload code to the serverless platform, bypassing intricate configurations. This accelerates development and reduces overhead. User-friendly interfaces and tools further improve the packaging, uploading, and management process, enhancing the deployment experience.

3.2.4 Serverless platforms

As we explore the serverless landscape, we encounter a multitude of platforms, encompassing both proprietary and open-source solutions. Noteworthy proprietary platforms, including AWS Lambda [19], Microsoft Azure Functions [20], Google Cloud Functions [21], and IBM Cloud Functions [126], furnish developers with robust tools and seamless integrations, simplifying the deployment and management of serverless functions. On the open-source front, platforms like Knative [22], Apache OpenWhisk [9], OpenFaaS [24], and OpenLmbda [127] present versatile options, granting organizations the ability to tailor their serverless environment to their precise requirements. This selection of platforms empowers us to choose the one that best aligns with our project's demands and resonates with our technological preferences. The following bullets provide a brief overview of each of these platforms:

- **AWS Lambda**: Amazon Web Services (AWS) Lambda stands at the forefront of serverless computing, providing a powerful platform for executing code without the complexities of infrastructure management. Since its debut in November 13, 2014, serverless computing has been steadily capturing more attention, driving other major cloud providers to join the movement

by introducing their own serverless platforms. Lambda's "pay-as-you-go" pricing model charges based on actual execution time and memory usage, ensuring cost efficiency. Its integration with various AWS services enables the creation of event-driven architectures, where Lambda functions respond to real-time changes and triggers from both AWS and external sources. With built-in auto-scaling, Lambda dynamically allocates resources to handle incoming requests, maintaining optimal performance. Through features like provisioned concurrency, cold start delays are mitigated, ensuring rapid execution.To enhance the monitoring capabilities of AWS Lambda, Amazon delivers tools like AWS CloudWatch [128] and AWS CloudTrail [129], facilitating the observation and logging of serverless functions.

- **Microsoft Azure Functions**: As a significant player in the serverless landscape, Azure Functions by Microsoft simplifies application development by allowing developers to focus solely on code logic. Much like AWS Lambda, the platform offers event-driven capabilities and seamless integration with other Azure services. Azure Functions ensures the observability of functions with tools like Azure Monitor, which provides comprehensive insights into function behavior. With a "pay-as-you-go" pricing model, users are billed based on memory consumption during execution, promoting cost efficiency. Also, supports multiple programming languages and through autoscaling ensures optimal performance. Its robust integration with Azure services and efficient resource management make it a versatile choice for modern application development within the Microsoft Azure ecosystem.

- **IBM Cloud Functions**: A notable contender in serverless computing, IBM Cloud Functions, based on the OpenWhisk framework, optimizes development by abstracting infrastructure management complexities. Following an event-driven model, it seamlessly integrates with diverse IBM Cloud services, fostering the creation of flexible and responsive solutions. Utilizing consumption-based pricing, IBM Cloud Functions charges users solely for resources used during function execution. Compatibility with multiple programming languages allows developers to work seamlessly in their preferred coding environments. The platform's dynamic auto-scaling mechanism ensures optimal performance, adapting to changing workloads.

- **Knative**: The Knative framework operates as an open-source platform for building, deploying, and managing serverless workloads on Kubernetes clusters, making it a powerful tool for developers. At its core, Knative leverages Kubernetes to create an extensible platform that automates the orchestration of serverless deployments. Serving as a middleware layer, Knative encompasses two key components: Serving [130] and Eventing [131]. The Serving component focuses on handling traffic and scale-to-zero deployments, while Eventing manages event-driven architectures. Knative's extensible nature allows developers to customize and extend its capabilities to suit their specific application needs. This framework aligns seamlessly with the Kubernetes ecosystem, harnessing the power of Docker containers [132] and Kubernetes [23] clusters to simplify the development and management of serverless applications. More about Knative in section 3.2.5.

- **Apache OpenWhisk**: Apache OpenWhisk functions as a comprehensive open-source serverless computing framework, integrating key components to streamline application development. In OpenWhisk, HTTP requests can be utilized to transform function invocations, which are then directed to the Nginx [133] server that supports the Web protocol. From there, the Nginx server forwards the request to the controller, which collaborates with CouchDB [134], to store the data of the application. Then, OpenWhisk takes advantage of Kafka [135], a messaging system built on the publish-subscribe model [136] for efficient message transmission, to connect Controller with Invokers, responsible for managing the execution of functions. Finally, the controller sends, through Kafka, a message to the Invokers to initiate the execution of function instances (on worker nodes) in response to the requests, managing their runtime environments, and collecting results upon completion. OpenWhisk's programming model encompasses fundamental concepts: actions represent executable functions, triggers denote predefined events, and rules define the binding relationship between actions and triggers. This intricate system is designed for compatibility with Docker container engine and orchestration systems such as Kubernetes and OpenShift [137]. Figure 3.2.2 illustrates the architecture of Apache OpenWhisk.

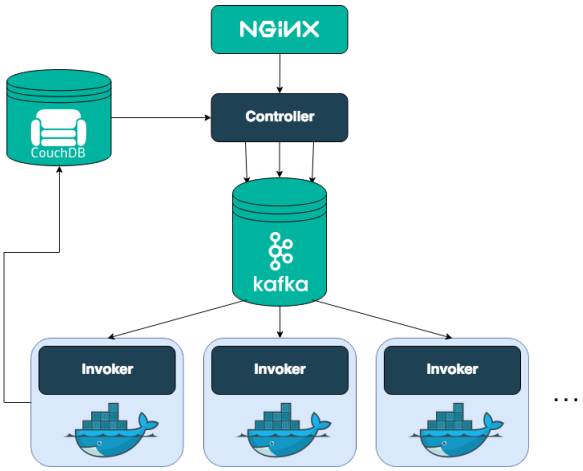


Figure 3.2.2: Apache OpenWhisk architecture [138]

- **OpenFaaS**: OpenFaaS comprises several key components that collectively enable its functionality. At its core is the Gateway, which acts as the entry point for invoking functions and managing requests. Functions themselves are packaged as Docker containers, ensuring portability and flexibility. The Gateway then invokes the orchestration system e.g. Docker Swarm or kubernetes (through *faas-netes* [25]) to execute the function. Additionally, the Prometheus monitoring system is integrated to collect performance metrics and ensure observability. OpenFaaS provides a set of built-in templates for various programming languages, streamlining function creation. Developers can utilize these templates to quickly define and package their functions. The platform also supports asynchronous function invocations through a queue, using NATS streaming, that enables decoupling of components for improved scalability and reliability. Furthermore, OpenFaaS incorporates an extensible ecosystem, allowing users to integrate additional services and plugins to extend its capabilities. This modular architecture empowers developers to create, deploy, and manage functions efficiently while taking advantage of the flexibility and scalability offered by OpenFaaS. Figure 3.2.3 illustrates the architecture of OpenFaas.

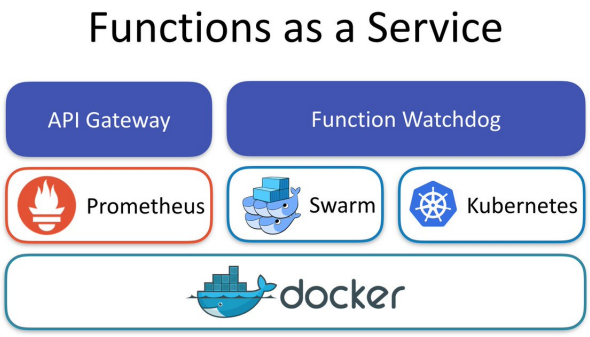


Figure 3.2.3: OpenFaas architecture [139]

- **OpenLambda**: OpenLambda stands as an open-source serverless platform under the Apache license, functioning on the foundation of Linux containers. Developers are required to upload their functions to the code store or function registry within OpenLambda. Upon triggering a serverless function, requests are directed to the load balancer component, which selects appropriate workers based on a configured algorithm to serve the incoming requests. Nginx, the software load balancer, is responsible for orchestrating function scheduling in the OpenLambda system.

Notably, OpenLambda supports both Docker containers and lightweight SOCK container system [140]. Additionally, OpenLambda offers interaction through CLI and API interfaces, allowing seamless coordination with diverse backends. The platform also integrates external monitoring tools for enhanced observability and performance analysis.

3.2.5 Overview of Knative

Selecting the Knative serverless framework over other open-source alternatives presents a compelling choice for enhancing modern application development practices. Knative stands out with its seamless integration into Kubernetes environments, allowing you to leverage your existing infrastructure investments while embracing serverless capabilities. This compatibility ensures a smoother transition and reduces the need for extensive rearchitecting. Knative's architecture provides developers with a familiar toolkit and coherent experience, as it harnesses Kubernetes-native tools for creating, deploying, and managing serverless applications. The platform's customizable auto-scaling mechanism ensures efficient resource utilization, automatically adjusting to workload changes for optimal performance. The ability to scale functions down to zero not only maximizes resource efficiency but also minimizes costs by charging only for actual resource consumption. Additionally, Knative's vibrant community fosters rapid innovation, resulting in the continuous development of features, integrations, and extensions that enhance the platform's versatility.

To establish a comprehensive foundation for understanding Knative's inner workings, a closer exploration of Docker, containerd, and Kubernetes becomes imperative.

Docker and Containerd

Docker serves as a potent tool for constructing, deploying, and overseeing applications within containers. Containers, being lightweight and self-contained executable packages, encapsulate all necessary components like code, runtime, system tools, and libraries essential for running an application. By doing so, Docker substantially streamlines the development and deployment process, eradicating compatibility issues across varying environments. Operating on a client-server model, Docker employs communication between the Docker client and the Docker daemon, responsible for container management. This interaction occurs via a REST API, enabling developers to seamlessly interact with the daemon and execute various container operations.

Central to Docker's architecture is *Containerd* [141], a foundational container runtime that plays a pivotal role in overseeing the execution of containers. Serving as the vital interface between Docker and the underlying system, containerd ensures the seamless orchestration of containers, encompassing everything from image management to execution. Containerd takes charge of managing container lifecycle events, distributing images, and handling runtime tasks. This essential component operates in tandem with containerd-shim, which acts as an intermediary layer responsible for setting up namespaces, configuring filesystems, managing networking, and imposing resource constraints. In this role, containerd-shim ensures that each container operates within its isolated environment, safeguarding isolation and security. This arrangement prevents any potential interference with other containers or the host system. Notably, the communication channel between Docker and containerd is facilitated by the gRPC framework [142], allowing for efficient and standardized data exchange, thus contributing to the robustness and reliability of the container ecosystem.

Finally, runc [143], a fundamental element within the Docker ecosystem, plays a pivotal role in the closing stages of the container execution process. Operating as a container runtime, runc adheres closely to the Open Container Initiative (OCI) [144] standards, which ensure uniformity and compatibility across diverse container runtimes. Runc is responsible for orchestrating the essential low-level functionalities required to initiate containers, in an isolated environment, encompassing tasks such as establishing namespace isolation, configuring control groups, and managing file system mounts.

Figure 3.2.4 demonstrates the aforementioned container tools stack .

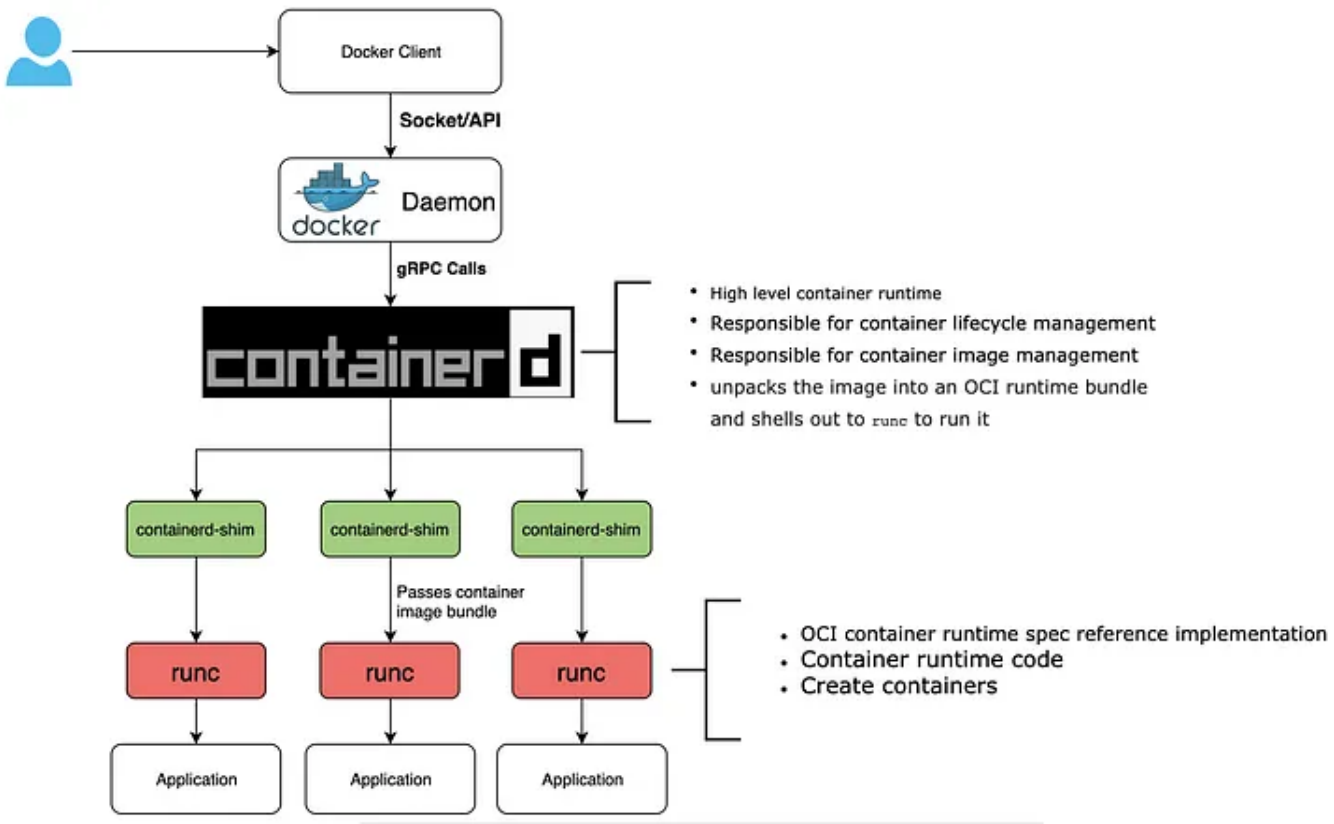


Figure 3.2.4: Container ecosystem stack [145]

Kubernetes

Kubernetes, often abbreviated as K8s, is a powerful open-source container orchestration platform that revolutionizes the deployment, scaling, and management of containerized applications. While Docker and containerd excel at encapsulating applications and managing their runtime environments, Kubernetes focuses on automating the deployment and scaling of these containers across a cluster of machines. This orchestration capability proves immensely beneficial as applications become more complex, involving numerous interconnected microservices that require efficient distribution, scaling, and fault tolerance.

Kubernetes introduces the concept of "pods," which are the smallest deployable units and can host one or more containers. It abstracts away the underlying infrastructure, allowing developers to specify the desired state of their application in terms of desired replicas, resource requirements, networking policies, and more. Kubernetes then ensures that the actual state matches the desired state by automatically scaling, distributing, and managing containers based on the defined specifications.

Kubernetes operates on a robust architecture, with multiple components [146], designed to manage the deployment, scaling, and operation of containerized applications across clusters of hosts. At its core is the Kubernetes Master, which acts as the control plane governing the cluster. The Master includes components such as the API server, responsible for handling API requests and orchestrating communication; the etcd datastore, which stores configuration data and provides consistency; the controller manager, which regulates the state of the system; and the scheduler, which assigns work to available worker nodes based on resource requirements and constraints.

On the worker nodes, the Kubernetes Node Agent, called kubelet, ensures the node's health and manages the communication between the Master and the node. Each node runs multiple Pods, with each Pod hosting one or more containers sharing network and storage resources. Container runtimes like CRI-O [147], containerd, or others execute these containers within the Pods. Kubernetes also employs the kube-proxy to maintain network rules allowing Pods to communicate with each other and with external networks.

Kubernetes' architecture promotes high availability, scalability, and fault tolerance. By distributing workloads across worker nodes and automatically rescheduling failed Pods, it ensures applications remain operational even in the presence of node failures. This architecture, coupled with its declarative configuration approach and support for diverse container runtimes, solidifies Kubernetes as a powerful platform for orchestrating and managing containerized applications at scale. Figure 3.2.5 illustrates the architecture of Kubernetes.

Knative Architecture

As outlined above, Knative builds upon Kubernetes and offers a comprehensive architecture to facilitate the deployment, management, and execution of modern serverless workloads. Comprising an array of interconnected components, Knative seamlessly extends Kubernetes' capabilities into the realm of serverless computing. At its core, Knative encompasses two essential components: Serving and Eventing.

- **Serving Component**: The Serving component [130] serves as the bedrock for deploying and overseeing serverless applications. It introduces distinct concepts such as *Revisions* [149] and *Routes* [150] to empower developers. *Revisions* capture the immutable snapshots of deployed application instances, providing a versioned history that supports easy rollbacks. Meanwhile, *Routes* facilitate traffic management by acting as intelligent routers that direct requests to the appropriate *Revisions* based on criteria like percentage splits or HTTP conditions. Figure 3.2.6 demonstrates the *Revisions-Routes* concept. Furthermore, the Serving component proposes an autoscaling model (also known as KPA [36]), pivotal for modern application architectures. This dynamic scaling feature, serves as a safeguard against the inefficiency of allocating excessive resources during periods of low or no incoming traffic. It dynamically adjusts the number of running instances based on real-time demand, ensuring that only the necessary containers are

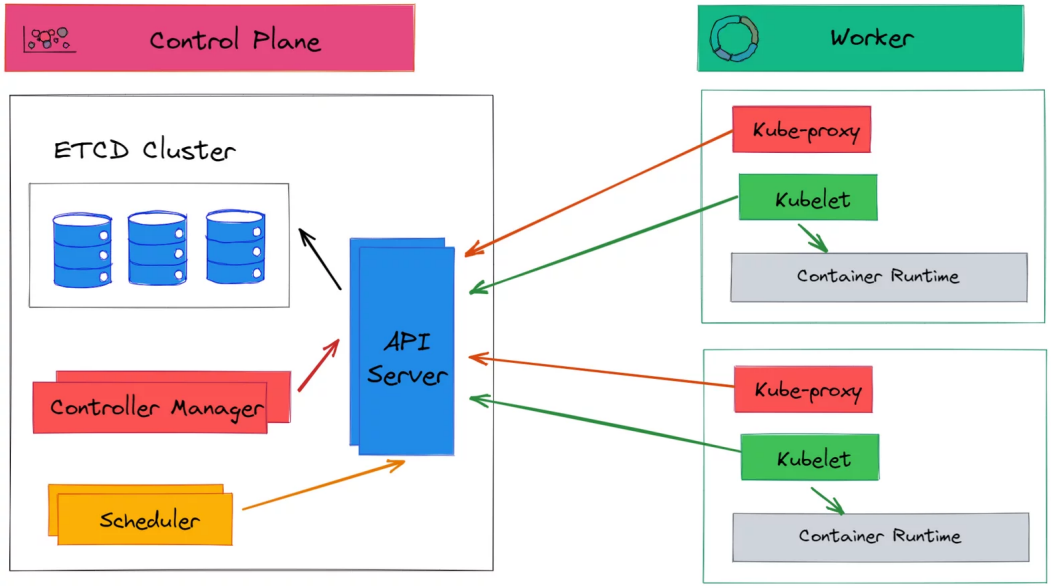


Figure 3.2.5: Kubernetes Architecture [148]

active. Notably, this can lead to the *scaling down* [151] of instances to an efficient zero state when no requests are pending, drastically minimizing resource consumption and associated costs. This meticulous orchestration of resources not only enhances operational efficiency but also aligns seamlessly with the principles of cost-effectiveness, making Knative an ideal choice to optimize resource utilization in a serverless environment.

- **Eventing component**: The Eventing component [131] is a foundational component within Knative's architecture, enabling developers to build highly responsive and event-driven applications with ease. At its core, this component focuses on the efficient management of events, which can encompass a wide range of triggers, from HTTP requests to messages from various sources. Knative's Eventing system is designed to simplify the orchestration of event production, consumption, and routing, facilitating the creation of dynamic applications. Within Knative, Event Producers are responsible for generating events. These producers can vary widely, encompassing GitHub repositories, cloud storage buckets, message brokers like Apache Kafka, and even custom. Each event producer is supported by an Event Source [152], that acts as a link between an event producer and an event sink [153]. For example Apache Kafka is supported by KafkaSource [154], which then sends the events-messages to a configured sink. This extensive support for different event producers ensures that Knative can seamlessly integrate with a wide spectrum of systems and services, allowing developers to create event-driven workflows that suit their unique requirements. Sinks on the other hand, can be a Knattive Service [155], a Channel [156] or a Broker [157]. Knative Services are the applications, deployed in the Knative ecosystem via a configuration file, e.g. a YAML file [35]. Channels serve as intermediaries for event distribution. They act as event buses where produced events are deposited and can subsequently be retrieved by interested consumers. Subscriptions [158] complement this by specifying which events from a Channel a particular service or function should consume. Brokers also serve as intermediaries for routing events, but are more generic and can be part of larger messaging systems. It is worth mentioning that Triggers [159], that subscribed to a broker, are used to route events from him to a Target, e.g. a Knative Service. Knative Eventing further provides a robust set of building blocks for constructing complex event-driven workflows. Sequences and parallel flows [160] allow developers to define both sequential and parallel sets of actions triggered by specific events, while Triggers offer a declarative approach for specifying which events should initiate particular actions. These abstractions simplify the composition of sophisticated event-driven applications,

providing flexibility in defining the flow of events and actions.

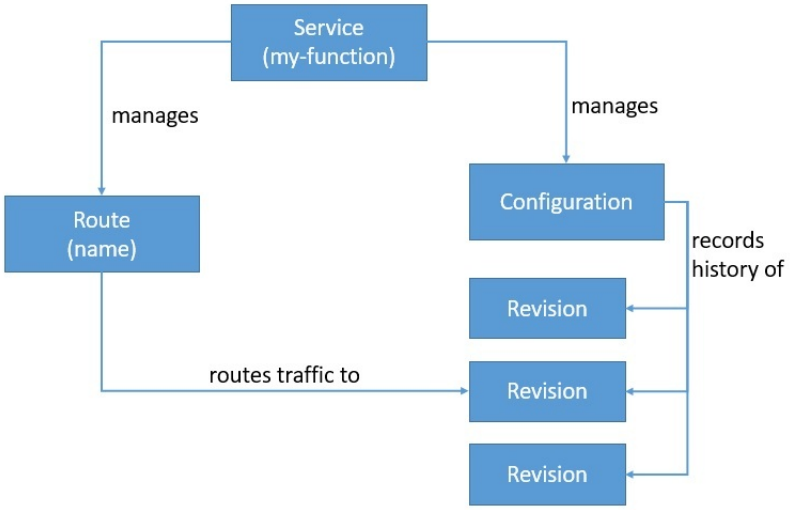


Figure 3.2.6: Knative Revision Routing [161]

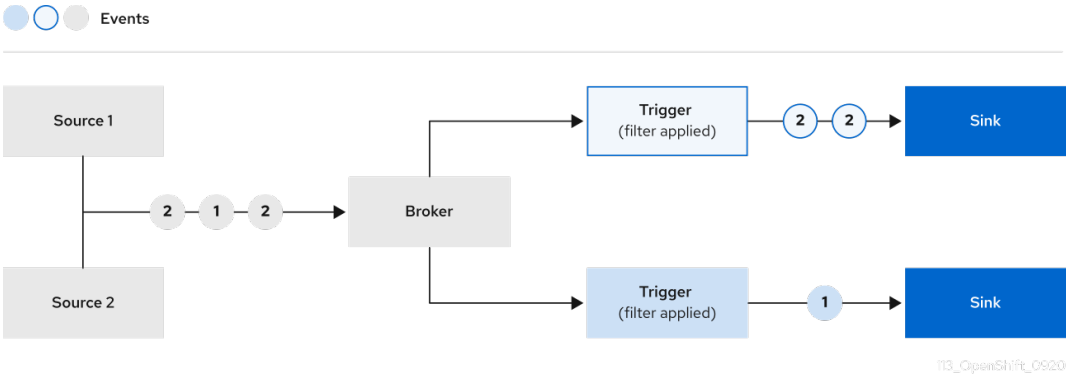


Figure 3.2.7: Knative Workflow using Channel and Subscriptions [157]

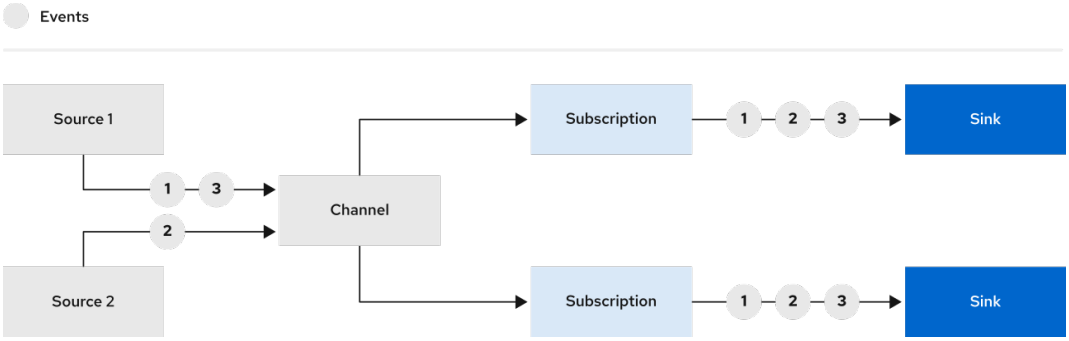


Figure 3.2.8: Knative workflow using Broker and triggers [156]

Knative, with its versatile architecture and robust event-driven capabilities, stands as a pivotal component in the world of modern serverless computing. Its foundational features, such as eventing, seamless scalability, and container orchestration, empower developers to craft dynamic applications

that respond swiftly to changing demands. Beyond its fundamental offerings, Knative's adaptability extends to specialized use cases, including running deep learning applications like deep neural networks (DNNs). Leveraging the platform's dynamic scaling and resource allocation, organizations can efficiently deploy and manage resource-intensive DNN workloads, optimizing inference processes and delivering sophisticated AI-driven solutions. Knative's role in the serverless ecosystem transcends conventional boundaries, offering a versatile foundation for a wide array of applications, from event-driven microservices to cutting-edge deep learning frameworks.

3.3 Edge Computing

Edge computing represents a paradigm shift in the world of information technology, fundamentally altering how we process, store, and analyze data. As Cao et al. [162] explains, in a digital landscape marked by the explosive growth of connected devices, IoT sensors, and real-time applications, traditional cloud computing models face limitations in addressing the demands of low latency, high bandwidth, and data privacy. Edge computing, born out of the need to overcome these constraints, offers a decentralized approach to computation. At its core, it involves deploying computing resources, including servers, gateways, and data centers, closer to the data sources and endpoints. This proximity to data generation points significantly reduces the time it takes for data to travel to distant cloud servers and back, resulting in lower latency and faster response times. Edge computing is not a one-size-fits-all solution; instead, it is a flexible and scalable architecture that can be customized to meet the specific needs of diverse applications. It finds applications in various domains, from autonomous vehicles and smart cities to healthcare and industrial automation, where real-time decision-making is paramount. In essence, edge computing represents a shift from the era of centralized cloud computing to a distributed model that harnesses the power of computation at the edge, promising a future where data is processed where it's needed, when it's needed.

Key Principles of Edge Computing:

Edge computing is founded on a set of core principles that redefine how we process and manage data in the digital age. Three fundamental pillars underpin edge computing: low latency, proximity to data, and distributed processing:

- **Low Network Latency**: In an era where real-time responsiveness is paramount, edge computing brings computing resources closer to the data source, drastically reducing latency. Unlike traditional cloud computing, where data travels to distant data centers for processing, edge devices process data locally or in nearby edge servers. This immediacy is critical for applications like autonomous vehicles, where split-second decisions can be a matter of life and death.
- **Proximity to Data**: Edge computing recognizes that not all data should, or can, be sent to centralized data centers. By processing data at or near the source, edge computing optimizes bandwidth usage and minimizes the risk of data bottlenecks. This is particularly valuable for devices in remote locations or with limited connectivity, as it allows them to function independently, even when disconnected from the cloud.
- **Distributed Processing**: Edge computing operates on a distributed model, decentralizing the processing workload across a network of edge devices and servers. This distributed architecture enhances fault tolerance, as failures in one part of the network don't cripple the entire system. It also aligns with the scalability requirements of modern applications, ensuring that computing resources can be dynamically allocated where and when needed.

These principles address the shortcomings of traditional cloud computing, which often struggles with latency issues due to data traveling long distances, especially in global-scale applications. Edge computing brings computation closer to the data source, mitigating latency concerns. It also reduces the strain on network infrastructure by minimizing data transfers and enhances overall system resilience through distributed processing. As we delve deeper into edge computing's applications and implications, these principles will emerge as the bedrock upon which its transformative power is built.

General Architecture of Edge Computing

As Keyan Cao et al. demonstrate [162], the general architecture of edge computing constitutes a network structure in a federated form, effectively extending cloud services to the network's edge. This is achieved through the introduction of edge devices situated between terminal devices and cloud computing resources.

This architecture ca be divided into three layers, the Terminal layer, the Edge Layer and the Cloud Layer. The following is a concise overview of the composition and roles of each layer:

- **Terminal Layer**: The Terminal layer encompasses a diverse range of devices connected to the edge network, including mobile devices and various Internet of Things (IoT) devices like sensors, smartphones, cameras, vehicles, and more. Within this layer, devices function both as data consumers and data providers. Their primary role is data sensing and collection. The emphasis in this layer is on data perception rather than computational capabilities. This results in countless devices in the Terminal layer collect various raw data, which is then transmitted to the upper layers for storage and processing.
- **Edge Layer**: The edge layer constitutes the core of the edge computing architecture and is positioned at the network's edge. It comprises widely distributed edge nodes strategically positioned between terminal devices and the cloud. Components in the edge layer include base stations, access points, routers, switches, gateways, and more. The edge layer facilitates connections with terminal devices, receiving and processing data uploaded by these devices. It then connects with the cloud layer and forwards processed data to the cloud. Due to its proximity to end-users, the edge layer is especially suited for real-time data analysis and intelligent processing. This proximity enhances efficiency and security compared to traditional cloud computing.
- **Cloud Layer**: The cloud layer represents the apex of the cloud-edge computing federation and comprises a collection of high-performance servers and storage resources, boasting robust computational and storage capabilities. This layer primarily serves as a potent data processing center, particularly suitable for tasks requiring extensive data analysis, such as routine maintenance and critical business decision support. In addition to permanent data storage from the edge computing layer, the cloud computing center can undertake analysis tasks that the edge layer may be unable to handle. It also executes processing tasks that necessitate the integration of global information. The cloud module possesses the flexibility to dynamically adjust deployment strategies and algorithms for the edge computing layer in accordance with prescribed control policies.

The aforementioned architecture of Edge Computing is illustrated in figure 3.3.1.

Use Cases and Applications of Edge Computing

Edge computing presents a plethora of real-world applications and use cases, where its advantages shine brightly. Some of the key domains benefiting from edge computing include:

- **Internet of Things (IoT)**: Internet of Things (IoT) devices constitute a diverse ecosystem, spanning from household smart thermostats to intricate industrial sensors. These devices share a common trait: they generate an overwhelming volume of data. In the context of IoT, edge computing emerges as a pivotal solution. Rather than transmitting all this data to distant cloud servers, edge devices process it locally. As Salam Hamdan et al. [164] explains, this approach carries multiple benefits. First and foremost, it slashes latency to a minimum. In applications where real-time responses are critical, such as industrial automation or smart cities, this is indispensable. Imagine a smart city's traffic management system. By employing edge nodes at intersections, data from traffic lights can be processed on-site. This allows for instant decision-making, optimizing traffic flow, and reducing congestion. Furthermore, processing data locally conserves precious bandwidth, a critical consideration in scenarios with limited connectivity. In essence, edge computing is the backbone that empowers IoT to deliver on its promises of enhanced efficiency, responsiveness, and resource conservation. Numerous architectural paradigms have

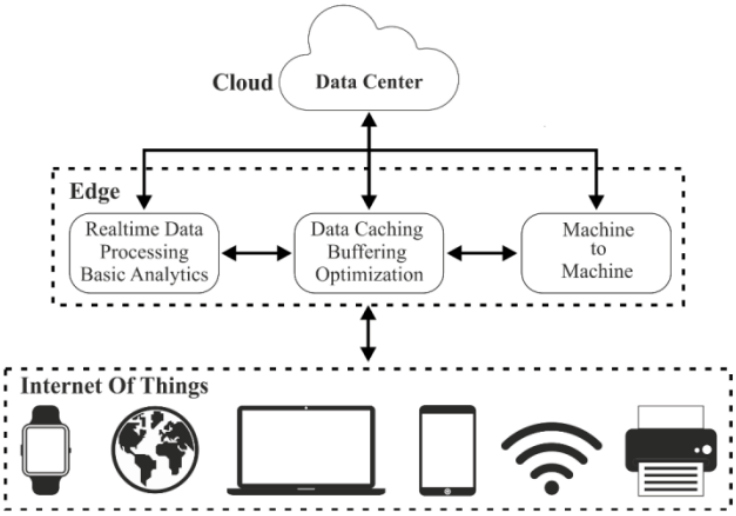


Figure 3.3.1: General Edge Computong Architecture [163]

been proposed to harness the potential of edge computing in IoT applications [164]. These range from hierarchical approaches to fully distributed models, each tailored to suit specific use cases and deployment scenarios.

- **Autonomous Vehicles**: Edge computing is a game-changer for autonomous vehicles, equipping them with split-second decision-making capabilities. These self-driving cars rely on an array of advanced sensors, cameras, and lidar systems to navigate the roads. Instead of sending all sensor data to remote data centers, edge devices within the vehicle process data in real-time [165]. This approach allows vehicles to instantly analyze their surroundings, identifying obstacles, pedestrians, and other vehicles, and react swiftly to changing traffic conditions. The result is reduced latency, enhanced privacy, and the ability to make crucial driving decisions swiftly and safely. Edge computing is the linchpin that ensures the success and safety of autonomous vehicles, marking a transformative shift in the realm of transportation.
- **Augmented Reality (AR)**: In the realm of augmented reality (AR) applications, where seamless and responsive experiences are of utmost importance, edge computing plays a key role. AR necessitates the precise alignment of virtual objects with the real world, creating immersive and interactive scenarios. Edge computing significantly reduces latency by processing data locally, ensuring that virtual overlays respond instantaneously to changes in the physical environment [166]. Consider AR glasses that offer real-time language translations or provide contextual information about landmarks as you explore a new city. With edge computing, these applications become more immersive, accurate, and engaging, as the digital and physical worlds seamlessly converge, opening up new possibilities for entertainment, education, professional use cases, and even revolutionizing fields like Remote Assistance [167] and Telemedicine [168].
- **Smart Grids**: As Cheng Fend et al. [169] point out, in the context of Smart Grids (SGs), the incessant generation of vast data volumes necessitates advanced data analytics algorithms to translate this data into actionable insights. These insights are pivotal for enhancing SG operations and services. The data analytics depend on Information and Communication Technologies (ICTs) performing a critical role in data collection, transmission, and processing [170]. Computing, a critical ICT function, underpins SG data analytics, serving as the linchpin for SG operations and services. Historically, centralized cloud computing prevailed as a dominant SG computing solution [171]. In this model, geographically distributed devices connect to cloud data centers, which make centralized decisions and issue control commands. However, this approach faces limitations, including bandwidth constraints, environmental diversity, and data privacy concerns. In response, edge computing has emerged, pushing computation from centralized nodes

to the communication network's edges. Edge Computing leverages computing resources near sensors and end-users for data analytics. This transition yields numerous advantages, including reduced system latency, lightened cloud center workload, improved scalability and availability, and fortified data security and privacy [172]. This shift towards edge computing represents a substantial leap in bolstering the efficiency and reliability of Smart Grids.

- **Healthcare**: As Rushit Dave et al. [173] mention, the healthcare sector represents another crucial domain poised to gain significant advantages from the adoption of edge computing. Edge computing holds the potential to streamline data flow, thereby enhancing overall efficiency within healthcare operations [174]. Moreover, the architectural framework of edge computing offers healthcare professionals the opportunity to reduce their reliance on remote centralized servers [175, 176] and enhanced data security and ethical integrity [177, 178, 179]. Currently, wearable devices and sensors are serving as essential tools for treating and actively monitoring patients who face conditions like Parkinson's disease, a heightened risk of heart attacks, and various severe health issues [180]. In these critical situations, edge computing can react promptly with negligible latency, substantially improving reliability and averting potential adverse events.

Edge Computing vs Cloud Computing: Challenges and opportunities

As already mentioned, in the context of data management and processing, Edge Computing and Cloud Computing represent two distinct paradigms. The following comparison, Sriram G. K. (2022) [181], aims to elucidate their contrasting attributes, applications, and associated considerations.

- **Speed and Response Time**: Edge Computing excels in terms of speed and rapid response. In today's competitive landscape, where milliseconds matter, Edge Computing reduces latency by processing data near its source. This reduced latency is invaluable, especially in sectors like finance and data-driven industries, where sluggish networks can result in substantial financial losses or customer dissatisfaction. By quickly accessing and processing user requests as they occur, Edge Computing ensures that organizations can provide rapid responses and maintain a competitive edge. Cloud Computing, while offering immense computing power, may face longer response times due to the geographical distance between its data centers and end-users. The centralized nature of Cloud Computing means that data often has to traverse long distances to reach the data center, which can introduce delays. While Cloud Computing is indispensable for tasks that require massive data processing and in-depth analysis, its response times may not meet the demands of real-time applications.

- **Data Analysis and Analytics**: Cloud Computing, with its centralized architecture, excels in providing a robust platform for data analysis and analytics. It leverages organized data centers distributed across regions, enabling it to process vast amounts of data from various sources efficiently. This is crutial for task like training a neural network. This scalability offers enhanced flexibility for Big Data management, making it an ideal choice for large enterprises dealing with complex web applications and comprehensive analytics. Cloud Computing's ability to handle data-intensive workloads and provide detailed insights sets it apart in this aspect. Edge Computing, on the other hand, processes data locally, close to its source. While this proximity allows it to deliver instantaneous results, it comes with limitations in terms of data processing capabilities. Edge devices are typically designed to handle smaller datasets efficiently. This design is well-suited for applications requiring real-time responses, such as IoT devices, but may not be suitable for complex and data-intensive analytics tasks.

- **Data Processing Capacity**: Cloud Computing boasts a plethora of data management applications and high processing capacity, making it suitable for large-scale data processing tasks. Major cloud providers like Microsoft Azure [182] and Amazon Web Services [183] offer comprehensive solutions for enterprises requiring extensive data processing capabilities. With the ability to handle vast datasets, Cloud Computing provides in-depth analysis and meaningful insights that may be unattainable with Edge Computing. Edge Computing, while delivering real-time results, faces limitations in processing substantial data. Its design, focused on low-latency, real-time

processing, means that Edge devices are often equipped to handle smaller datasets efficiently. This limited capacity can be a challenge for organizations dealing with extensive data processing needs, where Cloud Computing's superior processing power becomes a necessity.

- **Network Security**: Both Edge and Cloud Computing encounter unique cybersecurity challenges. Edge Computing presents a decentralized model that mitigates specific threats by distributing data across multiple locations. This distribution reduces the impact of network disruptions, ensuring that a disruption in one location does not affect the functioning of other networks. Furthermore, since data is processed locally at the Edge, only a portion of data is at risk in the event of a security breach. However, Edge Computing introduces its own security concerns. Edge devices, as entry points for data processing, can be vulnerable to cyberattacks, malware, and intrusions that may infect the network. Nevertheless, Edge Computing's distributed architecture allows for the implementation of security protocols that can efficiently address vulnerabilities without compromising the entire network's integrity. In contrast, Cloud Computing relies on centralized data centers that are susceptible to distributed denial of service (DDoS) attacks and major power outages. These centralized centers present attractive targets for cyber threats. Yet, they also offer the advantage of centralized security measures, which can be rigorously implemented to safeguard data.
- **Scalability, Versatility, and Reliability**: Cloud Computing stands out in terms of scalability, making it an ideal choice for large enterprises experiencing growth. It accommodates the expansion of data processing requirements, offering resources and computing power on demand. Additionally, Cloud Computing enables versatile data processing for macro data management, allowing organizations to handle a wide range of tasks efficiently. Edge Computing, conversely, excels in microdata management and is well-suited for enhancing network reliability. It facilitates expansion into local markets through partnerships with local data centers, eliminating the need for costly infrastructure investments. However, Edge Computing's resource constraints limit its scalability for large-scale data processing tasks.
- **Cost**: Cloud Computing is known for its comprehensive data management capabilities but often comes with a considerable price tag. While the services offered are valuable and scalable, the costs can escalate, especially for large data processing requirements. Expenses related to data storage and processing can deter some companies from adopting Cloud solutions. Edge Computing presents a more cost-effective alternative, particularly with the availability of affordable IoT devices and minimal additional costs. Organizations can deploy Edge devices without incurring significant overheads, making it an attractive option for cost-conscious projects. However, Edge Computing's cost-effectiveness comes with trade-offs in terms of functionality compared to Cloud Computing.
- **Standardized IoT Protocols**: Both Edge and Cloud technologies grapple with the challenge of lacking standardized IoT protocols. The absence of common standards or protocols can introduce data security concerns during data transfer or migration to Cloud infrastructure.

3.3.1 Infastructure of Edge Computing

In edge computing, a diverse array of hardware components plays a pivotal role in shaping its capabilities. These devices, ranging from IoT sensors to specialized edge servers, form the backbone of edge computing infrastructure. Some of these key devices are:

- **Internet of Things (IoT) Devices**: IoT devices serve as the bedrock of edge computing. These encompass a wide range of sensors, actuators, and smart objects that collect data from the physical world. Examples include temperature sensors, motion detectors, cameras, and even smart home appliances. IoT devices are typically resource-constrained and are designed to transmit data to edge nodes or servers for processing, making them instrumental in applications like smart homes, industrial automation, and environmental monitoring.
- **Edge Servers**: Edge servers are versatile computing devices strategically positioned at the edge

of the network. These servers act as intermediaries between IoT devices and the centralized cloud. Their processing and storage capabilities can vary significantly based on the specific application requirements. In some instances, edge servers demand substantial processing power, equipped with GPUs, FPGAs, or specialized processors, along with memory and storage to handle data-intensive tasks. Conversely, there are scenarios where the emphasis is on minimizing their footprint, making them as compact and efficient as possible. This adaptability ensures that edge servers can meet the diverse needs of edge computing applications, whether it involves resource-intensive data processing or serving as unobtrusive attachments to existing assets.

- **Edge Data Centers**: Larger-scale edge computing implementations may include edge data centers. These facilities house multiple edge servers, storage systems, and networking equipment. Edge data centers are strategically located in proximity to end-users and IoT devices to minimize latency.
- **Edge Gateways**: Edge gateways serve as aggregation points for data collected from various IoT devices. They consolidate and preprocess data before forwarding it to higher-level edge servers or cloud data centers. Edge gateways often integrate connectivity options like Wi-Fi, Bluetooth, and cellular networks, ensuring seamless communication with a variety of IoT devices. These gateways play a critical role in ensuring data efficiency and reliability in edge computing setups.
- **Single-Board Computers (SBCs)**: Single-board computers like the Raspberry Pi [28] and NVIDIA Jetson series [29] offer a compact yet potent solution for edge computing. These credit card-sized computers feature CPUs, GPUs, and various connectivity options. In addition to these, microcontrollers like STM32 [30] (STMicroelectronics) and development platforms like Arduino are also noteworthy in the world of edge computing. These devices cater to a wide range of edge computing needs, making them valuable assets for prototyping and deploying edge applications. Their popularity continues to grow, making them integral tools for projects spanning home automation, robotics, AI-powered edge devices, and more.
- **Industrial PCs**: In industrial settings, ruggedized industrial PCs are often deployed at the edge. These devices are designed to withstand harsh environmental conditions, making them ideal for manufacturing and process automation. They handle tasks like real-time monitoring, quality control, and predictive maintenance.

In summary, Edge computing represents a transformative paradigm shift in the world of computing. Unlike traditional cloud computing, which centralizes data processing in distant data centers, edge computing brings computation closer to where data is generated, often at the network's periphery. This approach offers several key advantages. It drastically reduces latency, enabling real-time decision-making critical for applications like autonomous vehicles and industrial automation. It enhances data privacy and security by keeping sensitive information localized. Edge computing is highly versatile, finding applications in various domains, from IoT to augmented reality. In the context of IoT, it allows for rapid data analysis and decision-making at the source, optimizing efficiency and enabling faster response times. However, edge computing faces challenges, such as defining the processing capabilities of edge servers, to ensure they meet specific application requirements. Nonetheless, this technology is revolutionizing how we handle data, making it faster, more secure, and adaptable to diverse scenarios, and it is poised to play an increasingly vital role in our connected world.

Chapter 4

Serverless Framework and Run-Time Scheduler

This chapter, We delve into the framework and scheduling mechanism central to our dissertation, providing a foundational understanding of our research approach.

Problem Definition

In the realm of contemporary computing, the efficient execution of neural networks across a diverse and intricate array of environments presents a formidable challenge. This challenge revolves around orchestrating the seamless execution of neural network layers while harnessing the full potential of available hardware resources. As these neural networks grow in complexity and sophistication, addressing this challenge becomes increasingly crucial. The demand arises for an innovative solution capable of optimizing the execution process by intelligently dispatching neural network layers across a heterogeneous computing cluster. This entails dynamically allocating tasks to the most suitable hardware context, whether it be CPU or GPU, while considering the unique characteristics of each layer. Additionally, this solution must provide the flexibility to cater to a wide spectrum of neural network architectures and configurations. The goal is to streamline the execution process, reduce latency, under strict SLAs, enhance resource utilization, and ultimately, unleash the full capabilities of neural networks in practical applications.

Furthermore, an integral aspect of this challenge lies in the dynamic scheduling of neural network layers, a complex task that requires careful consideration of each layer's requirements and the available hardware context. The scheduling problem adds an additional layer of complexity, demanding an intelligent approach to determine where and how to execute each layer optimally.

This dissertation explores and presents precisely such a solution—an optimized deployment framework that leverages Kubernetes, Knative, Deep Learning, and more specifically Reinrocement Learning, and other technologies to address the intricate interplay between efficient execution and dynamic scheduling in neural network deployments.

4.1 Serverless Framework for layered and full offloading of DNNs

As already mentioned, this framework leverages the robust infrastructure provided by Kubernetes and Knative to create a highly adaptable environment for neural network execution. Kubernetes, known for its efficient container orchestration and management, forms the foundation upon which this

framework builds. It facilitates the deployment and scaling of neural network layers across a distributed cluster of devices. In parallel, Knative extends Kubernetes by introducing a serverless framework for deploying and managing containerized applications, ensuring efficient resource allocation and seamless communication between services.

Together, these platforms lay the groundwork for a versatile and scalable system capable of handling neural network execution tasks with ease. The framework streamlines the complexities of multi-device, multi-context execution, offering support for both CPU and GPU environments. This architecture empowers users to optimize resource utilization, adapt to various execution scenarios, and effortlessly manage the intricate nuances of deploying neural networks across diverse computing clusters.

Framework Core: Execution Service

4.1.1 Layer Execution Service

At the heart of the framework lies a service responsible for executing individual neural network layers. This service is implemented exclusively in Python and serves as a critical component for handling the execution of layers in response to incoming CloudEvents [184]. The sequence of actions and the inner workings of the service are described below:

1. **Receiving and Processing CloudEvents**: The service operates as an event listener, awaiting incoming CloudEvents conveyed through HTTP requests. To handle these events, it leverages Flask [31], a popular Python web framework that establishes a REST API for the seamless reception and processing of CloudEvents. Each incoming CloudEvent carries crucial information essential for layer execution. This includes serialized input data in the form of a byte array, data shape, numerical format (e.g., float16) (needed for deserialization), the specific target layer, the server's IP address and the scheduling services port, hosted in the server.

2. **Multithreaded Execution**: Upon receiving a CloudEvent, the service spawns a dedicated thread to handle its processing. This multithreaded approach ensures that multiple requests can be served concurrently, optimizing resource utilization.

3. **Data Preprocessing**: Before executing the requested layer, the, dedicated to this CloudEvent, thread conducts essential data preprocessing tasks. It reshapes and casts the raw input data to match the specified shape and data type. This step utilizes the powerful capabilities of the NumPy package [32], a fundamental library for numerical computing in Python.

4. **Layer Execution**: Once the data is appropriately preprocessed, the dedicated thread proceeds to execute the targeted neural network layer. PyTorch [33], a deep learning framework, is employed for this purpose, ensuring efficient and accurate execution.

5. **Scheduling Next Layer**: After successfully executing the current layer, the thread determines the location for the next layer's execution. To make this decision, it communicates with the scheduling service. The scheduling service provides the next device to continue layer execution based on resource availability and optimization strategies (more in scheduling section).

6. **Inter-Device Communication**: If the scheduling service designates the same device for executing the next layer (local execution), the same thread seamlessly continues the execution process. However, if the next layer is assigned to a different device, the thread retrieves the shape and data type of output data, serializes them, creates a new CloudEvent and dispatches it to the designated device. In such a scenario, the destination device follows the same sequence of actions.

The described sequence of actions is illustrated in figure 4.1.1. Moreover, the *Extract data from CloudEvent and Reconstruct Input* block is explained in figure 4.1.2 and the *Serialize Output and get shape, data type & Create and Send Next CloudEvent* blocks are explained in figure 4.1.3.

In the previous steps, there has been no mention of the initialization of the neural network layers. However, this critical step is addressed through a preparatory technique known as *warmup*. When the

source code is invoked for the first time, warmup comes into play. This warmup process encompasses the loading and execution of the neural network layers, and its significance lies in priming the service for optimal subsequent performance. During warmup, the service loads the required model layers and conducts multiple execution cycles, typically around five times or as dictated by configuration. This enables the service to efficiently initialize and allocate resources for model execution, ensuring that subsequent layer executions encounter minimal delays. This initialization phase, often referred to as a *cold start*, signifies the challenges associated with the initial execution of a service in dynamic computing environments. During a cold start, various setup and initialization tasks can introduce latency, potentially impacting the responsiveness of the service. Warmup effectively preempts these cold start latencies by conducting multiple execution cycles.

Ultimately, it is worth to mention the reason why to use CloudEvents. CloudEvents serve as a pivotal component within the service infrastructure for several compelling reasons. Firstly, they provide a standardized and universally recognized format for event data, ensuring consistency and compatibility across diverse components and services within the system. This standardization simplifies event handling, reducing the intricacy of interpreting and processing incoming events. CloudEvents are not only easy to work with but also highly flexible, supporting various transport protocols like HTTP, which is ubiquitous in web communications. This flexibility makes it seamless to transmit events across the network, enabling efficient communication between different services. Moreover, CloudEvents can be extended with custom attributes, allowing for the inclusion of additional context or metadata specific to the event, enhancing its informativeness. Lastly, CloudEvents enjoy robust support within a growing ecosystem of tools, libraries, and platforms, enabling seamless integration with event brokers, serverless frameworks (like Knative), and event-driven architectures, thus further elevating the service's capabilities and its capacity to interoperate with diverse technologies and environments.

4.1.2 Whole Neural Network Execution Service

Additionally, one more service is implemented within our framework that follows the same core logic as the one previously discussed. However, it's important to note a key difference between these two services. While the first service quereis the server to specify where to execute the next layer of a neural network, the second service streamlines the process by executing the entire neural network. In other words, the second service automates the neural network execution process, eliminating the need for communication between the service and the server. The Flowchart of this service is shown in figure 4.1.4.

4.1.3 Deployment of Services

Once the service code is developed and optimized for neural network layer execution, the next critical step is the deployment of these services within the Kubernetes-Knative cluster. The deployment process is designed to ensure seamless scalability, efficient resource allocation, and responsiveness to varying workloads. Below, we outline the key steps and configurations involved in this deployment phase:

1. **Cluster Setup**: The foundation of our deployment begins with the establishment of the Kubernetes-Knative cluster. This involves utilizing kubeadm CLI [34] to add devices to the cluster, effectively integrating them into the computing environment. Furthermore, the setup process involves a plethora of complexities, such as configuring essential elements like Flannel for networking to facilitate smooth intra-cluster communication, and setting up Knative for streamlined serverless application management, providing effective resource allocation and auto-scaling functionalities to our framework. Moreover, it's essential to emphasize that within the cluster, the NVIDIA device plugin for Kubernetes [47] must be deployed. This plugin serves the critical functions of exposing the quantity of GPUs on each node within the cluster, monitoring their health, and enabling the execution of containers that require GPU execution context in the Kubernetes cluster. This cluster setup lays the groundwork for orchestrating the execution of neural network layers across heterogeneous hardware.

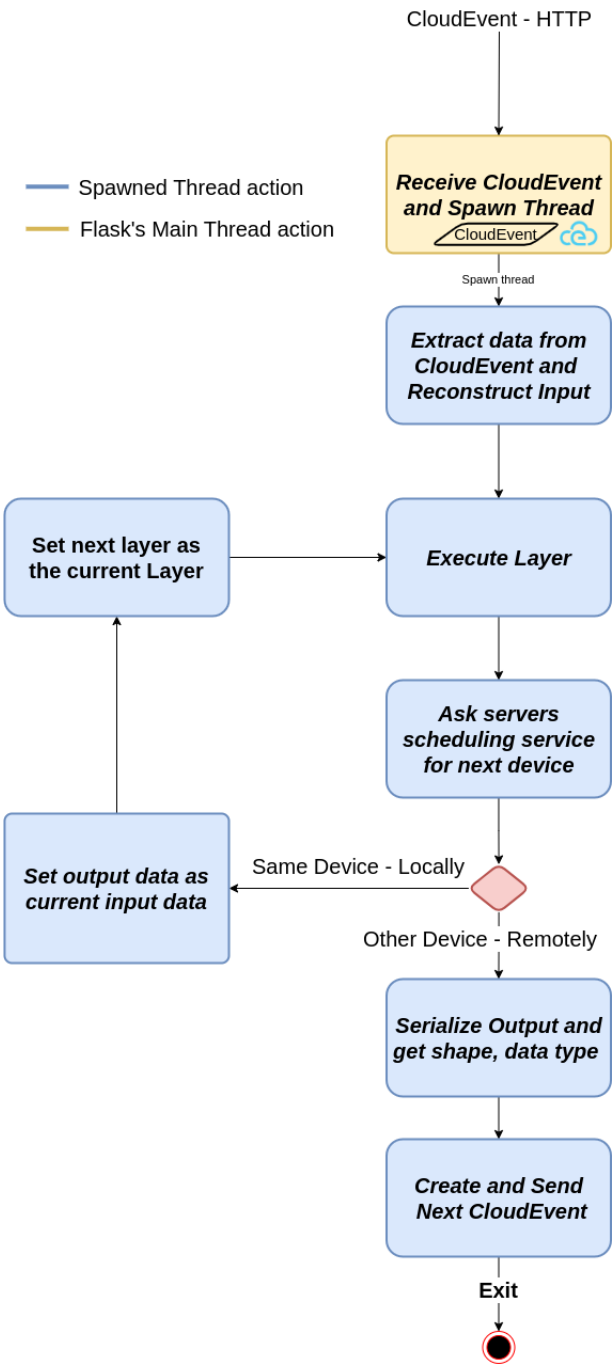


Figure 4.1.1: Layer Execution Service Flowchart

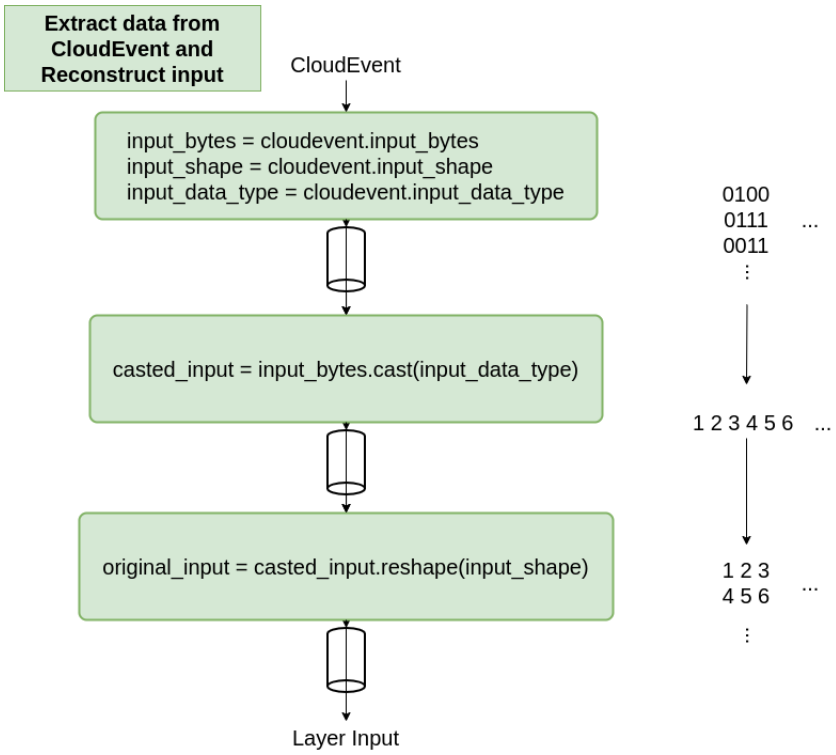


Figure 4.1.2: Extract data and Reconstruct input steps

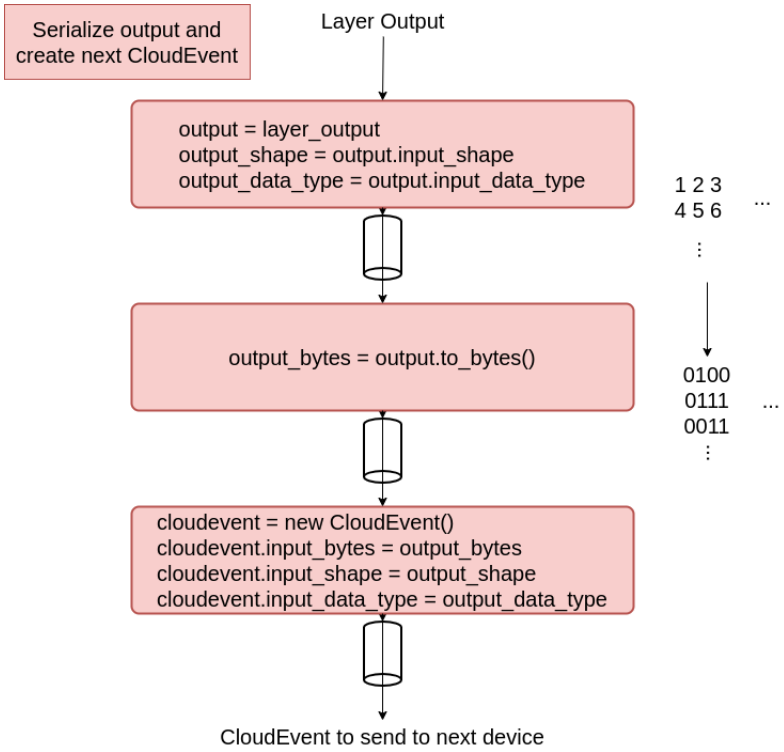


Figure 4.1.3: Serialize output and create next CloudEvent steps

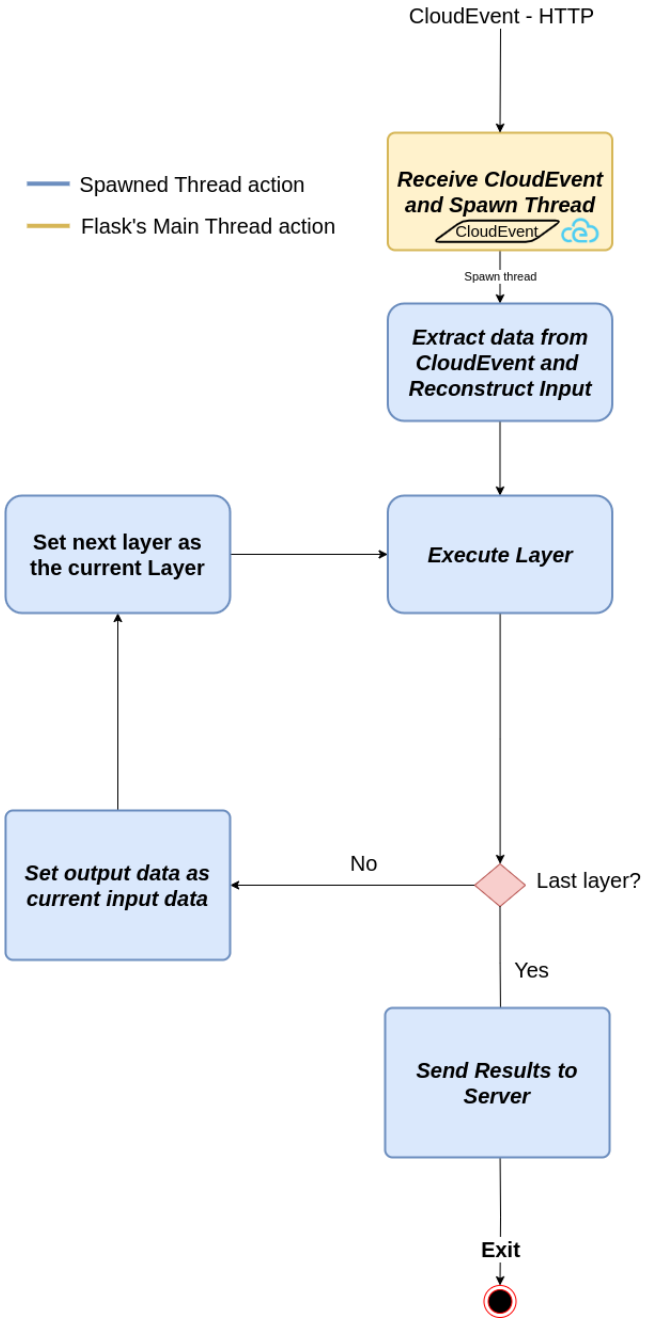


Figure 4.1.4: Whole Neural Network Execution Service Flowchart

2. **Service Docker Images**: In order to deploy the images to the created cluster, the service must be containerized. The source code, explained in the previous section, is encapsulated within Docker images. These images act as self-contained units, bundling not only the service code but also its dependencies. Once crafted, these images are puhsed on DockerHub, facilitating easy access and deployment across the cluster. It is worth to mention that the base image (the image that on top of it the services images are built) is the l4t-pytorch:r35.2.1-pth2.0-py3, pulled from the Nvidias NGC Catalog [185]. This image provides GPU support for the Jetson devices (more in evaluation section) that are used in this dissertation. More specifically contains essential packages and tools like PyTorch, Cuda Toolkit which allows PyTorch to run efficiently on NVIDIA GPUs, cuDNN, an optimized GPU library for deep neural networks that significantly speeds up the training and inference of deep learning models, Nvidia GPU Drivers required to interface with NVIDIA GPUs and other common Data Science libraries like NumPy, SciPy, and scikit-learn.

3. **YAML Configuration Files**: The orchestration of the services within the cluster is driven by YAML configuration files. Each YAML file serves as a blueprint, defining how a specific service should be instantiated and maintained. These files encompass a multitude of configurations, specifying resource requirements, environmental variables, deployment constraints, and scaling parameters. They play a pivotal role in ensuring each service's efficient operation within the cluster. Some examples of the configurations that a YAML file define are:

    - **Environmental Variables**: Several environmental variables are defined. These variables serve as a communication bridge, enabling seamless interaction between the service and the broader cluster environment. As an illustration, the environmental variable MODEL-FOLDER-NAME designates the models (such as Mobilenetv2) directory location within the container that holds his layers, which are intended to be loaded by the service.

    - **Node Affinity**: To achieve granular control over service placement, node affinity is configured within the YAML files. This affords the flexibility to explicitly designate which device within the cluster should host a particular service. Such fine-grained control is invaluable, especially in scenarios where optimization and efficient scheduling are imperative. Figure 4.1.5b illustrates an example configuration of Node Affinity in a YAML file.

    - **Concurrency configuration**: Each pod-service is configured to allow multiple threads to run concurrently within a service. For instance, setting concurrency to two threads per pod enables concurrent execution of neural network layers. However, it's essential to strike a balance, as excessively increasing concurrency can introduce latency due to Python's Global Interpreter Lock (GIL) [186]. Figure 4.1.5a illustrates an example configuration of concurrency in a YAML file.

    - **Scale Down Delay Configuration**: This configuration parameter determines the delay before a service pod is scaled down after it becomes idle. It can be set to a specific duration, such as 10 minutes, which means that if a pod remains idle for 10 minutes, it becomes a candidate for termination during the next scaling decision. This parameter helps balance resource optimization with responsiveness to incoming requests. Figure 4.1.5a illustrates an example configuration of Scale Down Delay in a YAML file. It is worth to mention that Scale Down Delay configuration helps in mitigating cold starts. A cold start refers to the process of initializing a container or pod, resulting in slower response times, as it involves launching the container and loading its dependencies (in this case loading the model too), configurations, and application code. So by keeping a pod alive for a specified duration minimizes the effects of cold starts on service performance, ensuring consistently responsive behavior.

4. **Autoscaling with KPA**: The Knative Pod Autoscaler (KPA) [36] is a critical component of our deployment strategy. It brings dynamic scalability to the forefront. KPA intelligently monitors the cluster's workloads and automatically adjusts the number of pods based on demand. This adaptive scaling mechanism optimizes resource utilization while ensuring the cluster remains

```yaml
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/scale-down-delay: "10m"
        autoscaling.knative.dev/target: "2"
```

(a) Concurrency and Scale Down Delay example Configurations

```yaml
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: device
          operator: In
          values:
            - agx-xavier-00
```

(b) Node affinity example configuration

responsive to varying workloads. Notably, KPA offers the unique capability of scaling pods down to zero, a feature that proves highly advantageous in resource-constrained edge computing environments.

4.1.4 General Architecture with scheduler

On top of this framework a centralized scheduler can determine where a neural network or a layer of it will be executed in a device, triggered by an incoming request. This generalized architecture is shown in figure 4.1.6.

4.2 Reinforcement Learning based Scheduler

The framework's orchestration extends beyond deployment, encompassing resource scheduling where reinforcement learning techniques come into play. In this phase, the framework adapts to diverse edge computing scenarios. Reinforcement learning is the foundation, enabling our framework to make informed decisions regarding neural network layer or whole neural network allocation. It's more than scheduling; it's a centralized orchestrator that continually assesses the environment, assigns resources, and ensures neural network execution. This dynamic scheduler is a cornerstone of our framework's versatility, navigating the ever-changing terrain of edge computing and ensuring responsive execution across various computing environments.

As previously mentioned, the fundamentals of Reinforcement Learning (RL) revolve around an agent, an environment, and a system of rewards. RL is fundamentally about the agent learning to make a sequence of decisions over time in an environment to maximize cumulative rewards. The agent interacts with the environment by taking actions and receiving feedback in the form of rewards or penalties. The objective is for the agent to learn a policy that maps states to actions in a way that maximizes its expected cumulative reward. This iterative learning process, driven by the pursuit of favorable outcomes, forms the essence of RL and underpins its application in various decision-making scenarios. Following these fundamentals, the primary goal of employing RL techniques in this dissertation is to develop a centralized dynamic and adaptive resource scheduling mechanism for orchestrating neural-network execution within an edge computing environment. This RL-driven scheduler aims to intelligently allocate resources to neural networks and neural network layers in real-time. The overarching objective is to optimize resource utilization, minimize latency, minimize energy, and ensure efficient execution of neural networks, ultimately enhancing the responsiveness and

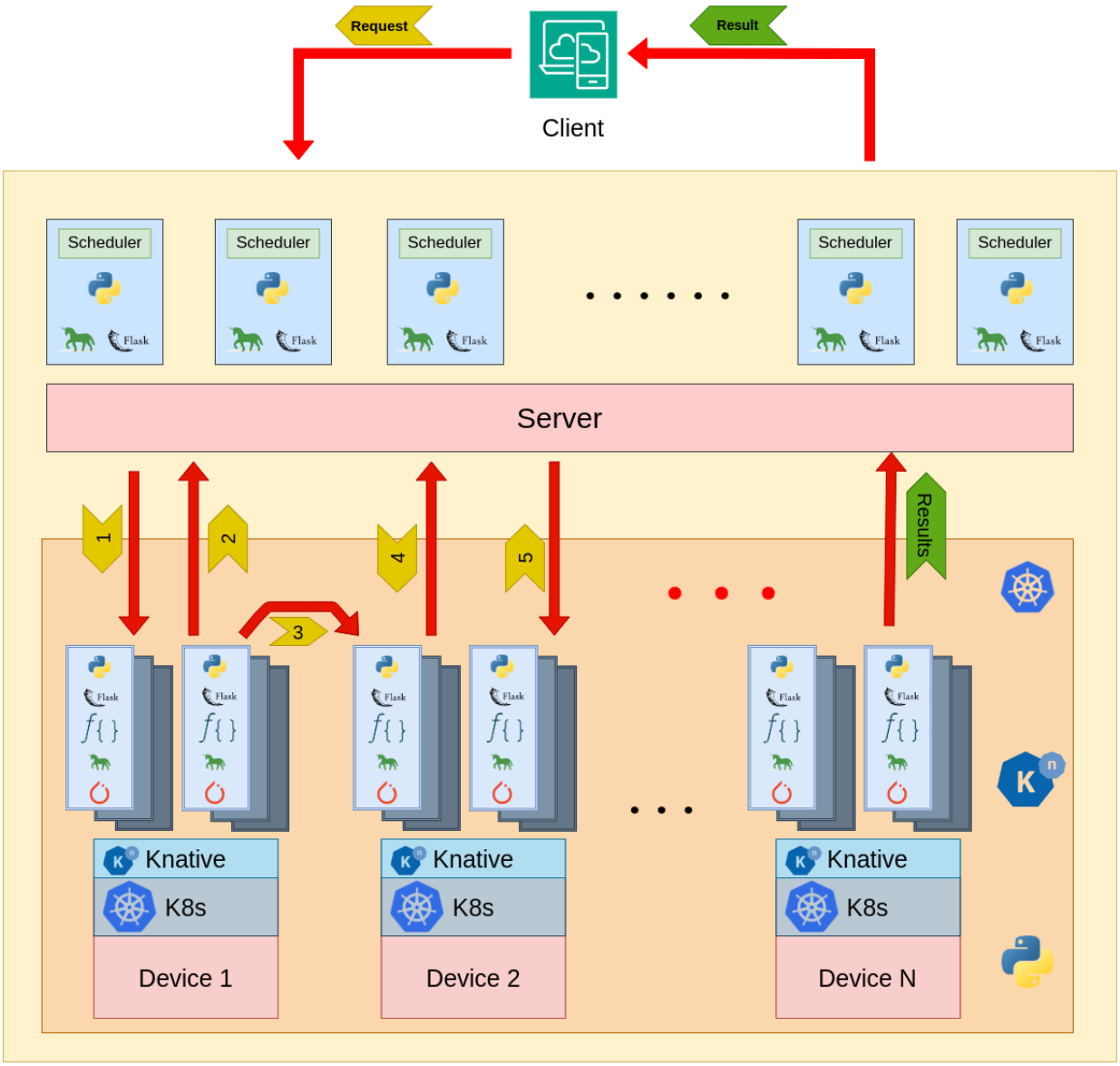


Figure 4.1.6: Whole Neural Network Execution Service Flowchart

adaptability of the framework across diverse edge computing scenarios.

To achieve this goal, several crucial components and parameters must be defined and integrated into the RL-based scheduling mechanism. These include defining the Observation Space (or State Space), Action Space, Reward structure and the RL agent architecture itself. Each element plays a pivotal role in shaping the scheduler's decision-making process, enabling it to effectively adapt to the dynamic demands of neural networks execution in edge computing environments. It is worth to mention that the implementation of the Reinforcement Learning algorithm is accomplished by using OpenAIs GYM [102] and OpenAIs StableBaselines3 [106].

Observation State Definition

The observation state plays a pivotal role in the efficiency and effectiveness of our reinforcement learning (RL)-based scheduling framework. Through the Observation State, the RL agent perceives the environment and makes decisions. Comprising a diverse array of data and parameters, the observation state forms the foundation for intelligent resource allocation.

Central to the efficiency and descriptiveness of the Observation State is a comprehensive approach to monitoring and data collection. In this regard, We employ two command-line interfaces (CLIs), namely tegrastats and perf, to capture performance metrics from the computing environment (e.g. computing devices). These tools are essential components of the monitoring strategy, enabling us to gain real-time insights into resource utilization, hardware behavior, and execution efficiency.

Moreover, the approach to Observation Space extends beyond mere data collection during runtime. We recognize the critical importance of offline profiling to understand the intrinsic performance characteristics of neural networks. To this end, We divide Observation Space into two distinct realms: the online phase and the offline phase. The online phase focuses on real-time monitoring, where dynamic decisions are made, while the offline phase involves in-depth layer and whole neural-network profiling, ensuring a holistic understanding of layer and neural-network behavior across diverse hardware contexts. This dual-phase Observation Space is the foundation upon which the RL-based scheduling algorithm operates, allowing us to make intelligent, context-aware decisions for efficient neural network execution.

Online Phase - Runtime Monitoring

- **Perf CLI**: During the online phase, where real-time decisions are made, We leverage the perf CLI to gather critical performance metrics, from each device. This includes:
  - **IPC (Instructions Per Cycle)**: Measuring the efficiency of CPU instruction execution.
  - **Cache Misses**: Assessing memory access patterns.
  - **Context Switches**: Tracking context switches to manage CPU resources efficiently.
  - **Page Faults**: Monitoring memory management.
- **Tegrastats CLI (Jetson-specific)**: In Jetson platforms, We also harness the tegrastats CLI to capture a spectrum of performance indicators, from each device, encompassing:
  - **Power Consumption**: Measuring power usage for energy-efficient scheduling.
  - **RAM Utilization**: Monitoring memory usage for optimal resource allocation.
  - **CPU Utilization (Mean and Standard Deviation)**: Analyzing CPU load across multiple cores to ensure balanced workloads.
  - **CPU Frequency (Mean and Standard Deviation)**: Tracking CPU clock speeds for dynamic allocation.
  - **GPU Utilization**: Evaluating GPU workload for efficient offloading of tasks.
  - **GPU Frequency**: Monitoring GPU clock speeds for optimal performance.

- **Devices current power modes**: The current power modes of devices are included in the input data provided to the RL agent.

- **Previous Layer execution Device**: In the case of layer execution, the device where the previous layer was executed is included too. This information helps reduce network latency by enabling the RL agent to make informed choices regarding the next layer's placement.

Offline Phase - Profiling

- **Layer and DNN Profiling**: The offline phase involves layer and DNN profiling, where We assess the performance of individual layers within neural networks. During profiling, the same performance measurements are extracted as in the runtime online phase.

- **CPU and GPU context**: Profiling is conducted in both CPU and GPU contexts, providing insights into how different layers and DNNs perform on diverse hardware components.

- **Layer Profiling Across Power Modes**: Profiling isn't limited to a single power mode; instead, it spans across all available power modes of the given devices. This comprehensive approach ensures that layer performance is assessed under various operational conditions. Profiling in different power modes provides insights into how layers and DNNs behave in scenarios of varying power consumption and performance capabilities, allowing the framework to adapt intelligently to the specific characteristics of each device.

  This multi-dimensional assessment (in execution context {CPU, GPU} and in different power modes of devices) is a key element in the strategy for efficient and adaptable neural network execution.

In a mathematical manner, Assuming that the number of devices equals N, the Observation State for a random layer of a random neural network or simply for a random neural network, that is profiled in each device, in each power mode in each context, would be:

For each device D_i where $i \in 0...N$ the current state vector, given from the online phase is:

$$D_i \leftarrow \begin{pmatrix} IPC_i \\ CacheMisses_i \\ ContextSwitches_i \\ PageFaults_i \\ Power_i \\ RAM_i \\ CPUs\_mean\_util_i \\ CPUs\_std\_util_i \\ CPUs\_mean\_freq_i \\ CPUs\_std\_freq_i \\ GPU\_util_i \\ GPU\_freq_i \\ Power\_Mode_i \end{pmatrix}$$

For each device D_i where $i \in 0...N$ the profiled vector of a layer of a neural-network or a neural-network for execution context c where $c \in \{CPU, GPU\}$ and power mode pm is:

$$D_{i\_c\_pm} \leftarrow \begin{pmatrix} IPC_{i\_c\_pm} \\ CacheMisses_{i\_c\_pm} \\ ContextSwitches_{i\_c\_pm} \\ PageFaults_{i\_c\_pm} \\ Power_{i\_c\_pm} \\ RAM_{i\_c\_pm} \\ CPUs\_mean\_util_{i\_c_pm} \\ CPUs\_std\_util_{i\_c\_pm} \\ CPUs\_mean\_freq_{i\_c\_pm} \\ CPUs\_std\_freq_{i\_c\_pm} \\ GPU\_util_{i\_c\_pm} \\ GPU\_freq_{i\_c\_pm} \\ ExecutionTime_{i\_c\_pm} \end{pmatrix}$$

This vector is computed for each device Di where $i \in 0...N$, each power mode pm, each execution context c where $c \in \{CPU, GPU\}$.

By combining (vertical stack) all the vectors from the offline phase, the N vectors from the online phase along with the previous layer execution device (one hot encoded) , in the case of layered execution, the Observation State is constructed.

It is important to mention that all the values are normalized with min-max normalization in order to be in the same scale.

Action Space Definition

The action space is the output of the RL model. Is the landscape of decisions, with the most complex scenarios involving the selection of the destination device for the next layer or next neural-network request, specification of the power mode for that device, and determination of the execution context (CPU or GPU). These granular decisions are important for precise and adaptable resource allocation.

Despite that fact that these granular decisions are important for precise and adaptable resource allocation, the action space is relatively large and for this reason the model requires a huge number of training steps in order to learn. To mitigate this complexity and facilitate practical training, explore simplified variations is required. These simplified action spaces may involve choosing only the next device (without power mode and context), selecting both the device and execution context, or specifying only the device and power mode.

In the case of selecting only the next device, with fixed execution contex and devices power modes, the action space is simple a vector AS with length equal with the number of devices and each position $AS[i]$ represents the probability of picking the device i:

$$AS \leftarrow \begin{pmatrix} Device_0 \\ Device_1 \\ \vdots \end{pmatrix}$$

In the case of selecting the next device and the execution context, the action space is simple a vector

AS with length equal with twice the number of devices:

$$AS \leftarrow \begin{pmatrix} Device_{0\_cpu} \\ Device_{0\_gpu} \\ Device_{1\_cpu} \\ Device_{1\_gpu} \\ \vdots \end{pmatrix}$$

If $i \mod 2 == 0$, position $AS[i]$ represents the probability of picking the device $D = i//2$ in CPU context. On the other hand if $i \mod 2 == 1$, position $AS[i]$ represents the probability of picking the device $D = i//2$ in GPU context.

In the case of selecting the next device and power mode of the device, the action space is simple a vector AS with length equal with the number of devices x the power modes:

$$AS \leftarrow \begin{pmatrix} Device_{0\_pm\_0} \\ Device_{0\_pm\_1} \\ Device_{0\_pm\_2} \\ \vdots \\ Device_{1\_pm\_0} \\ Device_{1\_pm\_1} \\ Device_{1\_pm\_2} \\ \vdots \end{pmatrix}$$

Assuming that for each device the number of power modes is P, then the $AS[i]$ represents the probability of picking device $D = i//P$ in power mode $PM = i \mod P$.

The most complex scenario is to encode both contexts and power modes in the action space along with devices. In this scenarion, the action space is formulated by simply mixing the two previous cases:

$$AS \leftarrow \begin{pmatrix} Device_{0\_pm\_0\_cpu} \\ Device_{0\_pm\_0\_gpu} \\ Device_{0\_pm\_1\_cpu} \\ Device_{0\_pm\_1\_gpu} \\ \vdots \\ Device_{1\_pm\_0\_cpu} \\ Device_{1\_pm\_0\_gpu} \\ Device_{1\_pm\_1\_cpu} \\ Device_{0\_pm\_1\_gpu} \\ \vdots \end{pmatrix}$$

If $i \mod 2 == 0$, position $AS[i]$ represents the probability of picking the device $D = i//(2 * P)$, in power mode $P = (i//2) \mod P$ in CPU context. On the other hand if $i \mod 2 == 1$, position $AS[i]$ represents the probability of picking the device $D = i//(2 * P)$, in power mode $P = (i//2) \mod P$ in GPU context.

Reward System(s)

The reward function, at its essence, serves as a guidepost for the reinforcement learning model. Its purpose is to provide a clear and measurable indication of the desirability of certain actions taken within the framework. The reward function got a single aim – to encourage actions that enhance the overall efficiency and performance of the neural network execution process. It evaluates the outcomes of various resource allocation decisions, assigning values that reflect the success or failure of these decisions in achieving the optimization goals. In doing so, it helps the model learn to make choices that align with the framework's objectives, ultimately fostering efficiency and responsiveness in diverse edge computing environments.

Creating an effective rewarding system in the context of reinforcement learning is a complex and iterative process. It involves a deep understanding of the framework's objectives and the intricacies of edge computing. Edge computing introduces various variables that require adaptability. Finding a suitable rewarding system involves experimentation and ongoing development, fine-tuning the reward function to match the dynamic nature of edge computing. This process is marked by continuous learning and evolution, ensuring the framework's effectiveness in diverse and evolving environments.

The primary objective of the rewarding system revolves around ensuring that the neural network execution framework consistently meets or surpasses the Service Level Agreement (SLA). We set a baseline performance metric during the profiling phase, specifically targeting a time threshold of 2 times the worst execution time observed during profiling. This threshold serves as our benchmark for acceptable performance. When the framework achieves execution times better than this benchmark, we generate a positive reward that factors in both improved performance and reduced power consumption during the layer or whole neural-network execution. However, if the execution times exceed the SLA threshold, we consider only the deviation from the benchmark when determining the negative reward value. This rewarding concept forms the cornerstone of our reinforcement learning-based resource allocation strategy, striking a balance between performance optimization and power efficiency. Some rewarding systems that follow this concept are:

- **Rewarding System with Substraction**:
  - **Negative Reward**: In the case where execution $\gneq$ the SLA:

$$neg\_rew \leftarrow execution\_time - SLA\_time$$

  - **Positive Reward**: In the case where execution $\leq$ the SLA:

$$pos\_rew \leftarrow 1/(10 \times power\_consumption \times execution\_time)$$

  The reward values are in range of [-1, 1] because the times and power consumption are normalized to the range of 0-1.

- **Rewarding System with Division**:
  - **Negative Reward**: In the case where execution $\gneq$ the SLA:

$$neg\_rew \leftarrow -(execution\_time/SLA\_time)$$

  - **Positive Reward**: In the case where execution $\leq$ the SLA:

$$pos\_rew \leftarrow 1/(10 \times power\_consumption \times execution\_time)$$

  In case of layered execution many times, particularly in the initial phases of training, the model exhibits random behavior, resulting in execution times, including network latency, that can be up to two or even three orders of magnitude longer than expected. Consequently, this leads to substantial negative rewards. While this behavior isn't inherently detrimental, an issue arises

as the model evolves. Over time, the model tends to favor executing layers on the same device consistently. This preference minimizes network-related delays, such as data serialization and deserialization, HTTP overhead, and data transfer times, resulting in comparatively minor negative rewards. This situation may appear favorable when the device isn't under significant load. However, it becomes problematic under high load conditions. Even when subjected to device load simulations, the model persists in selecting the same device for execution. To mitigate the issue of enormous negative rewards during initial training phases, a reward clipping mechanism is employed, constraining rewards to a fixed range, typically [-10, +10].

Reinforcement Learning Model Architecture

Reinforcement Learning (RL) encompasses a diverse array of model architectures, each designed to tackle distinct challenges and scenarios. To optimize resource allocation for neural network execution, We have at our disposal a rich selection of RL methodologies. However, it's worth noting that our choice of RL architectures is somewhat constrained by the tools at our disposal. Specifically, We leverage the Stable Baselines3 Python package, which offers a set of well-established RL algorithms, including Advantage Actor-Critic (A2C) [39], Deep Deterministic Policy Gradient (DDPG) [40], Deep Q Network (DQN) [74], Hindsight Experience Replay (HER) [42], Proximal Policy Optimization (PPO) [18], Soft Actor-Critic (SAC) [44], and Twin Delayed DDPG (TD3) [40]. These RL models offer unique insights and capabilities, enabling us to tailor our resource allocation strategy to the complex and dynamic landscape of edge computing.

In this dissertation, We have chosen to employ the Proximal Policy Optimization (PPO) algorithm as our primary reinforcement learning method. PPO stands out for several reasons, making it a well-suited choice for addressing the resource allocation challenges We encounter in our dissertation:

- **Stability and Robustness**: PPO is known for its stability and robustness in training RL models. It often converges to a good policy reliably, which can be crucial in real-world applications where consistent performance is required.
- **Sample Efficiency**: PPO typically requires fewer samples (interaction with the environment) compared to other algorithms like DDPG or A3C. This makes it more suitable for scenarios where collecting data is costly or time-consuming, as in our case.
- **Safety and Exploration**: PPO incorporates a *trust region* approach, which limits policy updates to ensure that the new policy does not deviate too far from the old one. This safety mechanism helps prevent catastrophic policy updates, making it safer for exploration in critical environments.
- **Applicability to High-Dimensional Inputs**: PPO can handle high-dimensional state spaces and observations effectively, which is common in complex real-world. From *Observation State Definition* section its obvious that the input in our models is high dimensional too.
- **Proven Performance**: PPO has demonstrated strong performance in various domains, including robotics, autonomous vehicles, and game playing, which underscores its suitability for a wide range of tasks.

Moreover, We will incorporate a separate DQN (Deep Q-Network) model as an alternative scheduling mechanism within our system. This contributes to a comprehensive evaluation of our system's performance, enabling us to compare and assess how different reinforcement learning algorithms influence scheduling decisions. By introducing this alternative mechanism, We enhance the thoroughness and completeness of our experimentation, offering a more holistic understanding of our scheduler's capabilities across various scenarios.

Load Simulation During Training

To simulate the incoming request workload during actual inference, We employ a Non-Homogeneous Poisson Process (NHPP) [45]. This model allows us to recreate a diverse range of workloads that devices

within the cluster might experience in real-world scenarios. Unlike a homomorphic Poisson distribution, which assumes a constant request rate, the non-homomorphic variant introduces variability in request arrivals, reflecting the dynamic nature of edge computing environments. By using this simulation approach, We gain a more accurate representation of the demands placed on the framework, enabling us to train my reinforcement learning model to make informed decisions in the face of varying workloads and resource constraints.

A common way to simulate a Non-Homogeneous Poisson Process (NHPP) is Thinning [46]. Thinning allows for the generation of a Non-Homogeneous Poisson process characterized by a time-varying arrival rate, often denoted as λ(t), where λ(t) represents the intensity of events at a given time 't'. To replicate the dynamic ebb and flow of request arrivals in real-world scenarios, we adopt a non-homogeneous Poisson process with λ(t) modeled as a sine function zeroed in the negative part. This choice emulates bursts of requests interspersed with periods of inactivity. The core idea behind thinning remains consistent: events are initiated from a homogeneous Poisson process operating at a constant rate, typically referred to as λ_{max}. This homogeneous process generates events uniformly across time intervals. Subsequently, for each generated event, the corresponding intensity λ(t) at the event's timestamp is computed using the sine function. Random numbers within the range of 0 to 1 are then generated for each event. By comparing these random numbers to the ratio of λ(t) to λ_{max}, events are selectively retained or 'thinned out.' If the random number falls below this ratio, the event is retained; otherwise, it is discarded. Through this process, thinning ensures the generation of a non-homogeneous Poisson process that accurately reflects the dynamic fluctuations in event arrival rates, making it a valuable technique for simulating real-world scenarios with varying event intensities.

Algorithm 3 NHHP with Thinning

Input: Ω {A frequency that cycles every X time units, in radians}
Output: Number of incoming requests in next timestep

function NHPPThinning(Ω)
$\lambda(t)$ = lambda t: $max(0, \lambda_{max} * cos(\Omega * t))$
$t \leftarrow 0.0$
while $True$ **do**
  $t+=$ ExponentialVariate(λ_{max}) {A random exponential distribution floating number}
  $ran =$ Random$(0, 1)$ {A random number between 0 and 1}
  if $ran \leq \lambda(t)/\lambda_{max}$ **then**
    yield $poisson(\lambda = \lambda(t))$
  else
    yield 0
  end if
end while
return

Algorithm 3 implements the Non Homogeneous Poisson Process with Thinning and figure 4.2.1 illustrates the way this algorithm produces the events.

Auxiliary Services

Monitoring Execution

To closely monitor the execution of neural network and accurately compute the associated metrics, a coordinated approach is employed. On each device within the cluster, a dedicated Flask REST API service resides. When a layer or a neural-network execution is initiated, an HTTP request is sent to the start monitor endpoint on the device. Subsequently, on the device side, a process is spawned to execute the tegrastats CLI, enabling the measurement of power consumption. Simultaneously, the execution time is tracked using Python's time package. These two processes work cohesively, recording power consumption and execution time throughout the execution. The monitoring process concludes

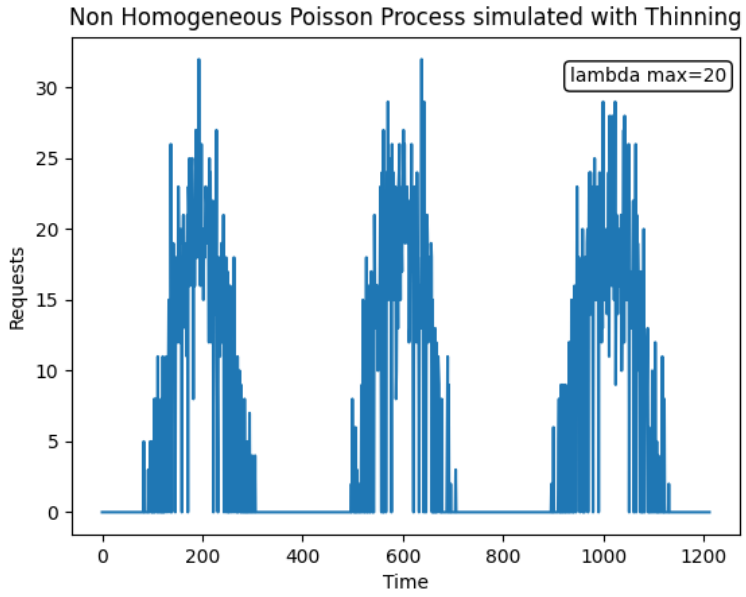


Figure 4.2.1: Non Homogeneous Poisson Process with Thinning, $\lambda_{max} = 20$

when the device requests the location for the next layer or posts the end of execution, prompting a final HTTP request to the stop monitor endpoint. This action halts the power consumption monitoring process and finalizes the execution time measurement. The combined data from these processes are used to compute the final reward of the models decision.

Real-time State Tracking

Simultaneously, an additional service within each device leverages the Python requests package to periodically transmit the device's current state to the main server. These state updates, which include essential performance metrics, are dispatched at regular intervals, typically every 500 milliseconds, and are facilitated through the perf and tegrastats CLI tools. This dynamic data collection approach ensures that the framework remains constantly informed about device performance and resource utilization in real-time, forming a crucial part of the model's input. It empowers the framework to make informed decisions and dynamically optimize resource allocation in response to ever-evolving edge computing scenarios.

Parallel Training of Multiple RL Models

The architecture of the services and the central server has been designed to facilitate the simultaneous training of multiple RL models. The process is relatively straightforward: each RL training process is linked with a corresponding monitor process responsible for tracking executions on the devices. These monitor processes are configured to listen on a predefined port, often designated as port 10000, within each device. Additionally, to ensure consistency and accuracy, all RL training processes obtain real-time device states from a shared process within each device, which periodically transmits the latest device state data. This collaborative setup empowers the framework to concurrently train multiple RL models. It effectively harnesses parallelism to enhance hyperparameter tuning, ultimately optimizing the performance of RL models and resource allocation across a wide array of dynamic edge computing scenarios.

In order to gain a deeper understanding of the framework, We provide an example to showcase how this

architecture and framework operate. This scenario is about the layered execution Service described in 4.1.1, the whole Neural-Network execution Service, described in 4.1.2, is straight forward.

In figure 4.2.2, a computational scenario of our framework is illustrated. More specifically, A server and three devices collaborate to execute a four-layer neural network in response to a client's request, orchestrated by a central server driven by an Reinforcement Learning model. The sequential process unfolds as follows: the server initiates by dispatching the first neural network layer to Device 2. Once Device 2 completes this layer's execution, it queries the RL model for the location of the next layer. The RL model designates Device 1 for executing the second layer, after which Device 1 transmits the third layer to Device 0, following the RL model's guidance. Device 0, upon completing the execution of the third layer, consults the RL model once more, which instructs Device 0 to finalize the execution locally for the fourth and final layer. Device 0 carries out this computation and subsequently returns the results to the central server. Lastly, the server communicates these results to the client, thus concluding the entire process.

The *S-D-I*, *S-D* and *D-D* mentioned in figure 4.2.2 are the abbreviations of the following CloudEvent Structures:

- **Server-Device Init CloudEvent Structure (S-D-I)**:
  - Next Service URL (includes model name, next device and execution context)
  - Servers Scheduling Service IP and Port
- **Server-Device Final CloudEvent Structure (S-D-F)**:
  - Output Vector of last layer
- **Device-Device CloudEvent Structure (D-D)**:
  - Layer index to execute
  - Serialized Input data
  - Input data shape
  - Input data data_type
  - Servers Scheduling Service IP and Port
- **Server-Device CloudEvent Structure (S-D)**:
  - Next Service URL (includes model name, next device and execution context)
  - Servers Scheduling Service IP and Port

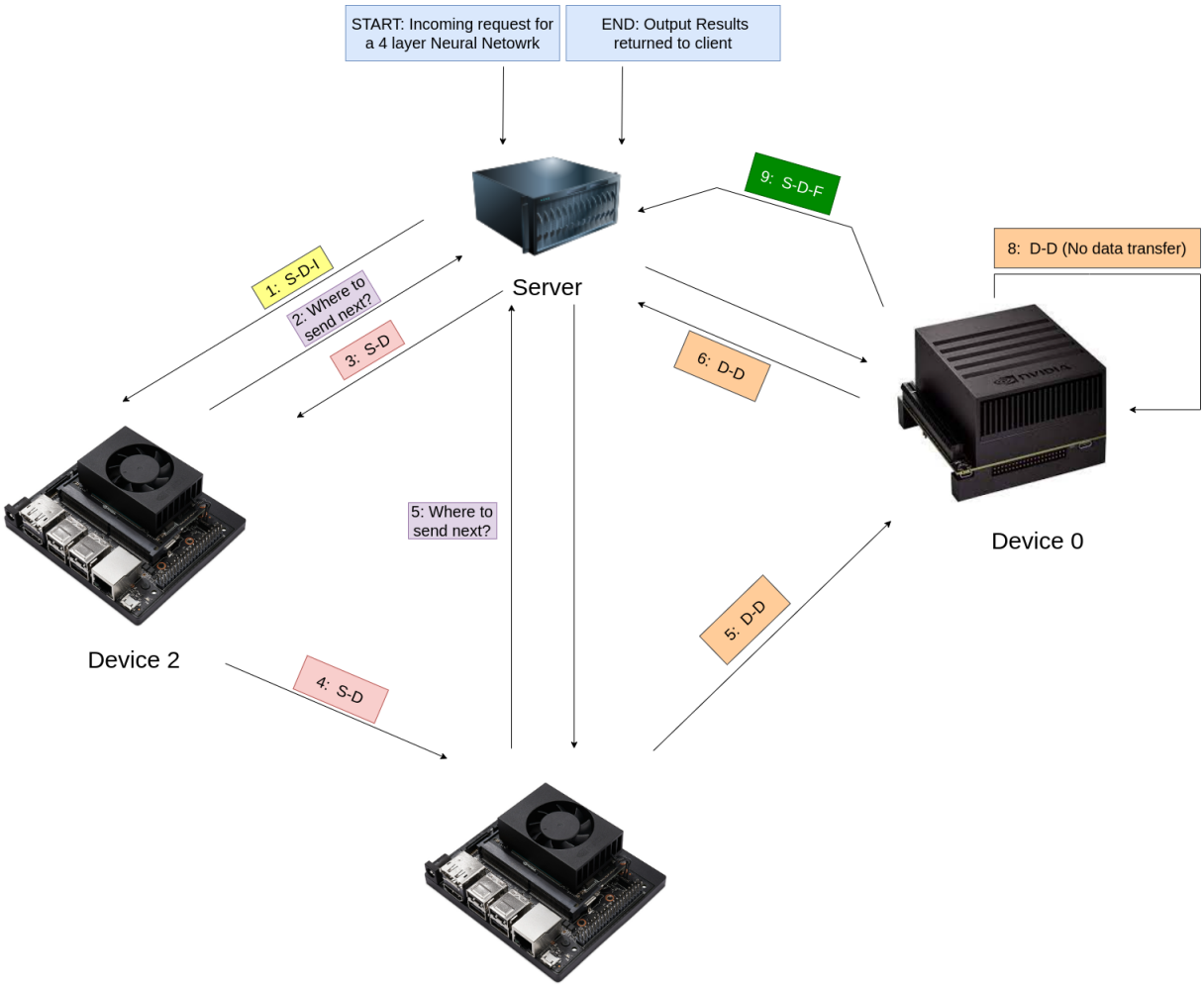


Figure 4.2.2: A simple visualized example of architecture

Chapter 5

Evaluation

This section details the evaluation process we used to assess the framework and scheduling model described in sections 4.1 and 4.2 respectively.

5.1 Experimental Setup

The experimental setup encompasses three edge devices and a central control unit. These components play pivotal roles in investigating resource allocation within edge computing scenarios.

Edge Devices

At the core of this experimental configuration are three edge devices. The Jetson AGX Xavier, a high-performance system-on-a-chip (SoC) platform, provides substantial computational power. Accompanying it are two Jetson Xavier NX devices, recognized for their energy efficiency and suitability for edge applications. These edge devices are responsible for executing the various layers of neural networks by running the service described in sections 4.1.1 and 4.1.2. They are not involved in the decision-making processes related to scheduling and resource allocation; instead, they carry out the computations as directed by the central control unit. This setup closely emulates practical edge scenarios, where tasks are efficiently distributed across edge devices to mitigate latency and enhance overall system efficiency.

Central Control Unit (VM)

Supplementing this ensemble of edge devices is a virtual machine (VM) hosted on a centralized server, functioning as the central control unit. The VM assumes a pivotal role in the orchestration and optimization of resource distribution throughout the network. It oversees and governs resource allocation decisions by hosting the Reinforcement Learning algorithm described in section 4.2.

The table 5.1 provides a detailed insight of the devices technical characteristics.

5.2 Implementation Challenges

In the pursuit of translating theoretical concepts into practical solutions, the implementation phase of this research confronted a multitude of intricate challenges.

- **GPU Support - Kubernetes**: To enable GPU execution context within the Kubernetes cluster, the deployment of the NVIDIA plugin is imperative. This plugin seamlessly integrates GPU

1lightgray

Device	CPU	L2	L3	RAM	GPU
Xavier AGX	8-Core ARM v8.2 64-Bit	8MB	4MB	32GB LPDDR4X	512-core NVIDIA Volta and 64 Tensor cores
Xavier NX	6-core NVIDIA Carmel ARM® v8.2 64-bit	6MB	4MB	8GB LPDDR4x	384-core NVIDIA Volta™ GPU and 48 Tensor Cores
VM	16-core Intel Xeon (Skylake, IBRS)	64MB	256MB	16GB	–

Table 5.1: Technical characteristics of heterogeneous Edge nodes and Cloud Server

support into the cluster, a crucial requirement for the efficient execution of deep learning workloads. However, it's important to note that the compatibility between the NVIDIA plugin and Jetson devices hinges on the version of NVIDIA's Jetpack software [48]. Specifically, the NVIDIA plugin is compatible with Jetpack versions $\geq$ 5.0 (actually with nvidia-docker [49] >= 2.0 and nvidia-container-toolkit [50] >= 1.7.0 that are installed in Jetpack $\geq$ 5.0). Consequently, to ensure alignment, the Jetsons must undergo a reflash process, updating them to a compatible Jetpack version, such as 5.0.1. Furthermore, the PyTorch image used as base image, which is is 4t-pytorch:r35.2.1-pth2.0-py3, sourced from the NGC Catalog, impose significant storage demands (13GB). Considering the constrained EMMC storage on Jetson devices, a practical solution involves the utilization of external USB flash drives. These external storage options serve to augment storage capacity, facilitating the seamless accommodation of these substantial images. By implementing this, potential storage constraints are effectively mitigated.

- **GPU Support - Pytorch**: Loading models onto the GPU with PyTorch results in huge RAM utilization overheads (around 1-1.5GB), which is not aligned with the resource constraints of edge computing, particularly on Jetson Xavier NX devices. For this reason, the evaluation will be done only in CPU execution context. Future work will address this issue to optimize GPU model loading for improved edge computing efficiency.

- **Reinforcement Learning Action Space**: In the realm of Reinforcement Learning (RL), agents encounter the task of traversing the action space to acquire an optimal policy. However, when confronted with a large action space, the agent's exploration process becomes notably protracted, consequently extending the duration of training sessions. As a response to this challenge, the action space has been simplified, reducing its dimensionality to three elements. Each of these three elements corresponds to a distinct device, enabling the agent to solely determine the location for executing the subsequent layer. One more simplification involves the utilization of one neural network (MobilenetV2) during training.

- **Reinforcement Learning Tuning**: Tuning in reinforcement learning is a complex and time-consuming task. This study will focus on a limited scope of experimentation, utilizing four distinct reward systems outlined in the scheduling section. Furthermore, the exploration of hyperparameters will primarily revolve around varying the Generalized Advantage Estimation Lambda parameter, specifically considering values of [0.9, 0.7, 0.5, 0.3, 0.1].

- **Load Simulation**:

5.3 Experimental Results

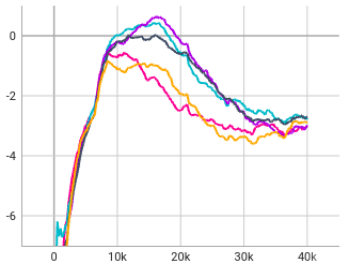
As previously stated, the conducted experiments comprehensively covered two distinct rewarding systems, namely subtraction and division, encompassing a spectrum of lambda (λ) values, precisely [0.1, 0.3, 0.5, 0.7, 0.9], all executed within the established framework tailored specifically for this study. Both services, encompassing layer-by-layer execution and holistic offload, will undergo thorough evaluation. To assess the performance of the Reinforcement Learning algorithms, We will employ a set of dummy schedulers.

- **Random Scheduler**: This scheduler randomly assigns neural networks for execution on the available devices.
- **Round Robin Scheduler**: The round-robin scheduler allocates neural networks in a cyclic fashion to the devices.
- **Less CPU Utilization Scheduler**: This scheduler selects the device with the lowest CPU utilization for neural network execution.

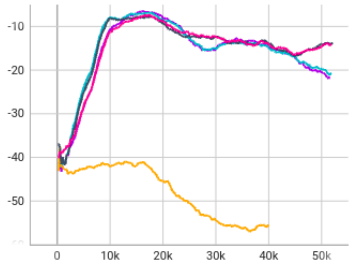
Reward During training

This section presents the episodic mean rewards of some training sessions, providing a direct view of the encountered challenges and the model's progress in reinforcement learning.

As an illustration, figures 5.3.1a and 5.3.1b displays the episodic mean rewards for the training sessions where the subtraction and division rewards were employed with full NN offload, featuring different lambda values for the Generalized Advantage Estimator.



(a) Subtraction mean episodic Reward during Training with full NN offload



(b) Division mean episodic Reward during Training with full NN offload

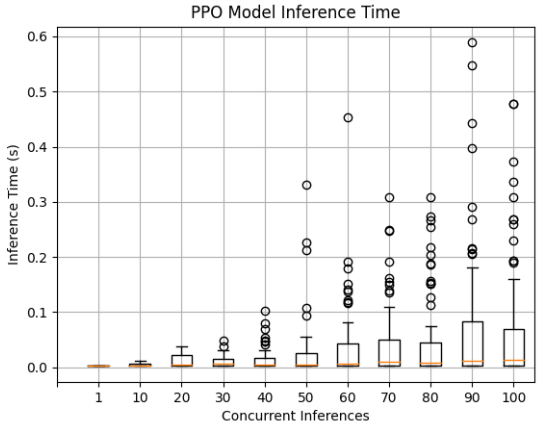
The episodic mean rewards, as observed in the figures, initially display a consistent upward trend, showcasing the model's progress. However, it becomes evident that after a certain number of episodes, these rewards start to decrease. This decline in rewards prompts us to select the model when it attains the highest reward per episode, as this signifies the model's peak performance before any degradation in its learning process occurs.

The rewards in the layered network execution follow a similar pattern. Again we select the model when it attains the highest reward per episode.

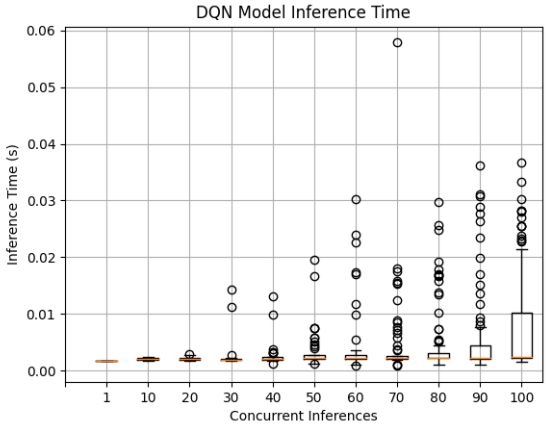
Scheduler overhead

It is imperative to consider and quantify the overhead incurred by the RL schedulers in our system-framework, which is simply an Multi-Layer Perceptron (MLP) inference. In Figures 5.3.2a and 5.3.2b, We present a visual representation of the overhead associated with the inference of both PPO and DQN models. Even when subjected to 100 concurrent inferences of each model, the **median** inference time remains within a few milliseconds. Notably, the PPO model exhibits slower performance compared to the DQN model, with a few exceptional cases reaching a total inference time of 600 milliseconds.

In contrast, the maximum inference time for the DQN model is a mere 6 milliseconds. Given the negligible median time, it is deemed inconsequential and is therefore not considered in the subsequent experiments.



(a) PPO model scheduler inference overhead for different number of concurrent executions



(b) DQN model scheduler inference overhead for different number of concurrent executions

Time and Energy system evaluation

In this section, We undertake an extensive evaluation of our scheduling system, providing experimental results of its real-time performance and energy efficiency. This evaluation is pivotal for assessing the practical viability of our system in to the other schedulers discussed in the 5.3 paragraph. Through a series of comprehensive experiments, We aim to provide an unbiased and data-driven analysis of our system's capabilities, highlighting its potential strengths and limitations. These experiments form the basis for further optimization and refinement of our system.

To understand the evaluation results, it's important to explain the naming convention used during the evaluation process. Each diagram's horizontal axis represents the schedulers. The initial letter in each label signifies the reward system employed, while the subsequent characters specify the model's configuration.

- S****_lambda_{LAMBDA VALUE}_{MODEL ITERATION}: PPO model with subtraction rewarding system, with *LAMBDA VALUE* the lambda in GAE and *MODEL ITERATION* the model iteration that is picked
- D****_lambda_{LAMBDA VALUE}_{MODEL ITERATION}: PPO model with division rewarding system, with *LAMBDA VALUE* the lambda in GAE and *MODEL ITERATION* the model iteration that is picked
- S****_DQN_lambda_{LAMBDA VALUE}_{MODEL ITERATION} (only with whole execution): DQN model with subtraction rewarding system, with *LAMBDA VALUE* the lambda in GAE and *MODEL ITERATION* the model iteration that is picked
- D****_DQN_lambda_{LAMBDA VALUE}_{MODEL ITERATION} (only with whole execution): DQN model with division rewarding system, with *LAMBDA VALUE* the lambda in GAE and *MODEL ITERATION* the model iteration that is picked
- {DUMMY SHEDULER}: Just the dummy scheduler.
- {DUMMY SCHEDULER}_load: The dummy scheduler with the services that send the devices states running. By default this state runs for the RL based schedulers. We evaluating the dummy

schedulers with this load in order to examine the sensitivity of dummy schedulers to external load.

Monitor Service Overhead

In this subsection, we focus exclusively on the monitor service overhead. Demonstrating the impact of the monitor service is crucial for a comprehensive assessment of its effects on system performance, resource utilization, and overall efficiency.

Figures 5.3.3 and 5.3.4 show the CPU utilization and Power consumption of monitor service across all devices.

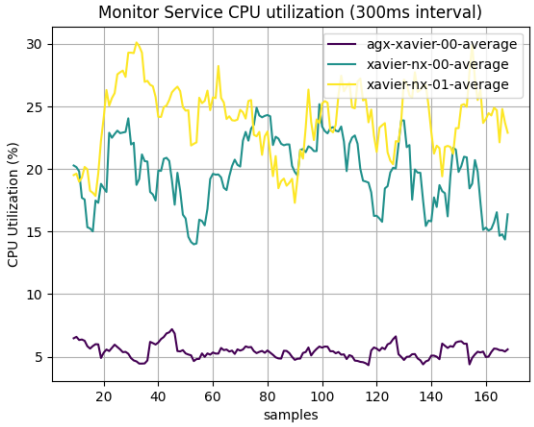


Figure 5.3.3: Monitor Service CPU utilization

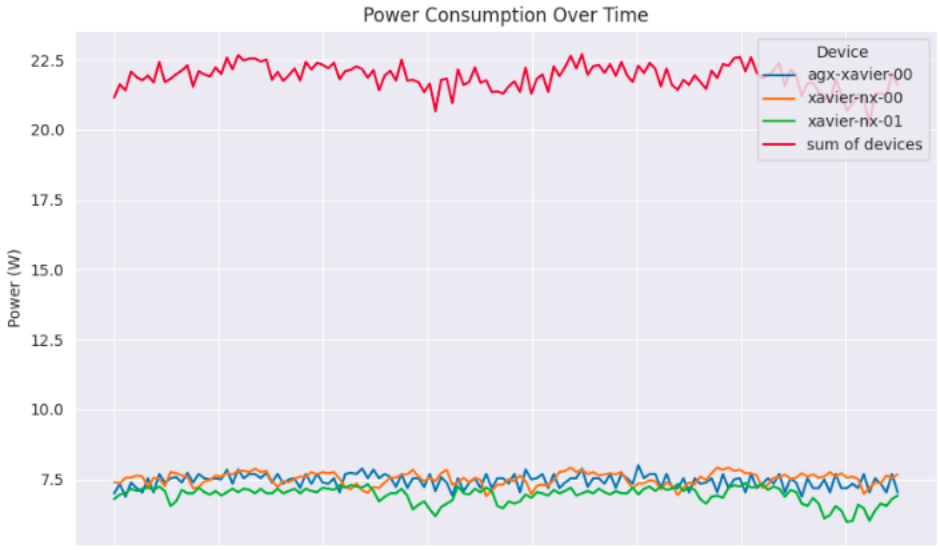


Figure 5.3.4: Monitor Service Power Consumption

Observing the results we see that in xavier-nx-00 device, the CPU utilization registers at a range of 20-28% approximately, in xavier-nx-01 registers at a range of 15-25%, while in the agx-xavier-00 device, it remains lower at around 5-6%. The monitor service has a relatively substantial performance impact in NX devices.

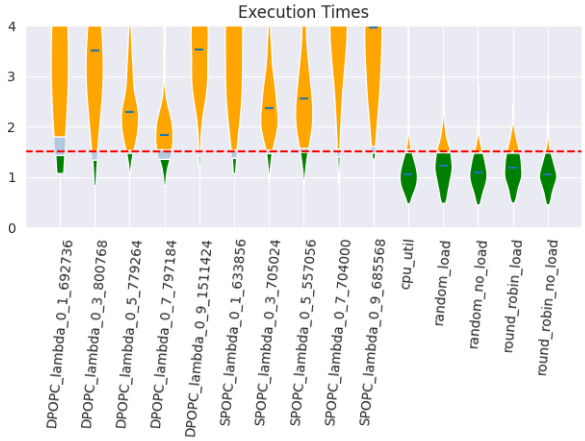
According to power consumption, across all devices, when exclusively running the monitor service, power consumption hovers consistently at approximately 7.5 Watts.

Coordinated Devices Experiment

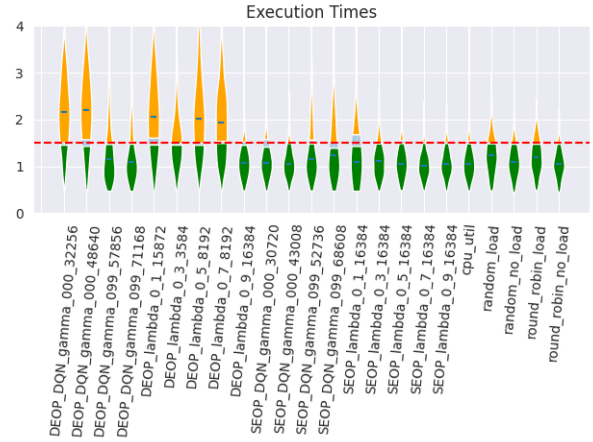
In the first type of experiment We focus on coordinated device behavior. In this scenario, certain devices, acting as clients, are synchronized to initiate their requests at approximately the same time within predefined intervals T (e.g. 3s). For instance, We may have multiple cameras synchronized to request image inference at the same time interval. This experiment allows us to simulate real-world situations where devices collaborate or coordinate their activities. By studying how our scheduler handles these coordinated requests, we gain insights into its ability to manage and optimize simultaneous workloads, which is particularly relevant in applications where precise timing and synchronization are critical for achieving desired outcomes.

This experiment is simulated using the Non Homogeneous Poisson Process, the same process used to simulate the load during training. Employing the algorithm 3, We conduct an experiment with $\lambda_{max} = 15$. This algorithm, in each timestep, specifies the devices that send concurrent requests. The interval T is equal to 3.

For $\lambda_{max} = 15$ in figures 5.3.5a and 5.3.5b is illustrated the total execution time for whole and layered services. In figures 5.3.6a and 5.3.6b are illustrated the results according to the energy consumption. Moreover, in order to better understand the results, in figures 5.3.7a and 5.3.6b We provide the percentage of devices used during each experiment and in figures 5.3.8a and 5.3.8b the violation percentages (the percentage of requests that exceeded the SLA). Ultimately, for layered execution, figure 5.3.9 shows the fraction of network time and the fraction of execution time.

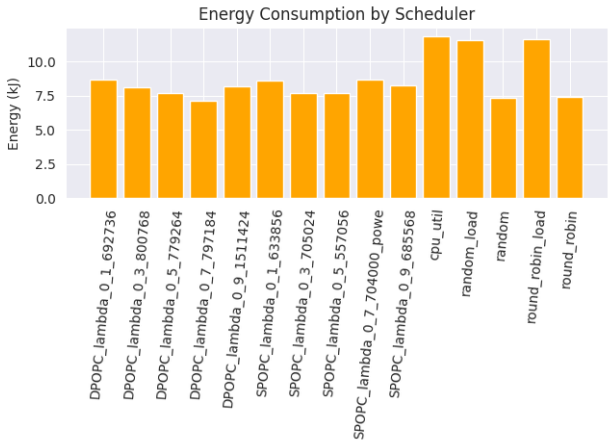


(a) Layered Request Execution time for $\lambda_{max} = 15$

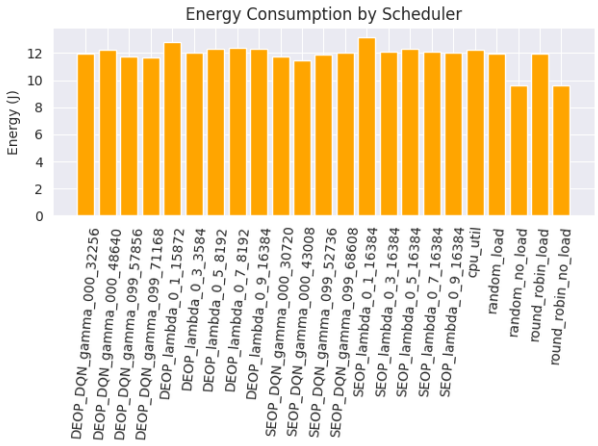


(b) Whole Request Execution time for $\lambda_{max} = 15$

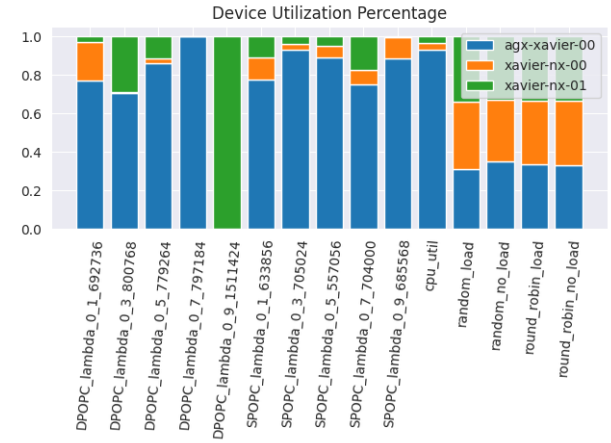
Layered Execution: The cumulative execution times and instances of violations are notably greater in the context of layered execution, as compared to the entirety of execution facilitated by dummy schedulers (as depicted in Figure 5.3.5a and Figure 5.3.8a). Upon close examination of Figure 5.3.9, it becomes evident that a substantial proportion of the overall time expenditure pertains to network-related activities. The execution itself constitutes a relatively small portion of the total time. The increased demand on network resources predominantly stems from the numerous requests generated at each neural network layer. This extensive request generation results in congestion within the server. Notably, a heightened neural network depth (more layers) corresponds with an increased volume of generated requests. Schedulers exhibiting more favorable temporal outcomes are those that maintain a consistent execution pathway within a single device, predominantly relying on a singular device for processing tasks. These schedulers, exemplified by DPOPC_lambda_0_7_797184, record substantially reduced network-related temporal overheads, given the absence of inter-device data transfers. However,



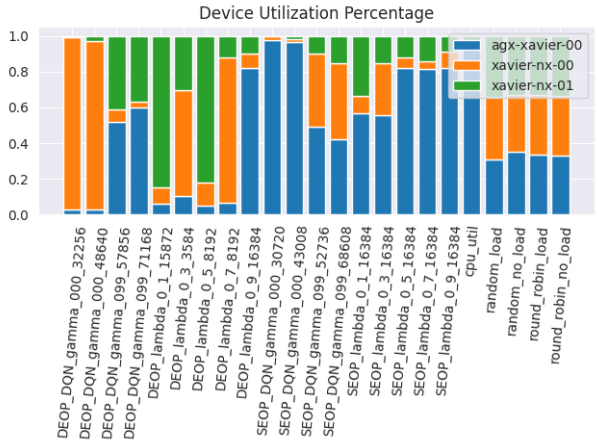
(a) Layered Request Execution Energy for $\lambda_{max} = 15$



(b) Whole Request Execution energy for $\lambda_{max} = 15$



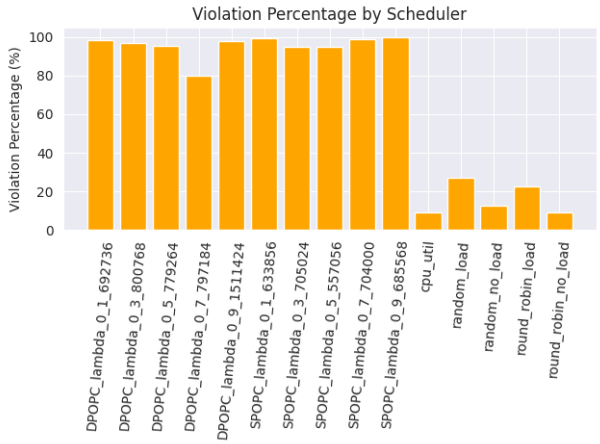
(a) Layered Request Execution Device utilization percentage for $\lambda_{max} = 15$



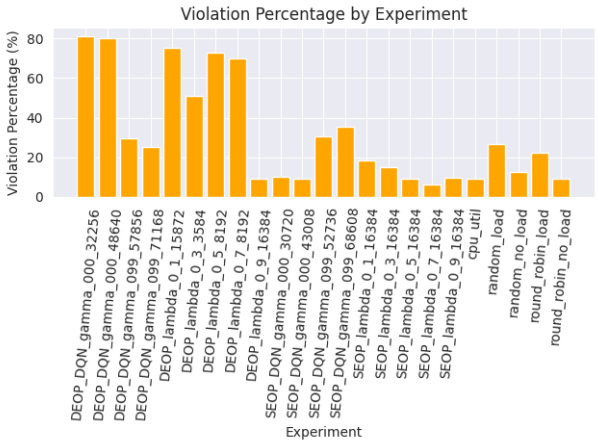
(b) Whole Request Execution Device utilization percentage for $\lambda_{max} = 15$

it is imperative to acknowledge that such schedulers inevitably lead to device overutilization. For instance, the DPOPC_lambda_0_9_1511424 scheduler, which offloads all tasks to xavier-nx-01, yields considerably extended total execution durations. In contrast, the DPOPC_0_7_797184 scheduler, which offloads all processing tasks onto the high-performance agx-xavier-00 device, achieves notably shorter total execution times. It is worth highlighting the responsiveness of the dummy schedulers with additional external workload fluctuations, manifesting as an approximate 16% elevation in violation rates when subjected to concurrent low-intensity workloads on each device (monitor workload). In terms of energy consumption, the schedulers consistently exhibit low energy utilization, averaging around 7.5 kilojoules. The DPOPC_lambda_0_7_797184 scheduler stands out with the lowest energy consumption, with 7 kilojoules, due to its exclusive reliance on agx-xavier-00 with no inter-device data transfers, as previously elucidated. The other schedulers also perform commendably in terms of energy efficiency. Notably, dummy schedulers and RL-based schedulers exhibit comparable energy consumption profiles, but in dummy schedulers the introduction of concurrent low-intensity workloads across devices leads to a noticeable increment in energy consumption, rising from approximately 7.5 kilojoules to around 12 kilojoules.

Whole Execution: The total execution times are quite similar, but there are some exceptions. These



(a) Layered Request Execution Violation percentage for $\lambda_{max} = 15$



(b) Whole Request Execution violation percentage for $\lambda_{max} = 15$

Figure 5.3.9: Execution vs Network time in layered execution

exceptions include the DEOP_DQN_gamma_000_32256, DEOP_DQN_gamma_000_48640, DEOP_lambda_0_1_15872, DEOP_lambda_0_3_3584, DEOP_lambda_0_5_8192, and DEOP_lambda_0_9_8192 schedulers. These schedulers heavily use xavier-nx-0* devices, which are less powerful than agx-xavier-00. This overuse leads to longer execution times, with violation percentages reaching up to 80%. On the other hand, most other RL-based schedulers primarily utilize agx-xavier-00, allocating only a portion of tasks to the xavier-nx-0* devices. This approach results in shorter execution times and even beats the dummy schedulers. Specifically, the round-robin scheduler with and without additional load, has violation percentages of 22% and 9.5%, respectively. The random scheduler has violation percentages of 25% and 15%, while the CPU utilization-based scheduler maintains a 10% violation rate. Among the non-exception RL-based schedulers, many achieve good performance with low violation percentages. For example, DEOP_lambda_0_9_16384, SEOP_DQN_gamma_000_43008, and SEOP_lambda_0_5_16384 all have violation percentages

of 9%. Notably, **SEOP_lambda_0_7_16384 achieves the lowest violation percentage at 5%**.According to the energy consumption, all the schedulers achieve similar performance with consumption around 12 kiloJoule. The lowest energy consumption is achieved by round robin and random, without the external load, and the round robin and random with external load also consume 12 kiloJoule. This indicates that the monitoring service responsible for reporting device status is the primary reason for the energy difference between the no-load dummy schedulers and other schedulers.

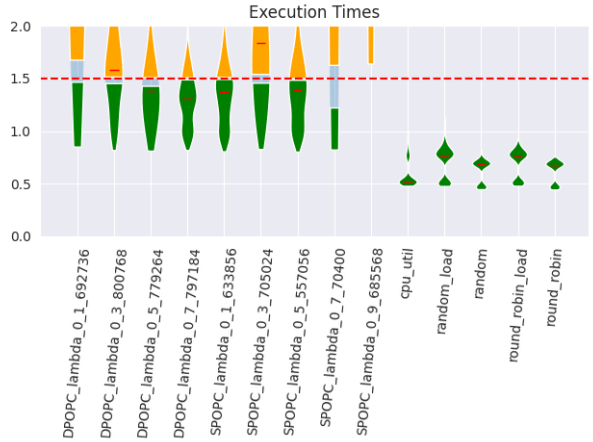
It is worth to mention that this evaluation experiment is challenging for our RL-based schedulers. The reason is that when simultaneously requests arrive in the server, which hosts the scheduler, and the scheduler processes all of these requests with nearly identical device statuses as input. Consequently, the scheduler schedules these requests without adequately considering their potential impact on the devices. This operational approach may lead to suboptimal and inefficient decisions, particularly within the context of the specific experimental framework.

Independent Clients Experiment

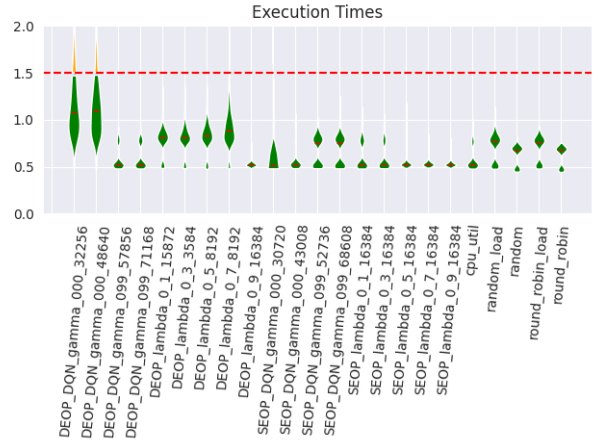
In this type of experiment devices asynchronously send requests to the server, that host the scheduler. The experiments revolve around two key parameters: the number of participating devices and the rate at which these devices send requests. By varying these factors, We evaluate the system's performance under a range of demand conditions.

LAMBDA_MAX=25, NUM_CLIENTS=1:

In this case there is only one client with bursting periods of requests. In figures 5.3.10a and 5.3.10b is illustrated the total execution time for whole and layered services. In figures 5.3.11a and 5.3.11b are illustrated the results according to the energy consumption. Moreover, in order to better understand the results, in figures 5.3.12a and 5.3.11b We provide the percentage of devices used during each experiment and in figures 5.3.13a and 5.3.13b the violation percentages (the percentage of requests that exceeded the SLA). Ultimately, for layered execution, figure 5.3.14 shows the fraction of network time and the fraction of execution time.

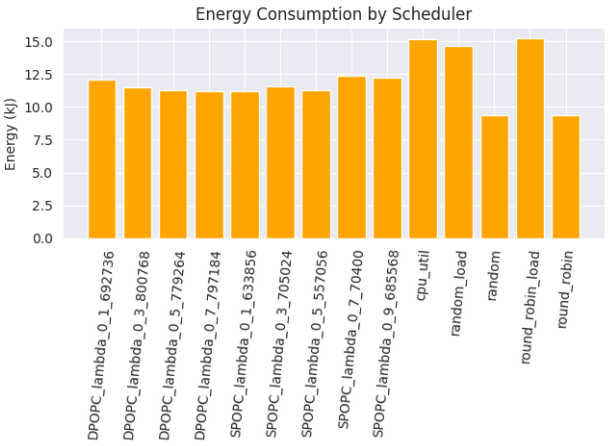


(a) Layered Request Execution time for $\lambda_{max} = 25$ and $num\_clients = 1$

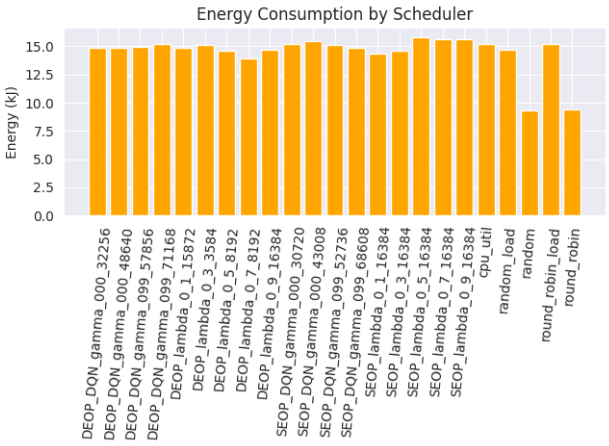


(b) Whole Request Execution time for $\lambda_{max} = 25$ and $num\_clients = 1$

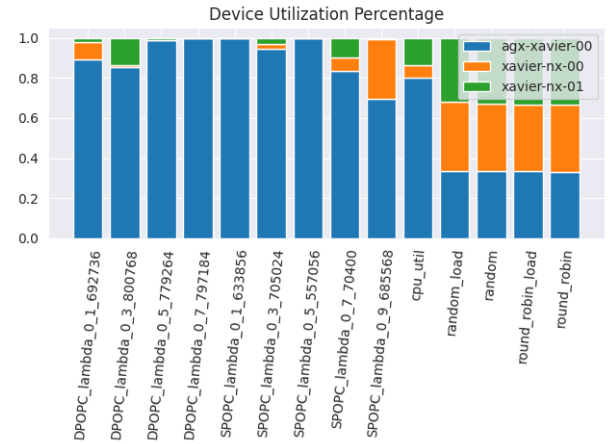
Layered Execution: Observing the figures 5.3.10a and 5.3.11a it becomes evident that the system operates under a relatively light load. Notably, in certain instances, layered execution demonstrates a relatively low percentage of violations. For instance, the DPOPC_lambda_0_7_791184 scheduler and the SPOPC_lambda_0_1_633856 scheduler exhibit violation percentages of 23% and 30%, respectively, a marked improvement from the previous experiment, where these schedulers had 80% 95%, respectively. Furthermore, figure 5.3.14 provides insights into the network time, revealing that



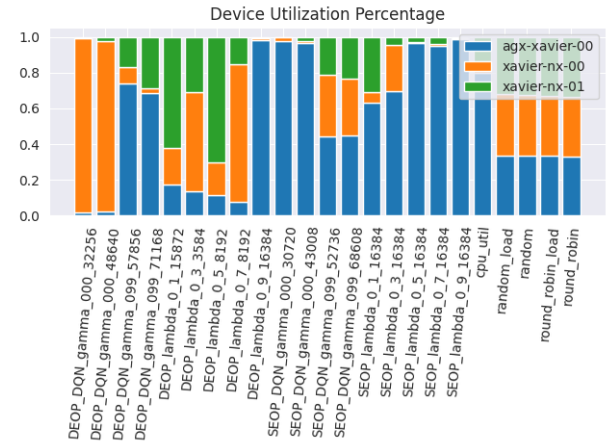
(a) Layered Request Execution Energy for $\lambda_{max} = 25$ and $num\_clients = 1$



(b) Whole Request Execution energy for $\lambda_{max} = 25$ and $num\_clients = 1$



(a) Layered Request Execution Device utilization percentage for $\lambda_{max} = 25$ and $num\_clients = 1$



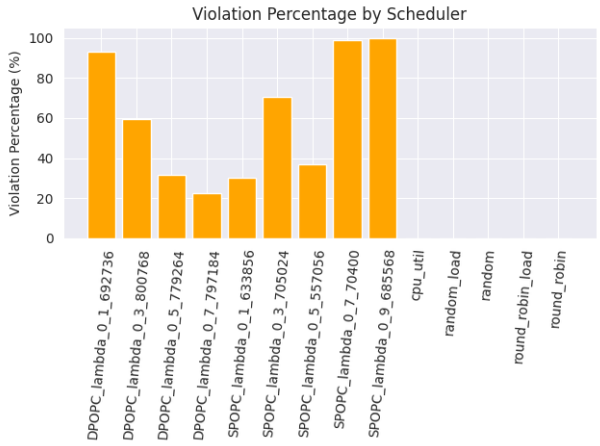
(b) Whole Request Execution Device utilization percentage for $\lambda_{max} = 25$ and $num\_clients = 1$

the majority of the schedulers exhibit significantly reduced network time compared to the previous experiment. This can be attributed to the absence of network congestion on the server, leading to enhanced system efficiency.

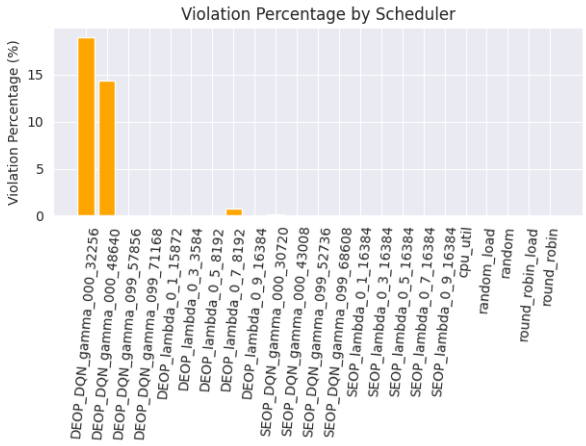
Of particular interest is the performance of the dummy schedulers, which, in the absence of external load, achieve zero violation percentages. However, when subjected to external load, these dummy schedulers appear to consume more energy compared to their counterparts.

Whole Execution: In terms of overall execution, as shown in Figure 5.3.13b, nearly all schedulers achieve a 0% violation rate, indicating their efficient handling of the relatively light workload. The only exceptions to this pattern are the DEOP_DQN_gamma_000_32256 and DEOP_DQN_gamma_000_48640 schedulers, which struggle due to routing all requests to the xavier-nx-00 resource.

Regarding energy consumption, figure 5.3.11b shows that the results remain similar with our previous experiment. Most models perform the same, except for the dummy schedulers without load indicating once again that the monitor service is responsible for a significant amount of energy consumption.



(a) Layered Request Execution Violation percentage for $\lambda_{max} = 25$ and $num\_clients = 1$



(b) Whole Request Execution violation percentage for $\lambda_{max} = 25$ and $num\_clients = 1$

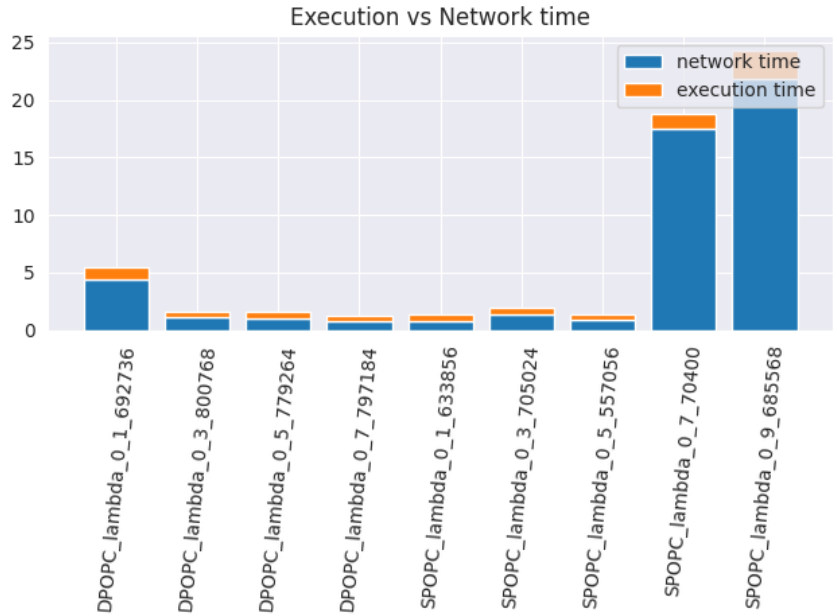


Figure 5.3.14: Execution vs Network time in layered execution for $\lambda_{max} = 25$ and $num\_clients = 1$

In the subsequent experiments, We have omitted the presentation of layered execution results. This omission is due to the notably poor performance in total execution times, rendering their display unnecessary.

LAMBDA_MAX=15, NUM_CLIENTS=5:

In this case there are five (5) clients with bursting periods requests. The burst rate is a bit smaller than the previous experiment ($lambda_{max} = 15$ instead of 25).

In this particular scenario, the performance of most schedulers exhibits a commendable level of efficiency. Notably, the dummy schedulers, when no external load is applied, achieve almost 0% violation

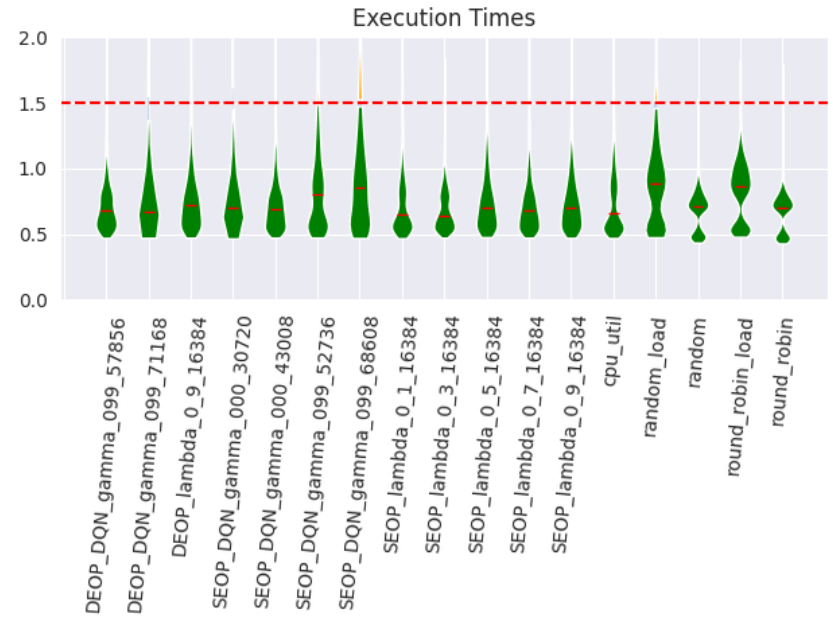


Figure 5.3.15: Whole Request Execution time for $\lambda_{max} = 15$ and $num\_clients = 5$

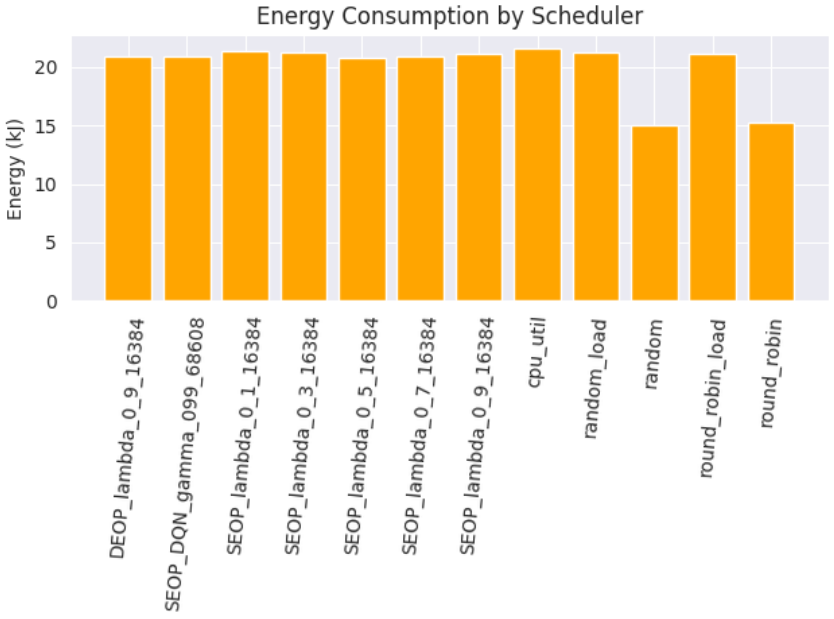


Figure 5.3.16: Whole Request Execution energy for $\lambda_{max} = 15$ and $num\_clients = 5$

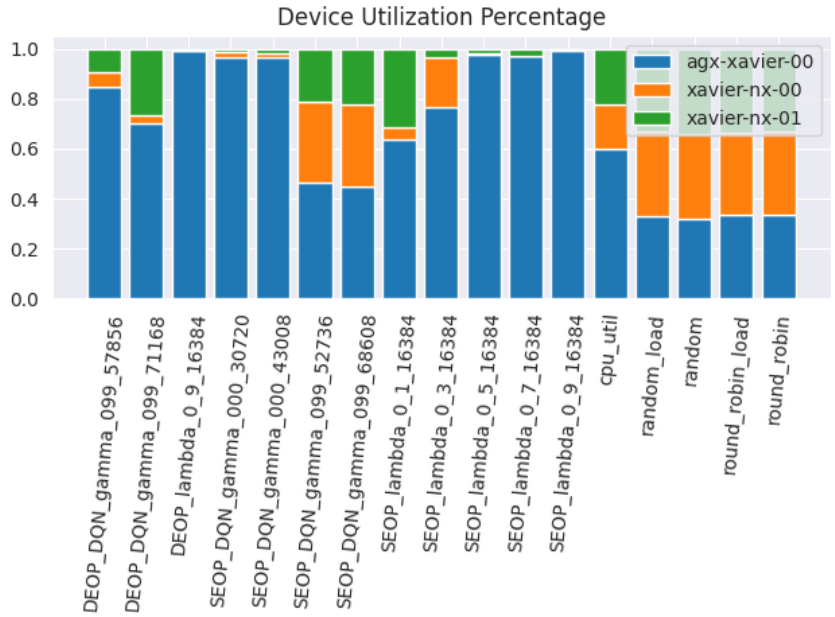


Figure 5.3.17: Whole Request Execution Device utilization percentage for $\lambda_{max} = 15$ and $num\_clients = 5$

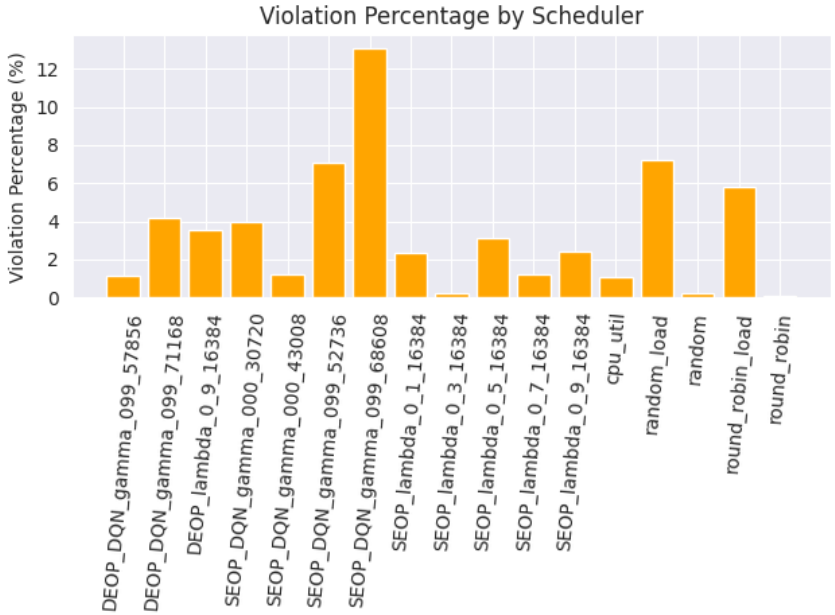


Figure 5.3.18: Whole Request Execution violation percentage for for $\lambda_{max} = 15$ and $num\_clients = 5$

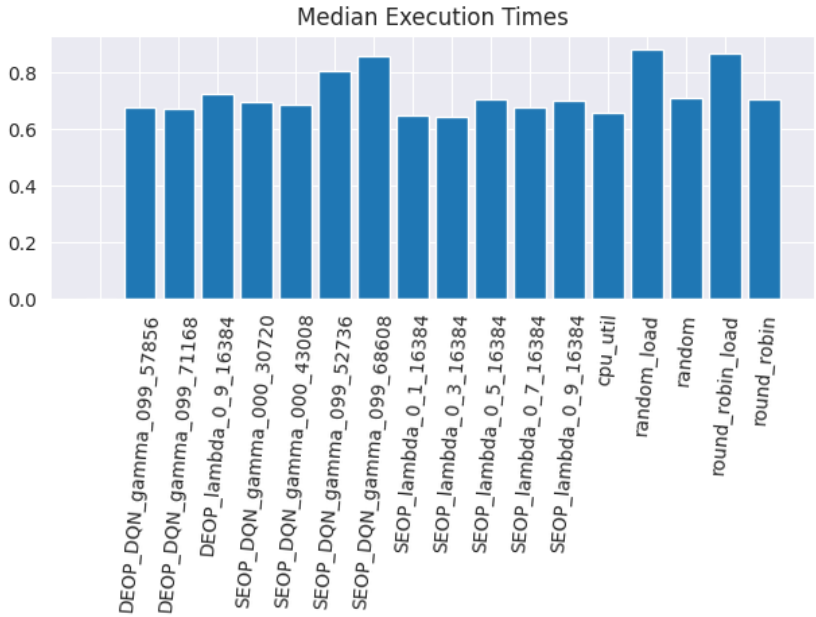


Figure 5.3.19: Median Request Execution time for $\lambda_{max} = 15$ and $num\_clients = 5$

rate, demonstrating their reliability in managing workloads. The CPU_UTIL scheduler also stands out, with a highly effective performance achieving a violation rate of less than 1%. Furthermore, the SEOP_lambda_0_3_16384 scheduler delivers an exceptional performance, with near-zero violation rates. This is evident from the data presented in Figure 5.3.15, where this scheduler, alongside SEOP_lambda_0_1_16384 and CPU_UTIL schedulers, records the lowest median execution times. From Figure 5.3.19, we can observe that SEOP_lambda_0_3_16384 scheduler leverages the powerful agx-xavier-00 device while judiciously offloading a significant portion of the workload to xavier-nx-0* devices. This strategic approach results in an impressive nearly 0% violation rate and the lowest median execution time. In contrast, other schedulers that overburden xavier-nx-0* devices or overly rely on the agx-xavier-00 device for task execution exhibit considerably worse violation rates. Regarding energy consumption, the results remain consistent with those of previous experiments.

LAMBDA_MAX=10, NUM_CLIENTS=8:

In this case there are eight (8) clients with bursting periods requests. The burst rate is a bit smaller than the previous experiment ($\lambda_{max} = 10$ instead of 15).

Upon examining Figure 5.3.24, it becomes readily apparent that the **SEOP_lambda_0_3_16384 scheduler demonstrates outstanding performance, maintaining a violation rate of less than 1%**. This performance is attributed to the dynamic allocation of 75% of the workloads to the agx-xavier-00 device and the remaining 25% to xavier-nx-0* devices, as illustrated in figure 5.3.22, when agx-xavier-00 is subjected to heavy loads. In contrast, the dummy schedulers, when they operate without external load, exhibit significantly less robustness, yielding violation rates that have increased from nearly 0% in previous experiment to 3-4%. Furthermore, the SEOP_lambda_0_3_16384 scheduler outperforms the dummy schedulers with external load (20% and 6.5% violation rates the random and round robin scheduler respectively). Other reinforcement learning (RL)-based schedulers, such as DEOP_DQN_gamma_099_71168, also deliver commendable performance. This is achieved by judiciously offloading a portion of requests to xavier-nx-0* devices, thereby preventing overutilization of agx-xavier-00 and resulting in lower violation rates. In contrast, schedulers that disproportionately rely on agx-xavier-00 to bear the load fail to handle the demands efficiently. For instance, DEOP_lambda_0_9_16384, SEOP_lambda_0_5_16384, SEOP_lambda_0_7_16384, and

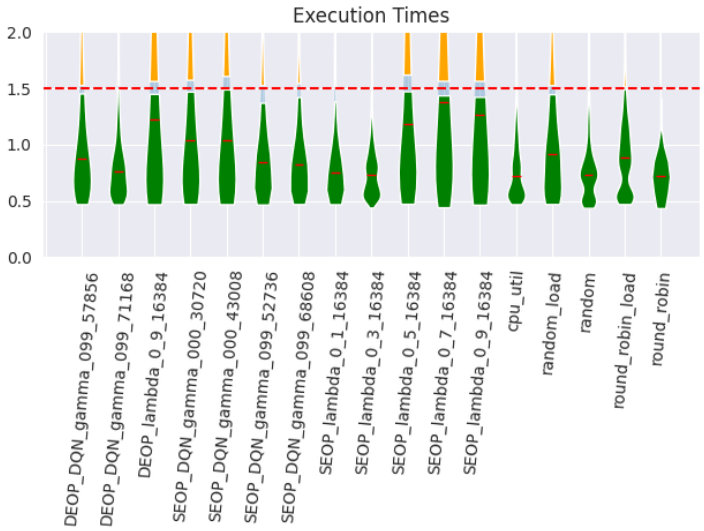


Figure 5.3.20: Whole Request Execution time for $\lambda_{max} = 10$ and $num\_clients = 8$

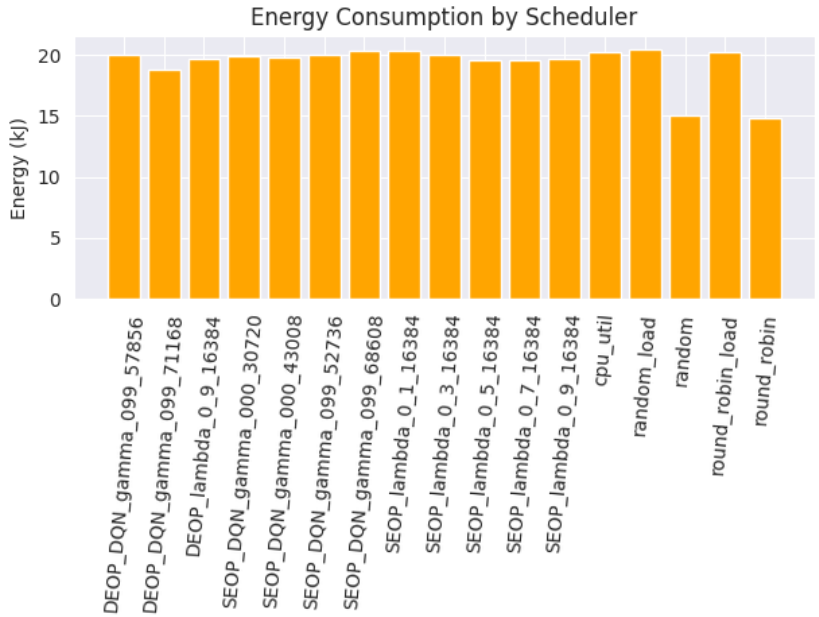


Figure 5.3.21: Whole Request Execution energy for $\lambda_{max} = 10$ and $num\_clients = 8$

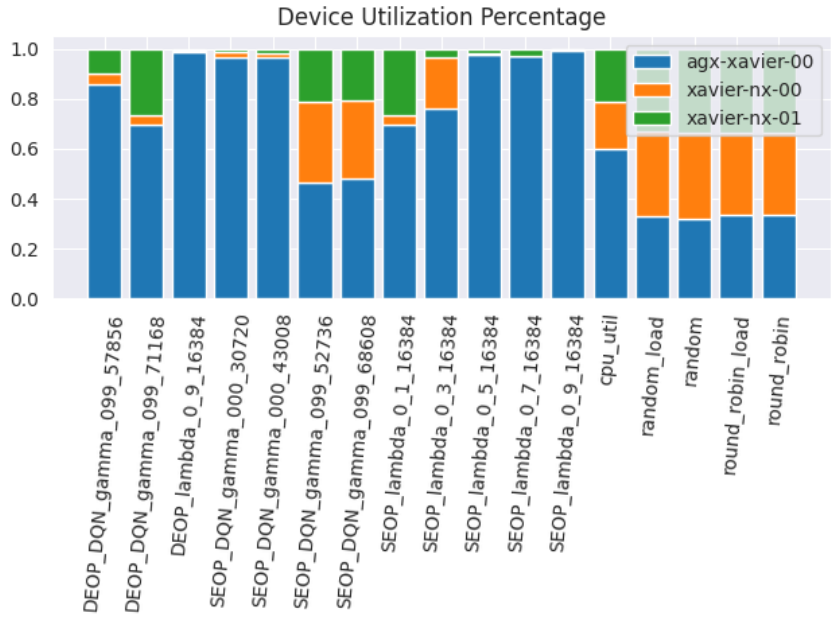


Figure 5.3.22: Whole Request Execution Device utilization percentage for $\lambda_{max} = 10$ and $num\_clients = 8$

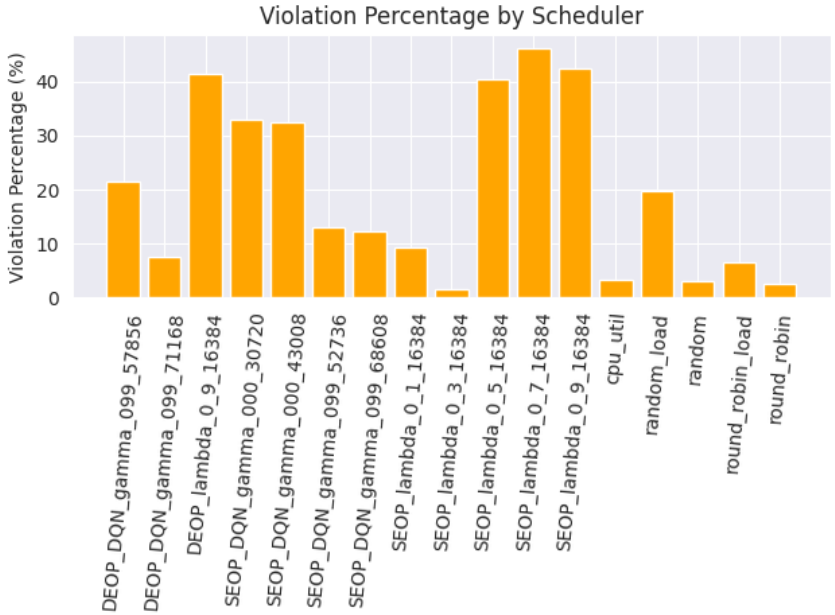


Figure 5.3.23: Whole Request Execution violation percentage for for $\lambda_{max} = 10$ and $num\_clients = 8$

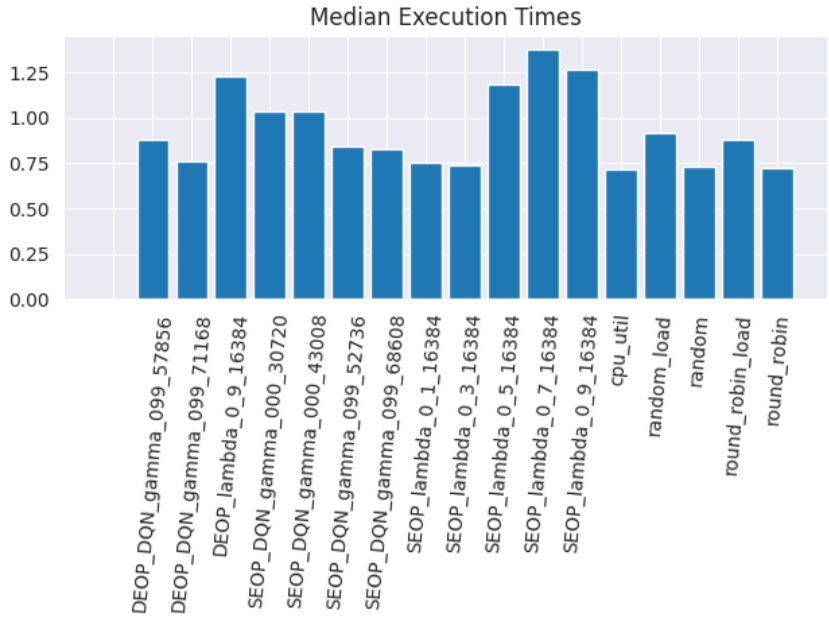


Figure 5.3.24: Median Request Execution time for $\lambda_{max} = 10$ and $num\_clients = 8$

SEOP_lambda_0_9_16384 exhibit violation rates of 40%. It is noteworthy that the cpu_util scheduler consistently maintains performance akin to the previous experiment, underscoring its robust nature. Lastly, as depicted in Figure 5.3.24, We observe that SEOP_lambda_0_3_16384, DEOP_DQN_gamma_099_71168, cpu_util, and the remaining dummy schedulers without workloads achieve the lowest median execution times. Notably, the energy consumption results remain consistent with those from the prior experiment.

Chapter 6

Conclusions

In this Diploma Thesis, a comprehensive framework has been established to enable the execution of Deep Neural Networks within a Serverless Edge Cluster. Additionally, a dynamic run-time scheduler has been developed, utilizing Reinforcement Learning techniques to intelligently manage the scheduling of DNN execution for incoming requests to the Edge Devices. The primary objective of this research endeavor has been to achieve optimal resource allocation within the Edge Devices while steadfastly adhering to Service Level Agreements (SLAs).

6.1 Discussion

In chapter 4.1, we develop a robust framework for the execution of Deep Neural Networks. This framework leverages the power of Kubernetes and Knative for deployment, harnessing the strengths of both technologies. Kubernetes, known for its robustness, forms the foundation of our framework, providing the reliability required for efficient DNN execution. Knative offers a range of valuable tools, including the Knative Pod Autoscaler, which dynamically adjusts the number of active pods in response to changing workloads for optimal resource utilization.

Furthermore, our framework offers the flexibility to support both layered and whole execution of Deep Neural Networks. This adaptability empowers users to choose the most suitable mode of execution, whether it be a layered approach for more granular control or whole execution for efficiency. The versatility of our framework equips it to address diverse use cases and aligns it with the evolving demands of edge computing environments.

In chapter 4.2, we delve into the development of a Reinforcement Learning scheduler designed to effectively manage the allocation of incoming requests to the devices. The primary objectives of this scheduler are twofold: to ensure that the Service Level Agreements (SLAs) are not violated and to minimize energy consumption within the system.

To address a spectrum of scenarios, We have designed schedulers for both layered and whole execution of Deep Neural Networks. This adaptability is pivotal, as it allows us to tailor our scheduling approach to the specific needs of diverse use cases, granting us the ability to optimize resource allocation and energy consumption across a wide range of edge computing environments.

6.2 Future Work

Various expansions and variants of our work may be proposed in the future.

- The layered execution suffers from the network time and the whole execution has a poor granularity control. To strike a balance between these two methods, We propose a hybrid approach. This approach involves the strategic placement of designated exit points within each Deep Neural Network, permitting scheduling decisions only at these predefined junctures. By implementing this strategy, We effectively minimize the cumulative network latency, as fewer exit points result in reduced network overhead. Simultaneously, We maintain a reasonable level of granularity control, ensuring that the computational process remains adaptable to the specific needs of the task at hand.
- An area of potential future development lies in the creation of a distributed version of the RL scheduler. The existing centralized server model proves to be less scalable in the context of a multi-device Edge Cluster. To address this scalability challenge, a distributed iteration of our scheduling mechanism offers a promising solution. This distributed approach entails the deployment of individual RL agents on each device, which collectively engage in a form of auction to determine the device responsible for executing a given request. This shift towards distribution can significantly enhance the scalability and adaptability of our scheduling system within complex and dynamic Edge Cluster environments.
- Algorithmic and coding optimizations, especially in monitor service due to the relatively high overhead.

Bibliography

[1] Hyuk-Jin Jeong et al. "IONN: Incremental Offloading of Neural Network Computations from Mobile Devices to Edge Servers". In: Oct. 2018, pp. 401–411. DOI: 10.1145/3267809.3267828.

[2] Stefanos Laskaridis et al. *SPINN: Synergistic Progressive Inference of Neural Networks over Device and Cloud*. Aug. 2020.

[3] Xueyu Hou, Yongjie Guan, and Tao Han. "Dystri: A Dynamic Inference based Distributed DNN Service Framework on Edge". In: Sept. 2023, pp. 625–634. DOI: 10.1145/3605573.3605598.

[4] Yanan Yang et al. "INFless: a native serverless system for low-latency, high-throughput inference". In: Feb. 2022, pp. 768–781. DOI: 10.1145/3503222.3507709.

[5] L. Lockhart et al. "Scission: Performance-driven and Context-aware Cloud-Edge Distribution of Deep Neural Networks". In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. Los Alamitos, CA, USA: IEEE Computer Society, 2020, pp. 257–268. DOI: 10.1109/UCC48980.2020.00044. URL:

[6] Andreas Kosmas Kakolyris et al. "RoaD-RuNNer: Collaborative DNN partitioning and offloading on heterogeneous edge systems". In: *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10137279.

[7] Liam Patterson et al. "A Hardware-Software Stack for Serverless Edge Swarms". In: Dec. 2021.

[8] Bin Wang, Ahmed Ali-Eldin, and Prashant Shenoy. "LaSS: Running Latency Sensitive Serverless Computations at the Edge". In: Apr. 2021.

[9] *What is Apache OpenWhisk?* URL: https://openwhisk.apache.org/.

[10] *Artificial Neural Networks*. URL: https://www.datasciencecentral.com/the-artificial-neural-networks-handbook-part-1/.

[11] Xun Ma et al. "Initial Margin Simulation with Deep Learning". In: *SSRN Electronic Journal* (Jan. 2019). DOI: 10.2139/ssrn.3357626.

[12] *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[13] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[14] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: (June 2014). DOI: 10.3115/v1/D14-1179.

[15] Yann Lecun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86 (Dec. 1998), pp. 2278 –2324. DOI: 10.1109/5.726791.

[16] Roohollah Amiri et al. "A Machine Learning Approach for Power Allocation in HetNets Considering QoS". In: (Mar. 2018).

[17] Christopher Watkins and Peter Dayan. "Technical Note: Q-Learning". In: *Machine Learning* 8 (May 1992), pp. 279–292. DOI: 10.1007/BF00992698.

[18] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL:

[19] *What is AWS Lambda?* URL: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html.

[20] *Azure Functions documentation*. URL: https://learn.microsoft.com/en-us/azure/azure-functions/.

[21] *Cloud Functions*. URL: https://cloud.google.com/functions.

[22] *Knative is an Open-Source Enterprise-level solution to build Serverless and Event Driven Applications*. URL: https://knative.dev/docs/.
[23] *Production-Grade Container Orchestration*. URL: https://kubernetes.io/.
[24] *Serverless Functions, Made Simple*. URL: https://www.openfaas.com/.
[25] *faas-netes - Serverless Functions For Kubernetes with OpenFaaS*. URL: https://github.com/openfaas/faas-netes/.
[26] *Edge vs Cloud Computing: Efficient Data Processing*. URL: https://semiconductor.samsung.com/emea/support/tools-resources/dictionary/edge-computing/.
[27] *Edge Computing Architecture*. URL: https://www.fsp-group.com/en/knowledge-app-42.html.
[28] *RasberryPi*. URL: https://www.raspberrypi.com/.
[29] *Nvidia Jetson*. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.
[30] *STM32 32-bit Arm Cortex MCUs*. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html.
[31] *Flask: A micro web framework written in Python*. URL: https://flask.palletsprojects.com/en/2.3.x/.
[32] *Numpy: The fundamental package for scientific computing with Python*. URL: https://numpy.org/.
[33] *Pytorch*. URL: https://pytorch.org/.
[34] *Creating a cluster with kubeadm*. URL: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/.
[35] *What is YAML?* URL: https://www.redhat.com/en/topics/automation/what-is-yaml.
[36] *Knative's KPA*. URL: https://knative.dev/docs/serving/autoscaling/kpa-specific/.
[37] *tegrastats Utility*. URL: https://docs.nvidia.com/drive/drive$_o s_5$.1.6.1L/$nvvib_d ocs$/$index.html page$/$DRIV E_O S_L inu$
[38] *perf: Linux profiling with performance counters*. URL: https://perf.wiki.kernel.org/.
[39] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *CoRR* abs/1602.01783 (2016). arXiv: 1602.01783. URL:
[40] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. eprint:
[41] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL:
[42] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: (July 2017).
[43] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL:
[44] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL:
[45] *Non-homogeneous Poisson Processes*. URL: https://stats.libretexts.org/Bookshelves/Probability$_T heory$/$Probabilit$ $homogeneous$_P oisson$_P rocesses$.
[46] *Simulating an inhomogeneous Poisson point process*. URL: https://hpaulkeeler.com/simulating-an-inhomogeneous-poisson-point-process/.
[47] *NVIDIA device plugin for Kubernetes*. URL: https://github.com/NVIDIA/k8s-device-plugin.
[48] *NVIDIA Jetpack*. URL: https://developer.nvidia.com/embedded/jetpack.
[49] *NVIDIA Docker*. URL: https://github.com/NVIDIA/nvidia-docker.
[50] *NVIDIA Container Toolkit*. URL: https://github.com/NVIDIA/nvidia-container-toolkit.
[51] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.
[52] *First neural network for beginners explained*. URL: https://www.v7labs.com/blog/neural-network-architectures-guide.
[53] Sepp Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.
[54] Qiong Liu and Ying Wu. "Supervised Learning". In: (Jan. 2012). DOI: 10.1007/978-1-4419-1428-6_451.

[55] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.
[56] Jiawei Zhang. *Gradient Descent based Optimization Algorithms for Deep Learning Models Training*. 2019. arXiv: 1903.03614 [cs.LG].
[57] *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.
[58] *Convolutions for Images*. URL: https://classic.d2l.ai/chapter$_c$*onvolutional* − *neural* − *networks/conv* − *layer.html*.
[59] *Pooling (CNN)*. URL: https://epynn.net/Pooling.html.
[60] *Fully-Connected Layer*. URL: https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html.
[61] .
[62] *Dropout Neural Network Layer In Keras Explained*. URL: https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab.
[63] *How to improve the model performance with Data Augmentation?* URL: https://datahacker.rs/tf-data-augmentation/.
[64] *Data Augmentation*. URL: https://paperswithcode.com/task/data-augmentation.
[65] Muddasar Naeem, Syed Rizvi, and Antonio Coronato. "A Gentle Introduction to Reinforcement Learning and its Application in Different Fields". In: *IEEE Access* 8 (Jan. 2020), pp. 209320–209344. DOI: 10.1109/ACCESS.2020.3038605.
[66] Yanbei Chen et al. *Semi-Supervised and Unsupervised Deep Visual Learning: A Survey*. 2022. arXiv: 2208.11296 [cs.CV].
[67] Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. "On the Complexity of Solving Markov Decision Problems". In: *CoRR* abs/1302.4971 (2013). arXiv: 1302.4971. URL:
[68] Tom Schaul et al. "Universal Value Function Approximators". In: July 2015.
[69] Michael L. Littman. "Value-function reinforcement learning in Markov games". In: *Cognitive Systems Research* 2.1 (2001), pp. 55–66. ISSN: 1389-0417. DOI: https://doi.org/10.1016/S1389-0417(01)00015-8. URL:
[70] Aviv Tamar, Sergey Levine, and Pieter Abbeel. "Value Iteration Networks". In: *CoRR* abs/1602.02867 (2016). arXiv: 1602.02867. URL:
[71] Matteo Pirotta et al. "Safe Policy Iteration". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 307–315. URL:
[72] Richard Sutton. "Learning to Predict by the Method of Temporal Differences". In: *Machine Learning* 3 (Aug. 1988), pp. 9–44. DOI: 10.1007/BF00115009.
[73] *Epsilon-Greedy Algorithm in Reinforcement Learning*. URL: https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/.
[74] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL:
[75] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL:
[76] Ziyu Wang, Nando de Freitas, and Marc Lanctot. "Dueling Network Architectures for Deep Reinforcement Learning". In: *CoRR* abs/1511.06581 (2015). arXiv: 1511.06581. URL:
[77] Matteo Hessel et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *CoRR* abs/1710.02298 (2017). arXiv: 1710.02298. URL:
[78] Alessandro Sebastianelli et al. "A Deep Q-Learning based approach applied to the Snake game". In: May 2021. DOI: 10.1109/MED51440.2021.9480232.
[79] Jan Peters. "Policy gradient methods". In: *Scholarpedia* 5 (Jan. 2010), p. 3698. DOI: 10.4249/scholarpedia.3698.
[80] John Schulman et al. "Trust Region Policy Optimization". In: *CoRR* abs/1502.05477 (2015). arXiv: 1502.05477. URL:
[81] Vijay Konda and John Tsitsiklis. "Actor-Critic Algorithms". In: *Society for Industrial and Applied Mathematics* 42 (Apr. 2001).

[82] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *CoRR* abs/1602.01783 (2016). arXiv: 1602.01783. URL:
[83] Elmar Diederichs. "Reinforcement Learning - A Technical Introduction". In: *Journal of Autonomous Intelligence* 2 (Aug. 2019), p. 25. DOI: 10.32629/jai.v2i2.45.
[84] *AlphaGo*. URL: https://www.deepmind.com/research/highlighted-research/alphago.
[85] *OpenAI Five defeats Dota 2 world champions*. URL: https://openai.com/research/openai-five-defeats-dota-2-world-champions.
[86] Oriol Vinyals et al. "StarCraft II: A New Challenge for Reinforcement Learning". In: *CoRR* abs/1708.04782 (2017). arXiv: 1708.04782. URL:
[87] B Ravi Kiran et al. *Deep Reinforcement Learning for Autonomous Driving: A Survey*. 2021. arXiv: 2002.00444 [cs.LG].
[88] Yunlong Song et al. "Autonomous Drone Racing with Deep Reinforcement Learning". In: *CoRR* abs/2103.08624 (2021). arXiv: 2103.08624. URL:
[89] Raghav Nagpal, Achyuthan Krishnan, and Hanshen Yu. *Reward Engineering for Object Pick and Place Training*. Jan. 2020.
[90] Mahsa Oroojeni Mohammad Javad et al. "A reinforcement learning-based method for management of type 1 diabetes: Exploratory study". en. In: *JMIR Diabetes* 4.3 (Aug. 2019), e12905.
[91] Tingting Zheng et al. "Learning how to detect: A deep reinforcement learning method for whole-slide melanoma histopathology images". In: *Computerized Medical Imaging and Graphics* 108 (2023), p. 102275. ISSN: 0895-6111. DOI: https://doi.org/10.1016/j.compmedimag.2023.102275. URL:
[92] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. "Deep Reinforcement Learning for De-Novo Drug Design". In: *CoRR* abs/1711.10907 (2017). arXiv: 1711.10907. URL:
[93] Mingyang Liu, Xiaotong Shen, and Wei Pan. "Deep reinforcement learning for personalized treatment recommendation". In: *Statistics in Medicine* 41.20 (June 2022), pp. 4034–4056. DOI: 10.1002/sim.9491. URL:
[94] Tidor-Vlad Pricope. "Deep Reinforcement Learning in Quantitative Algorithmic Trading: A Review". In: *CoRR* abs/2106.00123 (2021). arXiv: 2106.00123. URL:
[95] Gang Huang, Xiaohua Zhou, and Qingyang Song. *Deep reinforcement learning for portfolio management*. 2022. arXiv: 2012.13773 [q-fin.CP].
[96] Abdelali El Bouchti et al. "Fraud detection in banking using deep reinforcement learning". In: Aug. 2017, pp. 58–63. DOI: 10.1109/INTECH.2017.8102446.
[97] Roberto Maestre et al. "Reinforcement Learning for Fair Dynamic Pricing". In: *CoRR* abs/1803.09967 (2018). arXiv: 1803.09967. URL:
[98] Melanie Coggan. *Exploration and Exploitation in Reinforcement Learning*. Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University. 2004.
[99] Zihan Ding and Hao Dong. "Challenges of Reinforcement Learning". In: *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Ed. by Hao Dong, Zihan Ding, and Shanghang Zhang. Singapore: Springer Singapore, 2020, pp. 249–272. ISBN: 978-981-15-4095-0. DOI: 10.1007/978-981-15-4095-0_7. URL:
[100] Zhuangdi Zhu et al. "Transfer Learning in Deep Reinforcement Learning: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023), pp. 1–20. DOI: 10.1109/TPAMI.2023.3292075.
[101] Shangding Gu et al. *A Review of Safe Reinforcement Learning: Methods, Theory and Applications*. 2023. arXiv: 2205.10330 [cs.AI].
[102] *Gym*. URL: https://github.com/openai/gym.
[103] *Tensorflow: An end-to-end machine learning platform*. URL: https://www.tensorflow.org/.
[104] *Keras-rl*. URL: https://github.com/keras-rl/keras-rl.
[105] *StableBaselines*. URL: https://stable-baselines.readthedocs.io/en/master/.
[106] *StableBaselines3*. URL: https://stable-baselines3.readthedocs.io/en/master/.
[107] *Reinforcement Learning Library: pyqlearning*. URL: https://github.com/accel-brain/accel-brain-code/tree/master/Reinforcement-Learning.
[108] *Tensorforce: a TensorFlow library for applied reinforcement learning*. URL: https://tensorforce.readthedocs.io/en/latest/.

[109] *RLCoach*. URL: https://github.com/IntelLabs/coach.
[110] *TF-Agents is a library for reinforcement learning in TensorFlow*. URL: https://www.tensorflow.org/agents.
[111] *RLlib: Industry-Grade Reinforcement Learning*. URL: https://docs.ray.io/en/latest/rllib/index.html.
[112] Peter Mell. *The NIST Definition of Cloud Computing*.
[113] *Amazon EC2*. URL: https://aws.amazon.com/ec2/?nc1=h$_l s$.
[114] *Amazon S3*. URL: https://aws.amazon.com/s3/.
[115] *GMail*. URL: https://play.google.com/store/apps/details?id=com.google.android.gmhl=en$_U Sgl = USpli = 1$.
[116] *Docs*. URL: https://google-docs.en.softonic.com/.
[117] *AppEngine*. URL: https://cloud.google.com/appengine.
[118] *AppService*. URL: https://azure.microsoft.com/en-us/products/app-service/.
[119] Erwin Eyk et al. "Serverless is More: From PaaS to Present Cloud Computing". In: *IEEE Internet Computing* 22 (Sept. 2018), pp. 8–17. DOI: 10.1109/MIC.2018.053681358.
[120] Simon Eismann et al. "Serverless Applications: Why, When, and How?" In: (Sept. 2020).
[121] Davide Taibi, Josef Spillner, and Konrad Wawruch. "Serverless Where are we now and where are we heading?" In: *IEEE Software* 1 (Jan. 2021). DOI: 10.1109/MS.2020.3028708.
[122] Eric Jonas et al. "Cloud Programming Simplified: A Berkeley View on Serverless Computing". In: (Feb. 2019).
[123] Daniel Barcelona-Pons et al. "Stateful Serverless Computing with Crucial". In: *ACM Transactions on Software Engineering and Methodology* 31 (July 2022), pp. 1–38. DOI: 10.1145/3490386.
[124] *Deep Learning Inference*. URL: https://deci.ai/deep-learning-glossary/deep-learning-inference/.
[125] *Serverless Computing 101*. URL: https://hub.packtpub.com/serverless-computing-101/.
[126] *IBM Cloud Functions*. URL: https://cloud.ibm.com/functions/.
[127] *OpenLambda*. URL: https://github.com/open-lambda/open-lambda.
[128] *Amazon CloudWatch*. URL: https://aws.amazon.com/cloudwatch/.
[129] *What Is AWS CloudTrail?* URL: https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html.
[130] *Knative Serving*. URL: https://knative.dev/docs/serving/.
[131] *Knative Eventing - The Event-driven application platform for Kubernetes*. URL: https://knative.dev/docs/eventing/.
[132] *Docker*. URL: https://www.docker.com/.
[133] *NGINX*. URL: https://www.nginx.com/.
[134] *CouchDB*. URL: https://couchdb.apache.org/.
[135] *Apache Kafka*. URL: https://kafka.apache.org/.
[136] *What is Pub/Sub? The Publish/Subscribe model explained*. URL: https://ably.com/topic/pub-sub.
[137] *Red Hat OpenShift*. URL: https://www.redhat.com/en/technologies/cloud-computing/openshift.
[138] *An Architectural View of Apache OpenWhisk*. URL: https://thenewstack.io/behind-scenes-apache-openwhisk-serverless-platform/.
[139] *An Architectural View of OpenFaas*. URL: https://ericstoekl.github.io/faas/architecture/.
[140] Edward Oakes et al. "SOCK: Rapid Task Provisioning with Serverless-Optimized Containers". In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, July 2018, pp. 57–70. ISBN: 978-1-931971-44-7. URL:
[141] *Containerd*. URL: https://containerd.io/.
[142] *Introduction to gRPC*. URL: https://grpc.io/docs/what-is-grpc/introduction/.
[143] *Introducing runC: a lightweight universal container runtime*. URL: https://www.docker.com/blog/runc/.
[144] *Open Container Initiative*. URL: https://opencontainers.org/.
[145] *Docker and OCI Runtimes*. URL: https://medium.com/@avijitsarkar123/docker-and-oci-runtimes-a9c23a5646d6.

[146] *Kubernetes Components*. URL: https://kubernetes.io/docs/concepts/overview/components/.
[147] *cri-o*. URL: https://cri-o.io/.
[148] *Components of Kubernetes Architecture*. URL: https://techdozo.dev/kubernetes-architecture/.
[149] *About Knative's Revisions*. URL: https://knative.dev/docs/serving/revisions/.
[150] *Knative's Revision Traffic management*. URL: https://knative.dev/docs/serving/traffic-management/.
[151] *Knative's Scale Down to Zero*. URL: https://knative.dev/docs/serving/autoscaling/scale-to-zero/.
[152] *Knative Third Party Event Sources*. URL: https://knative.dev/docs/eventing/sources/third-party-sources.
[153] *Knative Sink*. URL: https://knative.dev/docs/eventing/sinks/.
[154] *Knative Source for Apache Kafka*. URL: https://knative.dev/docs/eventing/sources/kafka-source/.
[155] *About Knative Services*. URL: https://knative.dev/docs/serving/services/.
[156] *Knative Channels*. URL: https://knative.dev/docs/eventing/channels/.
[157] *Knative Brokers*. URL: https://knative.dev/docs/eventing/brokers/.
[158] *Knative Subscriptions*. URL: https://knative.dev/docs/eventing/channels/subscriptions/.
[159] *Knative Eventing Triggers*. URL: https://docs.triggermesh.io/1.25/brokers/triggers/.
[160] *Knative Flows*. URL: https://knative.dev/docs/eventing/flows/.
[161] *Knative Serving Routes example*. URL: https://www.baeldung.com/ops/knative-serverless.
[162] Keyan Cao et al. "An Overview on Edge Computing Research". In: *IEEE Access* PP (Jan. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.2991734.
[163] Irina Pustokhina et al. "An Effective Training Scheme for Deep Neural Network in Edge Computing Enabled Internet of Medical Things (IoMT) Systems". In: *IEEE Access* PP (June 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.3000322.
[164] Salam Hamdan, Moussa Ayyash, and Sufyan Almajali. "Edge-Computing Architectures for Internet of Things Applications: A Survey". In: *Sensors* 20.22 (2020). ISSN: 1424-8220. DOI: 10.3390/s20226441. URL:
[165] Jie Tang et al. "PI-Edge: A Low-Power Edge Computing System for Real-Time Autonomous Driving Services". In: *CoRR* abs/1901.04978 (2019). arXiv: 1901.04978. URL:
[166] Jinke Ren et al. "An Edge-Computing Based Architecture for Mobile Augmented Reality". In: *IEEE Network* PP (Jan. 2019), pp. 12–19. DOI: 10.1109/MNET.2018.1800132.
[167] Davide Calandra, Alberto Cannavamp;ograve; and Fabrizio Lamberti. *Improving AR-powered remote assistance: A new approach aimed to foster operator's autonomy and optimize the use of skilled resources - the International Journal of Advanced Manufacturing Technology*. 2021. URL:
[168] Denise D. Payán et al. "Telemedicine implementation and use in community health centers during COVID-19: Clinic personnel and patient perspectives". In: *SSM - Qualitative Research in Health* 2 (2022), p. 100054. ISSN: 2667-3215. DOI: https://doi.org/10.1016/j.ssmqr.2022.100054. URL:
[169] Cheng Feng et al. "Smart grid encounters edge computing: opportunities and applications". In: *Advances in Applied Energy* 1 (2021), p. 100006. ISSN: 2666-7924. DOI: https://doi.org/10.1016/j.adapen.2020.100006. URL:
[170] Xi Fang et al. "Smart Grid — The New and Improved Power Grid: A Survey". In: *IEEE Communications Surveys Tutorials* 14.4 (2012), pp. 944–980. DOI: 10.1109/SURV.2011.101911.00087.
[171] Samaresh Bera, Sudip Misra, and Joel J.P.C. Rodrigues. "Cloud Computing Applications for Smart Grid: A Survey". In: *IEEE Transactions on Parallel and Distributed Systems* 26.5 (2015), pp. 1477–1494. DOI: 10.1109/TPDS.2014.2321378.
[172] Weisong Shi et al. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.
[173] Rushit Dave, Naeem Seliya, and Nyle Siddiqui. "The Benefits of Edge Computing in Healthcare, Smart Cities, and IoT". In: *CoRR* abs/2112.01250 (2021). arXiv: 2112.01250. URL:

[174] Fatima Alshehri and Ghulam Muhammad. "A Comprehensive Survey of the Internet of Things (IoT) and AI-Based Smart Healthcare". In: *IEEE Access* 9 (2021), pp. 3660–3678. DOI: 10.1109/ACCESS.2020.3047960.

[175] Luca Greco et al. "Trends in IoT based solutions for health care: moving AI to the Edge". In: *Pattern Recognition Letters* 135 (May 2020). DOI: 10.1016/j.patrec.2020.05.016.

[176] Sujata Dash et al. "Edge and Fog Computing in Healthcare – A Review". In: *Scalable Computing: Practice and Experience* 20 (May 2019), pp. 191–206. DOI: 10.12694/scpe.v20i2.1504.

[177] Ping Zhang, Mimoza Durresi, and Arjan Durresi. "Multi-access edge computing aided mobility for privacy protection in Internet of Things". In: *Computing* 101 (July 2019), pp. 1–14. DOI: 10.1007/s00607-018-0639-0.

[178] Constandinos Mavromoustakis et al. "Socially Oriented Edge Computing for Energy Awareness in IoT Architectures". In: *IEEE Communications Magazine* 56 (July 2018), pp. 139–145. DOI: 10.1109/MCOM.2018.1700600.

[179] Junxia Li et al. "A Secured Framework for SDN-Based Edge Computing in IoT-Enabled Healthcare System". In: *IEEE Access* 8 (2020), pp. 135479–135490. DOI: 10.1109/ACCESS.2020.3011503.

[180] Md. Zia Uddin. "A wearable sensor-based activity prediction system to facilitate edge computing in smart healthcare system". In: *Journal of Parallel and Distributed Computing* 123 (2019), pp. 46–53. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2018.08.010. URL:

[181] Gopala Krishna Sriram. "EDGE COMPUTING VS. CLOUD COMPUTING: AN OVERVIEW OF BIG DATA CHALLENGES AND OPPORTUNITIES FOR LARGE ENTERPRISES". In: *International Research Journal of Modernization in Engineering Technology and science* 107 (Mar. 2022). DOI: https://doi.org/10.56726/irjmets.

[182] *Microsoft Azure*. URL: https://azure.microsoft.com/.

[183] *Amazon Web Services*. URL: https://aws.amazon.com/.

[184] *cloudevents: A micro web framework written in Python*. URL: https://cloudevents.io/.

[185] *NVIDIA NGC: AI Development Catalog*. URL: https://catalog.ngc.nvidia.com/?filters=orderBy=weightPopularDE

[186] *Global Interprete rLock*. URL: https://wiki.python.org/moin/GlobalInterpreterLock.