NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF INFORMATION TRANSMISSION SYSTEMS
AND MATERIAL TECHNOLOGY

Efficient Deep Learning in Mobile and Embedded Computing Environments

Ph.D. Dissertation

Ioannis Panopoulos

Athens, April 2025

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Αποδοτική Βαθιά Μάθηση σε Κινητά και Ενσωματωμένα Υπολογιστικά Περιβάλλοντα

Διδακτορική Διατριβή

Ιωάννης Πανόπουλος

Αθήνα, Απρίλιος 2025

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND
COMPUTER ENGINEERING
DIVISION OF INFORMATION TRANSMISSION SYSTEMS
AND MATERIAL TECHNOLOGY

Efficient Deep Learning in Mobile and Embedded Computing Environments

Ph.D. Dissertation

Ioannis Panopoulos

Advisory Committee: Iakovos S. Venieris
Dimitra Kaklamani
Nectarios Koziris

Approved by the seven-member Examination Committee on April 29, 2025.

.....	.....	.....
Iakovos S. Venieris	Dimitra Kaklamani	Nectarios Koziris
NTUA Professor	NTUA Professor	NTUA Professor

.....	.....	.....	.....
Emmanouel Varvarigos	Theodora Varvarigou	Dimitrios Askounis	Dimitrios Vergados
NTUA Professor	NTUA Professor	NTUA Professor	UniPi Professor

Athens, April 2025

.....
Ioannis Panopoulos
Doctor of Electrical and Computer Engineering, National Technical University of Athens

Περίληψη

Η βαθιά μάθηση έχει μετασχηματίσει ριζικά το πεδίο της τεχνητής νοημοσύνης, προσφέροντας σημαντικές προόδους σε τομείς όπως η επεξεργασία φυσικής γλώσσας, η όραση υπολογιστών και η αυτόνομη λήψη αποφάσεων. Ωστόσο, η συνεχώς αυξανόμενη πολυπλοκότητα των μοντέλων συνεπάγεται σημαντικές υπολογιστικές απαιτήσεις, καθιστώντας αναγκαία την αξιοποίηση ισχυρών υποδομών νέφους. Η εξάρτηση αυτή από κεντρικοποιημένους πόρους δημιουργεί περιορισμούς σε ό,τι αφορά την καθυστέρηση, την ιδιωτικότητα και τη διαθεσιμότητα, αποτελώντας εμπόδιο για την ανάπτυξη εφαρμογών τεχνητής νοημοσύνης σε κινητά και ενσωματωμένα συστήματα.

Η παρούσα διατριβή διερευνά τη διασταύρωση βαθιάς μάθησης και αποδοτικότητας σε περιβάλλοντα με περιορισμένους υπολογιστικούς πόρους, με στόχο τη διαμόρφωση ενός ολιστικού πλαισίου για την αποδοτική ανάπτυξη και εκτέλεση συστημάτων τεχνητής νοημοσύνης στις παρυφές του δικτύου. Η έρευνα εστιάζει σε τρεις επιμέρους μελέτες: (α) την ανάπτυξη του CARIn, ενός προσαρμοστικού πλαισίου για την εκτέλεση πολλαπλών νευρωνικών δικτύων σε ετερογενείς κινητές συσκευές, με χρήση τεχνικών βελτιστοποίησης πολλαπλών στόχων· (β) την ενδελεχή αξιολόγηση και προσαρμογή μοντέλων μετασχηματιστών για κινητά περιβάλλοντα, μέσω αρχιτεκτονικών βελτιστοποιήσεων χαμηλού κόστους· και (γ) τη σχεδίαση του A-THENA, ενός αποδοτικού συστήματος ανίχνευσης εισβολών σε IoT δίκτυα, βασισμένου σε μετασχηματιστές με χρονικά ευαίσθητες κωδικοποιήσεις θέσης.

Μέσα από αυτές τις συνεισφορές, η διατριβή διαμορφώνει ένα ολοκληρωμένο πλαίσιο για αποδοτική βαθιά μάθηση σε κινητά και ενσωματωμένα υπολογιστικά περιβάλλοντα. Εξερευνώντας τη διασύνδεση μεταξύ της βελτιστοποίησης μοντέλων, της προσαρμογής στο υλικό και των απαιτήσεων πραγματικών εφαρμογών, η έρευνα γεφυρώνει το χάσμα μεταξύ των τελευταίων εξελίξεων στην τεχνητή νοημοσύνη και της πρακτικής τους ανάπτυξης. Τα ευρήματα υπογραμμίζουν ότι η αποδοτικότητα δεν αποτελεί απλώς στόχο βελτιστοποίησης, αλλά θεμελιώδη παράγοντα για το μέλλον της τεχνητής νοημοσύνης, διασφαλίζοντας ότι οι τεχνολογίες βαθιάς μάθησης μπορούν να λειτουργούν απρόσκοπτα, βιώσιμα και ευφυώς σε ένα ευρύ φάσμα υπολογιστικών πλατφορμών.

Λέξεις-Κλειδιά

βαθιά μάθηση · τοπική συμπερασματολογία · κινητός υπολογισμός · ενσωματωμένος υπολογισμός · αποδοτική τεχνητή νοημοσύνη · ετερογένεια · βελτιστοποίηση · μοντέλα μετασχηματιστών · ανάπτυξη στις παρυφές · ανίχνευση εισβολών · διαδίκτυο των πραγμάτων

Abstract

Deep learning has fundamentally transformed the field of artificial intelligence, enabling significant advancements in areas such as natural language processing, computer vision, and autonomous decision-making. However, the ever-increasing complexity of modern models entails substantial computational demands, rendering the use of powerful cloud infrastructures essential. This dependence on centralized computing introduces limitations in terms of latency, privacy, and availability, posing a challenge for the deployment of AI applications on mobile and embedded systems.

This dissertation investigates the intersection of deep learning and efficiency in resource-constrained environments, with the aim of establishing a holistic framework for the efficient development and execution of AI systems at the network edge. The research focuses on three key studies: (a) the development of `CARIn`, an adaptive inference framework designed to execute multiple neural networks on heterogeneous mobile devices using multi-objective optimization techniques; (b) the thorough evaluation and adaptation of Transformer models for mobile environments through low-cost architectural and hardware-aware optimizations; and (c) the design of `A-THENA`, an efficient intrusion detection system for IoT networks, based on Transformers with time-aware positional encodings.

Through these contributions, this dissertation formulates a comprehensive approach to efficient deep learning in mobile and embedded computing environments. By exploring the interplay between model optimization, hardware adaptation, and real-world application requirements, this work bridges the gap between cutting-edge AI research and its practical deployment. The findings emphasize that efficiency is not merely an optimization objective but a foundational enabler for the future of AI, ensuring that deep learning technologies can operate seamlessly, sustainably, and intelligently across a wide range of computing platforms.

Keywords

Deep learning; On-device inference; Mobile computing; Embedded computing; Efficient AI; Heterogeneity; Optimization; Transformer models; Edge deployment; Intrusion detection; Internet of Things

Εκτεταμένη Περίληψη

Α Εισαγωγή

Οι πρόσφατες εξελίξεις στην τεχνητή νοημοσύνη (artificial intelligence – AI) καθοδηγούνται από την αναζήτηση ισορροπίας μεταξύ υψηλής επίδοσης και αποδοτικότητας, με κάθε σημαντική ανακάλυψη στη βαθιά μάθηση να εισάγει νέες δυνατότητες, αλλά ταυτόχρονα και νέες προκλήσεις. Ενώ η βαθιά μάθηση έχει φέρει επανάσταση σε τομείς όπως η επεξεργασία φυσικής γλώσσας (natural language processing – NLP), η όραση υπολογιστών (computer vision) και τα αυτόνομα συστήματα, η ταχεία πρόοδός της έρχεται με το κόστος αυξημένων υπολογιστικών απαιτήσεων. Τα σύγχρονα μοντέλα απαιτούν τεράστιες ποσότητες μνήμης, επεξεργαστικής ισχύος και ενέργειας, με αποτέλεσμα να εξαρτώνται σε μεγάλο βαθμό από υποδομές του νέφους (cloud). Ωστόσο, οι εφαρμογές απαιτούν όλο και περισσότερο λύσεις τεχνητής νοημοσύνης που λειτουργούν αποτελεσματικά σε πραγματικό χρόνο, πιο κοντά στον χρήστη, χωρίς υπερβολική εξάρτηση από κεντρικοποιημένους υπολογιστικούς πόρους.

Αυτό έχει τονίσει τη σημασία του κινητού και του ενσωματωμένου υπολογισμού, που παρουσιάζουν μια μοναδική πρόκληση: τη μετατόπιση της εστίασης από την «απεριόριστη» υπολογιστική ισχύ του νέφους στην αποδοτικότητα και την προσαρμοστικότητα. Αυτά τα περιβάλλοντα, συμπεριλαμβανομένων συσκευών όπως smartphones, αισθητήρων IoT και εξειδικευμένου υλικού παρυφών, απαιτούν λύσεις τεχνητής νοημοσύνης που μπορούν να λειτουργούν υπό σοβαρούς περιορισμούς πόρων, απαιτώντας τεχνικές βελτιστοποίησης που διατηρούν την ακρίβεια μειώνοντας ταυτόχρονα το μέγεθος, την κατανάλωση ενέργειας και την υπολογιστική πολυπλοκότητα. Η ανάπτυξη της βαθιάς μάθησης σε τέτοια περιβάλλοντα ξεκλειδώνει μετασχηματιστικές δυνατότητες, από τη λήψη αποφάσεων σε πραγματικό χρόνο και την βελτιωμένη ιδιωτικότητα έως πιο ανθεκτικά και ανεξάρτητα συστήματα τεχνητής νοημοσύνης.

Ωστόσο, προκύπτει η ανάγκη να ξεπεραστούν σημαντικές προκλήσεις, όπως η βελτιστοποίηση της εκτέλεσης σε ετερογενές υλικό, η διαχείριση της κατανάλωσης ενέργειας και η ενεργοποίηση της προσαρμοστικής μάθησης εντός περιορισμένων ορίων. Η παρούσα διατριβή τονίζει τους περιορισμούς των υπαρχουσών ερευνητικών προσπαθειών, όπως είναι η έλλειψη ενοποιημένων πλαισίων για συμπερασματολογία σε κινητές συσκευές, ο περιορισμένος βαθμός βελτιστοποίησης μοντέλων αιχμής—όπως οι μετασχηματιστές—για ανάπτυξη στις παρυφές του δικτύου και οι λιγότερο διερευνημένες εφαρμογές στα δίκτυα υπολογιστών και την κυβερνοασφάλεια, και παρουσιάζει νέες λύσεις που γεφυρώνουν το χάσμα μεταξύ έρευνας και πρακτικής ανάπτυξης σε κινητά και ενσωματωμένα υπολογιστικά περιβάλλοντα, διασφαλίζοντας την απρόσκοπτη ενσωμάτωση της τεχνητής νοημοσύνης στην καθημερινή ζωή.

Β Θεωρητικό Υπόβαθρο

Β.1 Βαθιά Μάθηση

Η βαθιά μάθηση (deep learning – DL) έχει αναδειχθεί ως ένα μετασχηματιστικό υποσύνολο της τεχνητής νοημοσύνης, ξεχωρίζοντας από την παραδοσιακή μηχανική μάθηση (machine learning – ML) μέσω της ικανότητάς της να εξάγει αυτόνομα ιεραρχικά μοτίβα από ακατέργαστα δεδομένα χρησιμοποιώντας τεχνητά νευρωνικά δίκτυα (artificial neural networks). Σε αντίθεση με τους αλγορίθμους μηχανικής μάθησης που βασίζονται σε χειροκίνητα σχεδιασμένα χαρακτηριστικά και τεχνογνωσία τομέα, η βαθιά μάθηση επιτρέπει

την αυτοματοποιημένη εξαγωγή χαρακτηριστικών, την επεκτασιμότητα, και τη γενίκευση σε διάφορες διεργασίες.

Η απαρχή της εντοπίζεται στη δεκαετία του 1950 με την εισαγωγή του perceptron, ενός απλού νευρωνικού μοντέλου που διέθετε βασικές ικανότητες μάθησης. Ωστόσο, η αδυναμία του να επιλύσει μη γραμμικά διαχωρίσιμα προβλήματα οδήγησε σε αμφισβήτηση της κατεύθυνσης του πεδίου και σε σταδιακή μείωση του ερευνητικού ενδιαφέροντος για αρκετές δεκαετίες. Η επιστημονική δραστηριότητα αναζωπυρώθηκε σημαντικά τη δεκαετία του 1980, με την εισαγωγή των perceptrons πολλαπλών επιπέδων και την αξιοποίηση του αλγορίθμου οπίσθιας διάδοσης, ο οποίος επέτρεψε την αποτελεσματική εκπαίδευση των νευρωνικών δικτύων μέσω της μεθόδου καθόδου κλίσης. Η πρόοδος αυτή κατέστησε δυνατή τη μοντελοποίηση πιο σύνθετων μη γραμμικών σχέσεων και σηματοδότησε την απαρχή μιας νέας ερευνητικής δυναμικής στον χώρο των τεχνητών νευρωνικών δικτύων. Ωστόσο, λόγω υπολογιστικών περιορισμών, η αξιοποίηση των βαθιών δικτύων παρέμεινε περιορισμένη μέχρι και τη δεκαετία του 2010, όταν οι εξελίξεις στο υλικό, η διαθεσιμότητα συνόλων δεδομένων μεγάλης κλίμακας και οι αρχιτεκτονικές βελτιώσεις κατέστησαν εφικτή την αξιοποίηση του πλήρους δυναμικού της βαθιάς μάθησης. Ως αποτέλεσμα, η βαθιά μάθηση εδραιώθηκε ως κυρίαρχη τεχνολογία στο πεδίο των σύγχρονων εφαρμογών τεχνητής νοημοσύνης.

Με την πάροδο του χρόνου, οι αρχιτεκτονικές βαθιάς μάθησης εξελίχθηκαν σημαντικά, ώστε να ανταποκρίνονται σε διαφορετικούς τύπους δεδομένων και σε αυξανόμενες υπολογιστικές απαιτήσεις. Αν και τα πρώιμα μοντέλα, όπως τα perceptrons πολλαπλών επιπέδων, αποτέλεσαν το θεμέλιο της βαθιάς μάθησης, παρουσίασαν περιορισμούς ως προς την επεκτασιμότητα και την ικανότητα διαχείρισης σύνθετων δομών πληροφορίας. Οι επακόλουθες εξελίξεις οδήγησαν στην ανάπτυξη εξειδικευμένων αρχιτεκτονικών, όπως τα συνελικτικά νευρωνικά δίκτυα (convolutional neural networks – CNNs) για την ανάλυση χωρικών δεδομένων και τα επαναληπτικά νευρωνικά δίκτυα (recurrent neural networks – RNNs) για την επεξεργασία ακολουθιακών πληροφοριών, ενισχύοντας σημαντικά τις επιδόσεις σε εφαρμογές επεξεργασίας εικόνας και φυσικής γλώσσας. Πιο πρόσφατα, τα μοντέλα μετασχηματιστών έχουν φέρει επανάσταση σε πολλούς τομείς της τεχνητής νοημοσύνης, καθιερώνοντας νέα πρότυπα επίδοσης ιδίως στην επεξεργασία φυσικής γλώσσας και στην όραση υπολογιστών.

Μετασχηματιστές

Οι μετασχηματιστές (Transformers) έχουν επιφέρει θεμελιώδη αλλαγή στον χώρο της βαθιάς μάθησης, υπερβαίνοντας τους περιορισμούς των επαναληπτικών μοντέλων, τα οποία δυσκολεύονταν στην αποτύπωση εξαρτήσεων μεγάλης εμβέλειας σε ακολουθιακά δεδομένα. Εισαχθείσα από τους Vaswani et al. το 2017, η αρχιτεκτονική του μετασχηματιστή καταργεί τις επαναληπτικές συνδέσεις υπέρ ενός πλαισίου πλήρως βασισμένου στον μηχανισμό προσοχής, επιτρέποντας παράλληλο υπολογισμό και βελτιωμένη αποδοτικότητα.

Στον πυρήνα της, η αρχιτεκτονική ακολουθεί μια δομή κωδικοποιητή-αποκωδικοποιητή, όπου ο κωδικοποιητής μετασχηματίζει την είσοδο σε αναπαραστάσεις που βασίζονται στα συμφραζόμενα, ενώ ο αποκωδικοποιητής αξιοποιεί αυτές τις αναπαραστάσεις για την παραγωγή της εξόδου. Βασικές καινοτομίες, όπως ο μηχανισμός αυτοπροσοχής πολλαπλών κεφαλών (multi-head self-attention), οι κωδικοποιήσεις θέσης (positional encodings), και τα δίκτυα εμπρόσθιας τροφοδοσίας, καθιστούν τον μετασχηματιστή ιδιαίτερα επεκτάσιμο και ευπροσάρμοστο σε πλήθος εφαρμογών.

Οι μετασχηματιστές έχουν θέσει νέα πρότυπα επίδοσης στην επεξεργασία φυσικής γλώσσας, αποτελώντας τη βάση υπερσύγχρονων μοντέλων, όπως τα Gemini και GPT-4. Παράλληλα, έχουν διεισδύσει δυναμικά στον χώρο της όρασης υπολογιστών, μέσω των

οπτικών μετασχηματιστών (Vision Transformers – ViTs), καθώς και σε πολυτροπικές εφαρμογές τεχνητής νοημοσύνης. Η ικανότητά τους να μοντελοποιούν πολύπλοκες εξαρτήσεις και να επεξεργάζονται μακροχρόνιες ακολουθίες με αποτελεσματικότητα έχει εδραιώσει τους μετασχηματιστές ως την κυρίαρχη αρχιτεκτονική της σύγχρονης τεχνητής νοημοσύνης. Οι συνεχιζόμενες εξελίξεις στην υποδομή και στις παραλλαγές τους υπόσχονται περαιτέρω πρόοδο σε ζητήματα επεκτασιμότητας, αποδοτικότητας και εύρους εφαρμογών.

Β.2 Υπολογιστική Παρυφών

Η υπολογιστική παρυφών (edge computing) έχει αναδειχθεί σε θεμελιώδες παράδειγμα της σύγχρονης υπολογιστικής, προσφέροντας μια αποτελεσματική απάντηση στους περιορισμούς των παραδοσιακών αρχιτεκτονικών που βασίζονται αποκλειστικά στο υπολογιστικό νέφος. Μετατοπίζοντας τους υπολογιστικούς πόρους εγγύτερα στην πηγή παραγωγής των δεδομένων, η υπολογιστική παρυφών επιτρέπει την επεξεργασία της πληροφορίας σε τοπικό επίπεδο, μειώνοντας την εξάρτηση από τη συνδεσιμότητα και τη μεταφορά δεδομένων σε απομακρυσμένα κέντρα δεδομένων. Αν και η υπολογιστική νέφους (cloud computing) έχει επαναπροσδιορίσει την αποθήκευση και την επεξεργασία δεδομένων σε μεγάλη κλίμακα, η έμφυτη εξάρτησή της από δίκτυα υψηλής ταχύτητας και κεντρικοποιημένη επεξεργασία εισάγει σημαντικές προκλήσεις, όπως καθυστερήσεις, περιορισμούς στο εύρος ζώνης, αυξημένους κινδύνους ασφάλειας και μειωμένη αξιοπιστία σε κινητά ή απομακρυσμένα περιβάλλοντα.

Η υπολογιστική παρυφών μετριάζει αυτά τα προβλήματα επιτρέποντας την επεξεργασία δεδομένων σε πραγματικό χρόνο απευθείας στις συσκευές των χρηστών ή σε κόμβους που βρίσκονται κοντά τους. Αυτή η προσέγγιση είναι ιδιαίτερα χρήσιμη σε εφαρμογές που απαιτούν εξαιρετικά χαμηλή καθυστέρηση, όπως τα αυτόνομα οχήματα, ο βιομηχανικός αυτοματισμός και η απομακρυσμένη παρακολούθηση στην υγειονομική περίθαλψη. Η έλευση των δικτύων 5G έχει ενισχύσει περαιτέρω την υιοθέτηση της υπολογιστικής παρυφών, προσφέροντας συνδεσιμότητα με εξαιρετικά χαμηλή καθυστέρηση και υψηλό εύρος ζώνης. Αυτό την καθιστά αναπόσπαστο συστατικό των αναδυόμενων τεχνολογιών όπως οι έξυπνες πόλεις, η επαυξημένη και εικονική πραγματικότητα, καθώς και τα οικοσυστήματα του διαδικτύου των πραγμάτων, στα οποία η αποδοτική και έγκαιρη επεξεργασία δεδομένων αποτελεί κρίσιμο παράγοντα επιτυχίας.

Κινητός Υπολογισμός

Ο κινητός υπολογισμός (mobile computing) επιτρέπει την απρόσκοπτη και σε πραγματικό χρόνο πρόσβαση σε υπολογιστικούς πόρους μέσω φορητών και ασύρματων συσκευών, προάγοντας τη συνδεσιμότητα ανεξαρτήτως τοποθεσίας. Αποτελεί ένα πολυδιάστατο τεχνολογικό πεδίο που περιλαμβάνει smartphones, tablets, φορετές συσκευές, καθώς και υπολογιστικά συστήματα ενσωματωμένα σε οχήματα, τα οποία αξιοποιούν ασύρματες τεχνολογίες επικοινωνίας όπως Wi-Fi, κυψελωτά δίκτυα (4G, 5G, B5G) και Bluetooth. Τα βασικά δομικά στοιχεία του κινητού υπολογισμού περιλαμβάνουν: (α) κινητές συσκευές σχεδιασμένες με γνώμονα τη φορητότητα και την ενεργειακή αποδοτικότητα, (β) ασύρματα δίκτυα που επιτρέπουν αξιόπιστη και ταχεία μετάδοση δεδομένων, και (γ) υπηρεσίες υπολογιστικού νέφους που ενισχύουν την υπολογιστική ικανότητα των συσκευών και παρατείνουν τη διάρκεια ζωής της μπαταρίας μέσω απομακρυσμένης επεξεργασίας.

Οι πρόσφατες εξελίξεις, όπως η υπολογιστική παρυφών πολλαπλής πρόσβασης (multi-access edge computing – MEC), ενισχύουν περαιτέρω την αποδοτικότητα του κινητού υπολογισμού, μειώνοντας ακόμη περισσότερο την καθυστέρηση και επιτρέποντας την τοπική κατανομή των υπολογιστικών πόρων στις παρυφές του δικτύου. Παράλληλα, τεχνολογίες όπως οι υπηρεσίες που βασίζονται στην τοποθεσία και τα συστήματα πληρωμών μέσω κινητών

συσκευών έχουν επεκτείνει σημαντικά τις δυνατότητες του πεδίου. Ο κινητός υπολογισμός έχει μεταμορφώσει κρίσιμους τομείς, όπως η οικονομία, η υγειονομική περίθαλψη και η ψυχαγωγία, επιτρέποντας την παροχή εξατομικευμένων υπηρεσιών, τη λήψη αποφάσεων σε πραγματικό χρόνο και την υλοποίηση ασφαλών και ταχέων συναλλαγών, αναδεικνύοντάς τον ως κεντρικό παράγοντα της σύγχρονης ψηφιακής εποχής.

Ενσωματωμένος Υπολογισμός

Ο ενσωματωμένος υπολογισμός (embedded computing) αναφέρεται σε εξειδικευμένα υπολογιστικά συστήματα που έχουν σχεδιαστεί για την εκτέλεση προκαθορισμένων λειτουργιών, συχνά σε πραγματικό χρόνο, εντός αυτόνομων ή ημιαυτόνομων συσκευών με ελάχιστη ή καθόλου ανθρώπινη παρέμβαση. Τα ενσωματωμένα συστήματα δίνουν έμφαση στην ενεργειακή αποδοτικότητα, την αξιοπιστία και τη λειτουργική σταθερότητα, γεγονός που τα καθιστά κρίσιμα σε εφαρμογές όπως ο βιομηχανικός αυτοματισμός, ο έλεγχος συστημάτων σε οχήματα, η υγειονομική περίθαλψη και τα ηλεκτρονικά είδη ευρείας κατανάλωσης.

Σε αντίθεση με τους υπολογιστές γενικής χρήσης, τα ενσωματωμένα συστήματα είναι στενά ενοποιημένα με το φυσικό τους υλικό, συμπεριλαμβανομένων αισθητήρων και μονάδων επικοινωνίας, ώστε να εκτελούν με ακρίβεια συγκεκριμένες λειτουργίες. Η αυστηρή σύνδεση λογισμικού και υλικού επιτρέπει την υλοποίηση αποδοτικών και ασφαλών υπολογιστικών λύσεων με περιορισμένους πόρους. Για παράδειγμα, στον βιομηχανικό τομέα, ο ενσωματωμένος υπολογισμός χρησιμοποιείται εκτενώς για προγνωστική συντήρηση, έλεγχο ποιότητας και βελτιστοποίηση λειτουργικών διαδικασιών, ενώ στον κλάδο της αυτοκινητοβιομηχανίας, αξιοποιείται για κρίσιμες λειτουργίες όπως η ανίχνευση σύγκρουσης, ο προσαρμοστικός έλεγχος πορείας και η αυτόνομη πλοήγηση.

Β.3 Ευφυής Υπολογιστική Παρυφών

Η ευφυής υπολογιστική παρυφών (edge intelligence – EI) συνιστά το σημείο σύγκλισης της τεχνητής νοημοσύνης και της υπολογιστικής παρυφών, επιτρέποντας την εκτέλεση ευφυών εφαρμογών με δυνατότητα λήψης αποφάσεων σε πραγματικό χρόνο, τοπικά, χωρίς εξάρτηση από κεντρικές υποδομές νέφους. Σε αντίθεση με τα παραδοσιακά συστήματα τεχνητής νοημοσύνης που βασίζονται στο νέφος και ενδέχεται να αντιμετωπίζουν καθυστερήσεις, κινδύνους απορρήτου και αυξημένη χρήση εύρους ζώνης, η ευφυής υπολογιστική παρυφών πραγματοποιεί την επεξεργασία των δεδομένων πλησίον της πηγής τους. Αυτή η προσέγγιση διασφαλίζει ταχύτερους χρόνους απόκρισης, αυξημένη ασφάλεια και καλύτερη διαχείριση των διαθέσιμων πόρων επικοινωνίας.

Ένα ιδιαίτερο χαρακτηριστικό αυτής της προσέγγισης είναι η δυνατότητα ενσωμάτωσης συλλογικών σχημάτων μάθησης, όπως η συνεργατική μάθηση (federated learning), που επιτρέπουν τη συνεχή βελτίωση των μοντέλων τεχνητής νοημοσύνης χωρίς την ανάγκη αποστολής ευαίσθητων δεδομένων σε απομακρυσμένους διακομιστές. Έτσι, η ευφυής υπολογιστική παρυφών καθίσταται ιδανική για εφαρμογές που απαιτούν προσαρμοστικότητα και διατήρηση του απορρήτου.

Τα κύρια χαρακτηριστικά της περιλαμβάνουν την ικανότητα εξατομίκευσης εμπειριών χρήστη, την πρόβλεψη αποτελεσμάτων και τη λήψη αποφάσεων με αυτονομία. Επιπλέον, διευκολύνει πιο φυσικές και ανθρωποκεντρικές αλληλεπιδράσεις, βελτιώνοντας τη λειτουργικότητα φωνητικών βοηθών, διεπαφών χρήστη και συστημάτων αναγνώρισης συναισθημάτων, προσφέροντας μια πιο διαισθητική εμπειρία. Η βιωσιμότητα αποτελεί επίσης θεμελιώδες πλεονέκτημα, καθώς η τοπική επεξεργασία και η ευφυής διαχείριση πόρων μειώνουν σημαντικά την κατανάλωση ενέργειας. Το γεγονός αυτό καθιστά την ευφυή

υπολογιστική παρυφών ιδιαίτερα κατάλληλη για τομείς όπως τα έξυπνα ενεργειακά δίκτυα, η γεωργία ακριβείας και οι περιβαλλοντικά ευαίσθητες εφαρμογές.

Β.4 Τοπική Συμπερασματολογία

Η τοπική συμπερασματολογία (on-device inference) αναφέρεται στην εκτέλεση μοντέλων βαθιάς μάθησης απευθείας σε συσκευές παρυφών, όπως smartphones, ενσωματωμένα συστήματα και συσκευές IoT, χωρίς την ανάγκη σύνδεσης με απομακρυσμένες υποδομές. Σε αντίθεση με τη φάση εκπαίδευσης, η οποία είναι ιδιαίτερα απαιτητική σε υπολογιστικούς πόρους και συνήθως πραγματοποιείται σε ισχυρά κέντρα δεδομένων, η συμπερασματολογία βασίζεται στη χρήση προεκπαιδευμένων μοντέλων για τη λήψη αποφάσεων σε πραγματικό χρόνο, καθιστώντας την κατάλληλη για ανάπτυξη σε περιβάλλοντα με περιορισμένους πόρους.

Η τοπική εκτέλεση τεχνητής νοημοσύνης προσφέρει πολλαπλά πλεονεκτήματα. Αρχικά, παρέχει εξαιρετικά χαμηλή καθυστέρηση, καθώς δεν απαιτείται απομακρυσμένη επικοινωνία για κάθε πρόβλεψη, ενισχύει την προστασία της ιδιωτικότητας των χρηστών διατηρώντας τα δεδομένα τοπικά, και μειώνει την κατανάλωση εύρους ζώνης και την εξάρτηση από κοστοβόρους υπολογιστικούς πόρους στο νέφος. Επιπλέον, διασφαλίζει υψηλή αξιοπιστία σε περιβάλλοντα με περιορισμένη ή διαλείπουσα συνδεσιμότητα, καθιστώντας την τεχνητή νοημοσύνη πιο επεκτάσιμη και βιώσιμη για εφαρμογές μεγάλης κλίμακας. Παρά τα πλεονεκτήματά της, η τοπική συμπερασματολογία συνοδεύεται από σημαντικές προκλήσεις. Οι περιορισμοί σε υπολογιστική ισχύ, μνήμη και ενεργειακή απόδοση που χαρακτηρίζουν τις συσκευές παρυφών καθιστούν δύσκολη την εκτέλεση μεγάλων μοντέλων, των οποίων οι απαιτήσεις σε αποθηκευτικό χώρο και επεξεργασία συχνά υπερβαίνουν τις δυνατότητες του υλικού. Η ετερογένεια των διαθέσιμων πλατφορμών—με διαφορές σε επεξεργαστές, μονάδες επιτάχυνσης και αισθητήρες—προσθέτει πολυπλοκότητα στην υλοποίηση και βελτιστοποίηση. Επιπλέον, η κατανάλωση ενέργειας αποτελεί κρίσιμο ζήτημα, καθώς η εκτέλεση αλγορίθμων τεχνητής νοημοσύνης μπορεί να μειώσει δραστικά τη διάρκεια ζωής της μπαταρίας, ενώ δυναμικοί περιβαλλοντικοί παράγοντες, όπως η θερμοκρασία, ενδέχεται να επηρεάσουν την απόδοση.

Η αντιμετώπιση αυτών των προκλήσεων απαιτεί καινοτόμες προσεγγίσεις στη συμπίεση μοντέλων, βελτιστοποιήσεις με επίγνωση του υποκείμενου υλικού, καθώς και έξυπνες στρατηγικές κατανομής πόρων που λαμβάνουν υπόψιν τους λειτουργικούς περιορισμούς της κάθε συσκευής. Μόνο μέσω τέτοιων τεχνικών θα καταστεί εφικτή η απρόσκοπτη και αποδοτική υλοποίηση της τεχνητής νοημοσύνης στις παρυφές του δικτύου.

Β.5 Αποδοτική Βαθιά Μάθηση

Η αποδοτική βαθιά μάθηση (efficient deep learning – EDL) αποσκοπεί στη διασφάλιση ότι τα βαθιά νευρωνικά δίκτυα μπορούν να λειτουργούν αποτελεσματικά σε περιβάλλοντα με περιορισμένους υπολογιστικούς πόρους, μεγιστοποιώντας την απόδοση σε επίπεδο υπολογισμού, μνήμης και ενέργειας. Παρότι η ακρίβεια παραμένει βασική επιδίωξη κατά τον σχεδιασμό μοντέλων, η υλοποίηση αποδοτικής τοπικής συμπερασματολογίας προϋποθέτει την ταυτόχρονη εξισορρόπηση πολλών κρίσιμων μετρικών απόδοσης. Παράγοντες όπως η καθυστέρηση (latency), η υπολογιστική πολυπλοκότητα (computational complexity) και ο ρυθμός διέλευσης (throughput) καθορίζουν την ταχύτητα και την αποτελεσματικότητα με την οποία ένα μοντέλο επεξεργάζεται δεδομένα, κάτι που είναι ιδιαίτερα σημαντικό σε εφαρμογές πραγματικού χρόνου. Επιπλέον, το μέγεθος του μοντέλου (model size) και το αποτύπωμα μνήμης (memory footprint) επηρεάζουν τη δυνατότητα ενσωμάτωσής του σε συσκευές με περιορισμένη χωρητικότητα αποθήκευσης και RAM, όπως οι συσκευές παρυφών. Η ενεργειακή κατανάλωση αποτελεί επίσης καθοριστικό παράγοντα, ειδικά σε φορητές ή

αυτόνομες συσκευές που εξαρτώνται από περιορισμένες ενεργειακές πηγές (*π.χ.*, μπαταρίες). Πέρα από τις αλγοριθμικές βελτιώσεις, η συμβατότητα με εξειδικευμένους επιταχυντές υλικού (accelerators), όπως GPUs και NPUs, καθίσταται ολοένα και πιο σημαντική για τη βελτιστοποίηση της εκτέλεσης. Η αξιοποίηση τέτοιων αρχιτεκτονικών επιταχύνει τη συμπερασματολογία και μειώνει την κατανάλωση ενέργειας, καθιστώντας την ανάπτυξη μοντέλων πιο αποδοτική.

Ωστόσο, η επίτευξη υψηλής επίδοσης και αποδοτικότητας απαιτεί την επίλυση αντισταθμίσεων μεταξύ των παραπάνω μετρικών. Για παράδειγμα, η μείωση του μεγέθους ενός μοντέλου μπορεί να επηρεάσει την ακρίβεια, ενώ η επιδίωξη χαμηλής καθυστέρησης ενδέχεται να αυξήσει την ενεργειακή κατανάλωση. Η διαχείριση αυτών των παραγόντων απαιτεί μια διεπιστημονική προσέγγιση, η οποία θα πρέπει να συνδυάζει αρχιτεκτονικό σχεδιασμό, μηχανική υλικού και βελτιστοποίηση λογισμικού, ώστε να διασφαλιστεί ότι τα μοντέλα τεχνητής νοημοσύνης παραμένουν υψηλής επίδοσης και ταυτόχρονα κατάλληλα για ανάπτυξη σε μια ποικιλία υλικών πλατφορμών.

Κβαντοποίηση

Η κβαντοποίηση (quantization) αποτελεί μια από τις πλέον διαδεδομένες τεχνικές συμπίεσης μοντέλων νευρωνικών δικτύων, στοχεύοντας στη μείωση του υπολογιστικού φόρτου και των απαιτήσεων σε μνήμη κατά την φάση της εκτέλεσης. Η βασική αρχή της μεθόδου συνίσταται στη μετατροπή των παραμέτρων και των υπολογισμών ενός δικτύου από αναπαραστάσεις κινητής υποδιαστολής υψηλής ακρίβειας (32-bit floating point) σε αναπαραστάσεις χαμηλότερης ακρίβειας, όπως 16-bit ή ακεραίους των 8 bits. Η μείωση της ακρίβειας οδηγεί σε μικρότερες απαιτήσεις αποθήκευσης, μειωμένη χρήση μνήμης και ταχύτερη εκτέλεση, καθιστώντας τα μοντέλα τεχνητής νοημοσύνης πιο κατάλληλα για ανάπτυξη σε συσκευές παρυφών με περιορισμένους πόρους.

Υπάρχουν διάφορα σχήματα κβαντοποίησης, όπως τα FP16, INT8, DR8, FX8 και FFX8, καθένα από τα οποία προσφέρει διαφορετικά επίπεδα αντισταθμίσεων μεταξύ ακρίβειας, υπολογιστικής αποδοτικότητας και συμβατότητας με το υποκείμενο υλικό. Η επιλογή της κατάλληλης στρατηγικής εξαρτάται από τις δυνατότητες της πλατφόρμας υλοποίησης (*π.χ.*, υποστήριξη από CPU, GPU ή NPU) και τις απαιτήσεις της εκάστοτε εφαρμογής, επιδιώκοντας την επίτευξη μιας ισορροπίας μεταξύ υπολογιστικής απόδοσης και απώλειας ακρίβειας.

Καθώς οι εφαρμογές τεχνητής νοημοσύνης σε συστήματα παρυφών συνεχίζουν να επεκτείνονται, η κβαντοποίηση και άλλες τεχνικές συμπίεσης, όπως το κλάδεμα παραμέτρων και η απόσταξη γνώσης, διαδραματίζουν καίριο ρόλο στην ενεργειακά αποδοτική και επεκτάσιμη ανάπτυξη μοντέλων σε πραγματικό χρόνο. Η αποτελεσματική ενσωμάτωση αυτών των τεχνικών αποτελεί αναγκαία προϋπόθεση για την υλοποίηση ευφυών συστημάτων που μπορούν να ανταποκριθούν στις απαιτήσεις σύγχρονων εφαρμογών σε υπολογιστικά περιβάλλοντα με περιορισμούς.

Γ Τοπική Συμπερασματολογία σε Κινητές Συσκευές

Η ραγδαία εξάπλωση των εφαρμογών βαθιάς μάθησης τα τελευταία έτη έχει οδηγήσει σε μια θεμελιώδη μετατόπιση προς την τοπική εκτέλεση αλγορίθμων τεχνητής νοημοσύνης, αντανακλώντας την αυξανόμενη ανάγκη για επεξεργασία σε πραγματικό χρόνο, προστασία του ιδιωτικού απορρήτου και μείωση της καθυστέρησης. Οι απαιτήσεις αυτές είναι ιδιαίτερα έντονες σε κρίσιμους τομείς, όπως η υγειονομική περίθαλψη, τα ευφυή οχήματα και οι εφαρμογές επαυξημένης πραγματικότητας.

Το παρόν κεφάλαιο διερευνά τις προκλήσεις που σχετίζονται με τη βέλτιστη εκτέλεση βαθιών νευρωνικών δικτύων σε κινητές συσκευές, δίνοντας έμφαση στην ετερογένεια του

υλικού (hardware heterogeneity), την ταυτόχρονη εκτέλεση πολλαπλών μοντέλων (multi-DNN execution) και την ανάγκη για δυναμική προσαρμογή των αποφάσεων κατά τον χρόνο εκτέλεσης (runtime adaptation). Στο πλαίσιο αυτό, παρουσιάζεται το **CARIn** (**C**onstraint-**A**ware and **R**esponsive **In**ference), ένα καινοτόμο πλαίσιο σχεδιασμένο για τη βελτιστοποιημένη ανάπτυξη μοντέλων τεχνητής νοημοσύνης σε κινητά περιβάλλοντα. Το CARIn υποστηρίζει εφαρμογές τόσο ενός όσο και πολλαπλών μοντέλων, λαμβάνοντας υπόψιν περιορισμούς πόρων και στόχους ποιότητας υπηρεσίας που ορίζονται δυναμικά από τον τελικό χρήστη.

Γ.1 Επισκόπηση Συστήματος

Σε αντίθεση με τις κλασικές προσεγγίσεις βελτιστοποίησης σε επίπεδο μοντέλου, το CARIn επικεντρώνεται αποκλειστικά στη διαχείριση σε επίπεδο συστήματος. Αντί να τροποποιεί τη δομή ή την εκπαίδευση των ίδιων των μοντέλων, αξιοποιεί μια αποθήκη προεκπαιδευμένων νευρωνικών δικτύων με ποικίλες αρχιτεκτονικές και εφαρμόζει τεχνικές κβαντοποίησης μετά την εκπαίδευση για την ενίσχυση της αποδοτικότητας. Ο βασικός στόχος του συστήματος είναι ο προσδιορισμός του καταλληλότερου ζεύγους μοντέλου-επεξεργαστή για κάθε περίπτωση, λαμβάνοντας υπόψιν τα χαρακτηριστικά της συσκευής, με σκοπό τη βέλτιστη εκτέλεση εφαρμογών βαθιάς μάθησης σε ετερογενή κινητά περιβάλλοντα. Το CARIn αποτελείται από τα ακόλουθα δύο βασικά συστατικά:

1. **Ένα εκφραστικό πλαίσιο βελτιστοποίησης πολλαπλών στόχων** (multi-objective optimization – MOO), το οποίο μοντελοποιεί τις απαιτήσεις των εφαρμογών βαθιάς μάθησης ως ένα πρόβλημα εξισορρόπησης περιορισμών και επιδόσεων.
2. **Τον αλγόριθμο ταξινόμησης και αναζήτησης με επίγνωση του χρόνου εκτέλεσης** (runtime-aware sorting and search – RASS), ο οποίος επιτρέπει την αποτελεσματική προσαρμογή του συστήματος σε δυναμικά μεταβαλλόμενες συνθήκες λειτουργίας. Ο RASS δημιουργεί αρχικά ένα σύνολο από εναλλακτικές διαμορφώσεις εκτέλεσης (δηλαδή διαφορετικούς συνδυασμούς μοντέλων και επεξεργαστών) και κατασκευάζει μια πολιτική εναλλαγής μεταξύ αυτών, ώστε να αντιμετωπίζονται αυτόματα αλλαγές στη διαθεσιμότητα των πόρων, διατηρώντας ταυτόχρονα την επιθυμητή ποιότητα υπηρεσίας.

Η συνολική ροή εργασιών του CARIn διαρθρώνεται σε δύο διακριτές φάσεις, η καθεμία με σαφώς καθορισμένο ρόλο στη διαδικασία βελτιστοποίησης και εκτέλεσης:

- **Εκτός σύνδεσης (offline) φάση:** Το σύστημα κατασκευάζει και επιλύει ένα πρόβλημα βελτιστοποίησης πολλαπλών στόχων για μια δεδομένη συσκευή, λαμβάνοντας υπόψιν: (α) τη διεργασία ή τις διεργασίες βαθιάς μάθησης που απαιτεί η εφαρμογή, (β) τους στόχους επιπέδου υπηρεσίας (service-level objectives – SLOs), και (γ) τα τεχνικά χαρακτηριστικά της συσκευής. Η έξοδος αυτής της φάσης περιλαμβάνει ένα βελτιστοποιημένο σύνολο διαμορφώσεων εκτέλεσης και μια αντίστοιχη πολιτική εναλλαγής μεταξύ τους.
- **Φάση χρόνου εκτέλεσης (online):** Ο Διαχειριστής Εκτέλεσης (Runtime Manager – RM) παρακολουθεί συνεχώς τη συμπεριφορά της εφαρμογής και τις συνθήκες του συστήματος. Όταν εντοπιστούν αλλαγές στους διαθέσιμους πόρους ή στην επίδοση, η διαμόρφωση εκτέλεσης προσαρμόζεται δυναμικά, βάσει της πολιτικής που έχει καθοριστεί στην προηγούμενη φάση.

Μέσω αυτού του σχεδιασμού, το CARIn προσφέρει μια ευέλικτη, προσαρμοστική και αποδοτική λύση για την εκτέλεση εφαρμογών βαθιάς μάθησης σε κινητές και ετερογενείς πλατφόρμες, λαμβάνοντας υπόψιν ρεαλιστικούς περιορισμούς και στόχους που ορίζονται από τον τελικό χρήστη.

Γ.2 Βελτιστοποίηση Πολλαπλών Στόχων

Η προσέγγιση της βελτιστοποίησης πολλαπλών στόχων υιοθετείται στο πλαίσιο του CARIn για δύο κύριους λόγους: (α) οι εφαρμογές βαθιάς μάθησης συνήθως συνοδεύονται από αντικρουόμενες απαιτήσεις επίδοσης, και (β) το μοντέλο αυτό επιτρέπει την παραγωγή ποικιλόμορφων και εναλλακτικών λύσεων, γεγονός που ενισχύει την προσαρμοστικότητα του συστήματος σε δυναμικά περιβάλλοντα. Οι αντικειμενικές συναρτήσεις και οι περιορισμοί της βελτιστοποίησης προκύπτουν άμεσα από τους στόχους επιπέδου υπηρεσίας που ορίζονται από την εφαρμογή.

Το σύστημα υποστηρίζει σενάρια τόσο ενός όσο και πολλαπλών μοντέλων, εξασφαλίζοντας ευελιξία και επεκτασιμότητα. Στην πρώτη περίπτωση, η βελτιστοποίηση βασίζεται σε ένα σύνολο από βασικές μετρικές, όπως το μέγεθος του μοντέλου, η υπολογιστική πολυπλοκότητα, η ακρίβεια, η καθυστέρηση, ο ρυθμός διέλευσης, η κατανάλωση ενέργειας και το αποτύπωμα μνήμης. Σκοπός είναι να διατηρείται υψηλή απόδοση συμπερασματολογίας χωρίς να θίγεται η λειτουργικότητα ή να παραβιάζονται οι περιορισμοί της πλατφόρμας. Σε περιβάλλοντα πολλαπλών μοντέλων, ενσωματώνονται πρόσθετες μετρικές επίδοσης, όπως ο κανονικοποιημένος χρόνος διεκπεραίωσης (normalized turnaround time), ο ρυθμός διέλευσης συστήματος (system throughput) και ο βαθμός δικαιοσύνης (fairness) μεταξύ των μοντέλων. Οι μετρικές αυτές επιτρέπουν την ισόρροπη κατανομή των πόρων και αποτρέπουν φαινόμενα συμφόρησης ή άνισης εξυπηρέτησης, τα οποία είναι κρίσιμα σε εφαρμογές με ταυτόχρονη εκτέλεση πολλαπλών διεργασιών τεχνητής νοημοσύνης.

Αν και οι κλασικοί αλγόριθμοι βελτιστοποίησης πολλαπλών στόχων μπορούν να παραγάγουν σύνολα αποδοτικών λύσεων (Pareto-optimal solutions), η μεταβλητότητα στους διαθέσιμους πόρους κατά τον χρόνο εκτέλεσης μεταβάλλει δυναμικά τον χώρο λύσεων και συχνά απαιτεί την επαναληπτική επίλυση του προβλήματος. Αυτή η απαίτηση καθιστά τη διαδικασία μη βιώσιμη για συστήματα πραγματικού χρόνου. Για την αντιμετώπιση αυτών των περιορισμών, το CARIn ενσωματώνει τον αλγόριθμο RASS, ο οποίος επιλύει το πρόβλημα βελτιστοποίησης μια μόνο φορά κατά τη φάση εκτός σύνδεσης, παράγοντας ένα πεπερασμένο σύνολο λύσεων που καλύπτει όλες τις ρεαλιστικές περιπτώσεις δυναμικής μεταβλητότητας. Η διαδικασία λειτουργεί σε δύο στάδια:

1. **Ταξινόμηση των πιθανών λύσεων** με βάση μια μετρική βελτιστότητας η οποία ποσοτικοποιεί την απόσταση κάθε λύσης από το ιδεατό σημείο (utopia point) του προβλήματος πολλαπλών στόχων.
2. **Αναζήτηση λύσεων** για τον εντοπισμό εκείνων που μπορούν να καλύψουν μελλοντικά σενάρια παραβίασης περιορισμών, καθώς και προσδιορισμός της πολιτικής εναλλαγής μεταξύ τους.

Κατά τον χρόνο εκτέλεσης, ο Διαχειριστής Εκτέλεσης αντιμετωπίζει ένα σύνολο από δυναμικές αλλαγές στους διαθέσιμους πόρους, οι οποίες επηρεάζουν τη λειτουργία του συστήματος. Τα πιο συνήθη ζητήματα αφορούν τους επεξεργαστές, όταν δηλαδή η απαιτούμενη υπολογιστική ισχύς υπερβαίνει τις δυνατότητες του ενεργού επεξεργαστή, οδηγώντας ενδεχομένως σε υπερθέρμανση ή υποβάθμιση απόδοσης. Αντίστοιχα, περιορισμοί μνήμης μπορεί να ανακύψουν όταν η χρήση RAM πλησιάζει το μέγιστο όριο, γεγονός που

επίσης απαιτεί προσαρμογή. Η στρατηγική του συστήματος προβλέπει εναλλαγές εντός της τρέχουσας διαμόρφωσης, οι οποίες περιλαμβάνουν:

- **Αλλαγή επεξεργαστή** (*π.χ.*, μεταφορά από CPU σε GPU)
- **Αλλαγή μοντέλου** σε ελαφρύτερη εκδοχή με χαμηλότερες απαιτήσεις
- **Συνδυασμό των παραπάνω**

Για παράδειγμα, όταν προκύπτουν περιορισμοί στον επεξεργαστή, το σύστημα δίνει προτεραιότητα στην αξιοποίηση ενός άλλου διαθέσιμου επιταχυντή. Εάν αυτό δεν είναι εφικτό, επιλέγεται το μοντέλο με την ελάχιστη υπολογιστική πολυπλοκότητα. Αντίστοιχα, σε συνθήκες περιορισμένης μνήμης, προτεραιότητα έχει η αντικατάσταση του μοντέλου με αυτό που παρουσιάζει το μικρότερο αποτύπωμα μνήμης.

Πρωταρχικός στόχος του RASS είναι να προσφέρει έναν αλγόριθμο χαμηλής υπολογιστικής πολυπλοκότητας, ικανό να υποστηρίξει ταχείες μεταβάσεις μεταξύ διαμορφώσεων εκτέλεσης. Αυτό επιτυγχάνεται μέσω της παραγωγής ενός μικρού αριθμού λύσεων (έως πέντε), που όμως είναι επαρκείς για την κάλυψη των περισσότερων ρεαλιστικών σεναρίων. Η αντίστοιχη πολιτική εναλλαγής διατηρείται επίσης μικρή και ευέλικτη, αποτελούμενη από έναν περιορισμένο αριθμό απλών κανόνων μετάβασης. Με αυτόν τον τρόπο, διασφαλίζεται η αποδοτικότητα του συστήματος χωρίς να απαιτείται εκ νέου βελτιστοποίηση κατά τον χρόνο εκτέλεσης.

Γ.3 Πειραματική Μεθοδολογία

Τα σύγχρονα ολοκληρωμένα κυκλώματα κινητών συσκευών (mobile SoCs) ενσωματώνουν πολλαπλούς ετερογενείς επεξεργαστές—όπως CPUs, GPUs και NPUs—οι οποίοι μπορούν να χρησιμοποιηθούν για την εκτέλεση νευρωνικών δικτύων. Ωστόσο, η πλήρης αξιοποίησή τους απαιτεί συχνά προσαρμογή των μοντέλων μέσω τεχνικών κβαντοποίησης, καθώς πολλοί από αυτούς τους επιταχυντές υποστηρίζουν μόνο αναπαραστάσεις μειωμένης ακρίβειας (*π.χ.*, INT8). Η εφαρμογή διαφορετικών σχημάτων κβαντοποίησης σε ένα σύνολο προεκπαιδευμένων μοντέλων επιτρέπει στο `CARIn` να διαμορφώσει έναν ευρύτερο χώρο πιθανών λύσεων, με διαφορετικά σημεία αντιστάθμισης μεταξύ ακρίβειας και υπολογιστικής αποδοτικότητας. Αυτό ενισχύει τη δυνατότητα του συστήματος να προσαρμόζεται σε ποικίλες απαιτήσεις απόδοσης και περιορισμούς υλικού.

Για την αξιολόγηση του `CARIn`, σχεδιάστηκαν και υλοποιήθηκαν τέσσερα διακριτά σενάρια εφαρμογής, έκαστο με συγκεκριμένους στόχους επιπέδου υπηρεσίας. Τα δύο πρώτα σενάρια αφορούν την εκτέλεση ενός μεμονωμένου μοντέλου, ενώ τα επόμενα δύο περιλαμβάνουν περιβάλλοντα με ταυτόχρονη εκτέλεση πολλαπλών μοντέλων, προσομοιώνοντας συνθήκες υψηλού φόρτου. Για κάθε σενάριο δημιουργήθηκε το αντίστοιχο υποσύνολο μοντέλων, από τα οποία το σύστημα μπορεί να επιλέξει κατά τη διαδικασία συμπερασματολογίας. Ενδεικτικά, το πρώτο σενάριο εξετάζει την κατηγοριοποίηση εικόνας σε πραγματικό χρόνο, με στόχο την υποστήριξη εφαρμογών που απαιτούν αναγνώριση 24 καρέ ανά δευτερόλεπτο (frames per second). Συνεπώς, τίθεται ως αυστηρός περιορισμός η μέγιστη καθυστέρηση των 41.67 ms ανά πρόβλεψη. Το αντίστοιχο πρόβλημα πολλαπλών στόχων στοχεύει στην ταυτόχρονη μεγιστοποίηση της ακρίβειας ταξινόμησης και του ρυθμού διέλευσης.

Η πειραματική αξιολόγηση πραγματοποιήθηκε σε τρεις διακριτές κινητές συσκευές, οι οποίες επιλέχθηκαν με κριτήριο την αντιπροσωπευτικότητα σε σχέση με τις κατηγορίες της τρέχουσας αγοράς (χαμηλού, μεσαίου και υψηλού κόστους). Κάθε συσκευή φέρει διαφορετικά χαρακτηριστικά υλικού και ενσωματώνει ετερογενείς επιταχυντές, επιτρέποντας τη δοκιμή του

συστήματος σε ρεαλιστικά, διαφοροποιημένα περιβάλλοντα. Μέσω αυτής της μεθοδολογίας, αξιολογείται η ικανότητα του CARIn να προσαρμόζεται δυναμικά, να επιλέγει αποδοτικές διαμορφώσεις εκτέλεσης και να διατηρεί υψηλή απόδοση κάτω από διαφορετικές απαιτήσεις εφαρμογής και ποικίλες υλικές πλατφόρμες.

Γ.4 Αποτελέσματα

Η απόδοση του CARIn αξιολογήθηκε πειραματικά έναντι δύο βασικών κατηγοριών μεθόδων αναφοράς:

1. **Απλοϊκές και εμπειρικές στρατηγικές επιλογής**, όπως αυτές που συναντώνται σε πραγματικές εμπορικές εφαρμογές.
2. **Το OODIn**, ένα πρόσφατο και σχετικό σύστημα βελτιστοποίησης με στόχο την προσαρμοστικότητα σε δυναμικές συνθήκες.

Οι εμπειρικές μέθοδοι αναφοράς περιλάμβαναν:

- **Την επιλογή του μοντέλου** με την υψηλότερη ακρίβεια ή το χαμηλότερο αποτύπωμα μνήμης, ανεξαρτήτως άλλων παραμέτρων.
- **Τη χρήση της διαμόρφωσης εκτέλεσης** που θεωρείται βελτιστοποιημένη για άλλη συσκευή, χωρίς εξειδίκευση στη στοχευμένη πλατφόρμα.
- **Την αντιμετώπιση του προβλήματος πολλαπλών στόχων** ως πολλαπλά ανεξάρτητα προβλήματα μεμονωμένου στόχου, αγνοώντας τις μεταξύ τους αλληλεπιδράσεις.

Εκτέλεση Ενός Μοντέλου
Στα δύο πρώτα σενάρια, όπου αξιολογείται η εκτέλεση ενός μεμονωμένου μοντέλου, το CARIn επιδεικνύει σαφώς ανώτερη απόδοση σε σύγκριση με όλα τα σημεία αναφοράς καθώς και με το OODIn. Συγκεκριμένα, στο 1ο σενάριο (κατηγοριοποίηση εικόνας σε πραγματικό χρόνο), το CARIn επιτυγχάνει βελτίωση 32.7% στον ρυθμό διέλευσης και μέση αύξηση 0.156 μονάδων στην ακρίβεια, ενώ στο 2ο σενάριο (κατηγοριοποίηση κειμένου με χαμηλό αποτύπωμα μνήμης), παρατηρείται επιτάχυνση κατά 19.9% στον χρόνο εκτέλεσης και μείωση κατά 2.8 MB στο αποτύπωμα μνήμης.

Εκτέλεση Πολλαπλών Μοντέλων
Κατά την αξιολόγηση στα σενάρια 3 και 4, όπου απαιτείται ταυτόχρονη εκτέλεση πολλαπλών μοντέλων, στο 3ο σενάριο, το CARIn διατηρεί την υπεροχή του έναντι των ανταγωνιστικών μεθόδων, εξασφαλίζοντας αποτελεσματική κατανομή πόρων και χαμηλή καθυστέρηση. Το 4ο σενάριο συνιστά μια πιο απαιτητική περίπτωση: οι περισσότερες μέθοδοι αναφοράς αδυνατούν να παράγουν οποιαδήποτε λύση που να ικανοποιεί τους περιορισμούς, ενώ όταν αυτό καταστεί εφικτό, η απόδοσή τους παραμένει συγκρίσιμη, γεγονός που αναδεικνύει την αναγκαιότητα ύπαρξης ποικιλίας μοντέλων στο σύστημα για αποτελεσματική αντιμετώπιση πολύπλοκων περιπτώσεων.

Προσαρμογή κατά τον Χρόνο Εκτέλεσης
Η προσαρμοστικότητα του Διαχειριστή Εκτέλεσης του CARIn αξιολογήθηκε στα σενάρια 1 και 3, προκειμένου να αποτιμηθεί η ικανότητα του συστήματος να ανταποκρίνεται σε δυναμικές αλλαγές στους διαθέσιμους πόρους και να αξιοποιεί τις εναλλακτικές διαμορφώσεις εκτέλεσης. Αν και σε ορισμένες περιπτώσεις το σύστημα ενδέχεται να λειτουργεί προσωρινά με μειωμένη απόδοση—λόγω, για παράδειγμα, μεταβατικών φάσεων εναλλαγής μοντέλου ή επιταχυντή—τέτοια φαινόμενα κρίνονται προσωρινά και διαχειρίσιμα, καθώς ο μηχανισμός προσαρμογής φροντίζει για την ταχεία επαναφορά σε αποδοτική λειτουργία. Το σημαντικότερο

είναι ότι σε όλες τις περιπτώσεις τηρούνται αυστηρά οι περιορισμοί που τίθενται από την εφαρμογή, διασφαλίζοντας τη διατήρηση της ποιότητας εμπειρίας του χρήστη.

Σύγκριση με το OODIn

Το σύστημα OODIn παρουσιάζει δύο βασικούς περιορισμούς:

1. **Αδυναμία πρόβλεψης επικείμενων αλλαγών στη διαθεσιμότητα πόρων:** Όταν συμβαίνουν αλλαγές, το σύστημα απαιτεί την εκ νέου επίλυση του προβλήματος βελτιστοποίησης, διαδικασία που συνοδεύεται από σημαντικό χρονικό κόστος, ιδιαίτερα όταν ο αριθμός των στόχων και το εύρος του χώρου λύσεων είναι μεγάλα.
2. **Απαίτηση για τοπική πρόσβαση σε όλα τα πιθανά μοντέλα:** Αυτό συνεπάγεται την ανάγκη αποθήκευσης ολόκληρου του συνόλου μοντέλων στη συσκευή, γεγονός που αυξάνει τις απαιτήσεις σε αποθηκευτικό χώρο και καθιστά δύσκολη την κλιμάκωση.

Συνολικά, τα αποτελέσματα καταδεικνύουν ότι το CARIn προσφέρει σημαντικά πλεονεκτήματα σε όρους επίδοσης, ευελιξίας και αποδοτικότητας προσαρμογής, καθιστώντας το κατάλληλο για ανάπτυξη σε ετερογενή και απαιτητικά κινητά περιβάλλοντα με περιορισμένους πόρους.

Γ.5 Συμπέρασμα

Η αποδοτική και ευέλικτη εκτέλεση νευρωνικών δικτύων σε κινητές συσκευές αποτελεί κρίσιμο ζητούμενο για την ικανοποίηση των διαρκώς αυξανόμενων απαιτήσεων των εφαρμογών τεχνητής νοημοσύνης. Το πλαίσιο CARIn, το οποίο παρουσιάστηκε στο παρόν κεφάλαιο, επιδιώκει να καλύψει αυτό το κενό προσφέροντας μια ολοκληρωμένη, δυναμικά προσαρμοζόμενη και συστηματικά θεμελιωμένη προσέγγιση.

Παρότι εξακολουθούν να υφίστανται σημαντικές προκλήσεις, όπως η ετερογένεια του υλικού, η ανάγκη για προσαρμογή κατά τον χρόνο εκτέλεσης, και η διαχείριση της ταυτόχρονης εκτέλεσης πολλαπλών μοντέλων, το CARIn εισάγει ένα νέο παράδειγμα σχεδίασης που ανταποκρίνεται αποτελεσματικά σε αυτές τις δυσκολίες. Μέσω της ενσωμάτωσης ενός εκφραστικού πλαισίου βελτιστοποίησης πολλαπλών στόχων, και της υλοποίησης του αλγορίθμου RASS, το σύστημα επιτυγχάνει σημαντική βελτίωση στην προσαρμοστικότητα και στην αποδοτικότητα, διατηρώντας παράλληλα την ικανοποίηση των στόχων απόδοσης που ορίζει ο τελικός χρήστης. Ο RASS διακρίνεται για την ικανότητά του να προβλέπει επικείμενες μεταβολές στους διαθέσιμους πόρους και να προετοιμάζει κατάλληλα σύνολα διαμορφώσεων εκτέλεσης, τα οποία μπορούν να ενεργοποιηθούν γρήγορα με ελάχιστο υπολογιστικό κόστος, ως απάντηση σε απρόβλεπτες συνθήκες. Με αυτόν τον τρόπο, το CARIn προσφέρει μια ρεαλιστική και αποτελεσματική λύση για την ανάπτυξη έξυπνων, αποδοτικών και ανθεκτικών εφαρμογών τεχνητής νοημοσύνης σε κινητές συσκευές.

Δ Εκτέλεση Μοντέλων Μετασχηματιστών σε Κινητές Συσκευές

Τα συνελικτικά νευρωνικά δίκτυα αποτέλεσαν επί σειρά ετών την κυρίαρχη αρχιτεκτονική στην όραση υπολογιστών, σημειώνοντας κορυφαίες επιδόσεις σε καθιερωμένες εργασίες, όπως η ταξινόμηση εικόνων (image classification), η ανίχνευση αντικειμένων (object detection) και η κατάτμηση εικόνων (image segmentation). Η επιτυχία τους οφείλεται, σε μεγάλο βαθμό, στην ικανότητά τους να αποτυπώνουν ιεραρχικές χωρικές συσχετίσεις στα οπτικά δεδομένα. Η ευρεία χρήση τους προκάλεσε έντονο ερευνητικό ενδιαφέρον για την αποδοτική υλοποίησή τους σε περιβάλλοντα με περιορισμένους πόρους, οδηγώντας σε πληθώρα βελτιστοποιήσεων και τεχνικών συμπίεσης που καθιστούν δυνατή την εκτέλεση τους σε συσκευές παρυφών.

Η εισαγωγή της αρχιτεκτονικής των μετασχηματιστών το 2017 σηματοδότησε σημείο καμπής στην επεξεργασία φυσικής γλώσσας, καταργώντας την εξάρτηση από τις επαναληπτικές δομές και προσφέροντας παράλληλη επεξεργασία και καλύτερη αποτύπωση μακροχρόνιων εξαρτήσεων. Η επίδραση των μετασχηματιστών δεν περιορίστηκε στα γλωσσικά δεδομένα, αλλά επεκτάθηκε δυναμικά και σε άλλους τομείς, όπως η όραση υπολογιστών, η αναγνώριση ομιλίας, και επιστημονικά πεδία αιχμής, συμπεριλαμβανομένης της ανακάλυψης φαρμάκων. Η ταχεία πρόοδος και διάδοση των μεγάλων γλωσσικών μοντέλων (large language models – LLMs), όπως τα Gemini, GPT, Llama, και Claude, έχει καθιερώσει περαιτέρω τους μετασχηματιστές ως την κυρίαρχη υπολογιστική αρχιτεκτονική στο ευρύτερο φάσμα της τεχνητής νοημοσύνης.

Παρά τα πλεονεκτήματα, οι μετασχηματιστές παρουσιάζουν ιδιαίτερα αυξημένες υπολογιστικές απαιτήσεις, τόσο κατά τη φάση της εκπαίδευσης όσο και κατά τη συμπερασματολογία. Παρόλο που τα CNNs έχουν ήδη περάσει από πολυετή διαδικασία βελτιστοποίησης, οι μετασχηματιστές εξακολουθούν σε μεγάλο βαθμό να εξαρτώνται από υποδομές υψηλής απόδοσης που παρέχει το υπολογιστικό νέφος, γεγονός που περιορίζει την αξιοποίησή τους σε περιβάλλοντα παρυφών. Κατά συνέπεια, ανακύπτει ένα κρίσιμο ερευνητικό ερώτημα: Κατά πόσο οι τεχνικές αποδοτικής εκτέλεσης που αποδείχθηκαν επιτυχείς για τα CNNs μπορούν να προσαρμοστούν στους μετασχηματιστές, ώστε να καταστούν βιώσιμοι για τοπική συμπερασματολογία; Η ανάγκη για απαντήσεις στο ερώτημα αυτό καθίσταται επιτακτική, καθώς οι εφαρμογές τεχνητής νοημοσύνης παρυφών επεκτείνονται ραγδαία σε πεδία όπως οι φωνητικοί βοηθοί, η πολυτροπική κατανόηση και η επαυξημένη πραγματικότητα.

Το παρόν κεφάλαιο εξετάζει συστηματικά την τρέχουσα κατάσταση γύρω από την εκτέλεση μοντέλων μετασχηματιστών σε κινητές συσκευές. Μέσω ενδελεχούς αξιολόγησης και πειραματικής ανάλυσης, παρουσιάζονται βελτιστοποιήσεις και στρατηγικές σχεδίασης που στοχεύουν στην ενίσχυση της αποδοτικότητας των μετασχηματιστών σε περιβάλλοντα με περιορισμένους πόρους, συμβάλλοντας στη μετάβαση από την εξάρτηση από το νέφος προς την αποκεντρωμένη και ευφυή επεξεργασία στις παρυφές.

Δ.1 Πειραματική Μεθοδολογία

Για την αξιολόγηση της εκτέλεσης μοντέλων μετασχηματιστών σε κινητές συσκευές, κατασκευάστηκε αρχικά μια βιβλιοθήκη από ποικίλα μοντέλα μετασχηματιστών, επιλεγμένα έτσι ώστε να καλύπτουν ένα ευρύ φάσμα απαιτήσεων σε υπολογιστικούς πόρους και λειτουργικές δυνατότητες. Τα εν λόγω μοντέλα είτε εκπαιδεύτηκαν εξαρχής, είτε προσαρμόστηκαν μέσω μεθόδων μεταφοράς μάθησης (transfer learning) για την εκτέλεση δύο ενδεικτικών εργασιών: ανάλυση συναισθήματος και παραγωγή κειμένου, οι οποίες αντιπροσωπεύουν κλασικές περιπτώσεις χρήσης στην επεξεργασία φυσικής γλώσσας.

Κατόπιν, σε κάθε μοντέλο εφαρμόστηκαν τρία διαφορετικά σχήματα κβαντοποίησης, με στόχο τη μελέτη της επίδρασής τους τόσο στην ακρίβεια όσο και στις μετρικές εκτέλεσης (*π.χ.*, καθυστέρηση, κατανάλωση ενέργειας, αποτύπωμα μνήμης). Η εφαρμογή της κβαντοποίησης είναι αναγκαία, ιδίως όταν η εκτέλεση στοχεύει να αξιοποιήσει εξειδικευμένες μονάδες υλικού, όπως DSPs και NPUs, οι οποίες συχνά λειτουργούν βέλτιστα με αριθμητική χαμηλής ακρίβειας (*π.χ.*, INT8). Το σύνολο των μοντέλων, περιλαμβανομένων των πλήρους ακρίβειας και κβαντισμένων εκδοχών τους, εκτελέστηκε σε δύο διαφορετικές κινητές συσκευές, επιλεγμένες ώστε να εκπροσωπούν ρεαλιστικές υλοποιήσεις παρυφών με διαφορετική υλική διαρρύθμιση και επιταχυντές.

Πριν από την εκτέλεση των μοντέλων και την καταγραφή των σχετικών μετρήσεων, πραγματοποιήθηκε ανάλυση της συμβατότητας μεταξύ των σχημάτων κβαντοποίησης και των

διαθέσιμων επεξεργαστών κάθε συσκευής. Τα αποτελέσματα της ανάλυσης κατέδειξαν ότι, από τους εξεταζόμενους επιταχυντές, μόνο η GPU παρουσίασε πλήρη συμβατότητα με τα μοντέλα μετασχηματιστών, προσφέροντας τη δυνατότητα για αποδοτική εκτέλεση με επιτάχυνση. Αντιθέτως, μονάδες όπως οι DSP και NPU εμφάνισαν περιορισμένη ή και ανεπαρκή υποστήριξη, με αποτέλεσμα σε πολλές περιπτώσεις η χρήση τους να οδηγεί σε υποβάθμιση της απόδοσης λόγω μείωσης της ακρίβειας ή μη υποστηριζόμενων λειτουργιών.

Η εν λόγω μεθοδολογία επιτρέπει την αποτίμηση της πραγματικής σκοπιμότητας και αποδοτικότητας της εκτέλεσης μοντέλων μετασχηματιστών στις παρυφές και θέτει τη βάση για την αναγνώριση κατάλληλων συνδυασμών μοντέλων, κβαντοποίησης και επιταχυντών, με σκοπό την ελαχιστοποίηση του υπολογιστικού κόστους χωρίς σημαντική απώλεια της επίδοσης.

Δ.2 Αποτελέσματα

Η πειραματική αξιολόγηση επικεντρώθηκε σε μια πολυπαραγοντική ανάλυση της εκτέλεσης μοντέλων μετασχηματιστών σε κινητές συσκευές. Συγκεκριμένα, εξετάστηκαν η τοπική ακρίβεια, η επίδοση των αρχικών αλλά και των κβαντισμένων μοντέλων στη CPU, καθώς και η συμπεριφορά των επιταχυντών υλικού, προκειμένου να αποτυπωθεί σφαιρικά η τρέχουσα κατάσταση και να εντοπιστούν οι βασικοί περιορισμοί.

Τοπική Ακρίβεια

Η αρχική ανάλυση εστιάζει στη συντήρηση της ακρίβειας κατά την εκτέλεση των μοντέλων τοπικά στις συσκευές, σε σύγκριση με το υπολογιστικό νέφος. Τα αποτελέσματα υποδεικνύουν ότι οι περισσότερες αρχιτεκτονικές χάνουν μέρος της ακρίβειάς τους κατά την εκτέλεση στη GPU, κυρίως λόγω ασυμβατοτήτων με πράξεις αριθμητικής ακρίβειας ή μη υποστηριζόμενων λειτουργιών κανονικοποίησης. Εξαίρεση αποτελεί το MobileBERT, ένα μοντέλο σχεδιασμένο ειδικά για κινητές συσκευές, το οποίο δεν χρησιμοποιεί τη συμβατική συνάρτηση ενεργοποίησης GELU, ούτε κανονικοποίηση επιπέδου (layer normalization). Η ιδιαιτερότητα αυτή φαίνεται να συμβάλλει στη σταθερότητα της ακρίβειας, υποδεικνύοντας πιθανά μονοπάτια αρχιτεκτονικών βελτιστοποιήσεων για βιώσιμη συμπερασματολογία σε κινητές συσκευές.

Επίδοση στη CPU

Η εκτέλεση των μοντέλων στην CPU αποκάλυψε σημαντικές αποκλίσεις ανάμεσα στις θεωρητικά βέλτιστες και στις πραγματικά αποδοτικές διαμορφώσεις:

- **Η χρήση της βιβλιοθήκης XNNPACK ως προκαθορισμένη επιλογή** δεν διασφαλίζει απαραίτητα τη χαμηλότερη καθυστέρηση, καθώς άλλες διαμορφώσεις με διαφορετικό αριθμό νημάτων μπορούν να προσφέρουν καλύτερη χρονική απόδοση.
- **Η ετερογένεια μεταξύ των συσκευών** καθιστά επιτακτική την ανάγκη για προσαρμοστική επιλογή διαμόρφωσης ανά συσκευή, καθώς η βέλτιστη διαμόρφωση εκτέλεσης δεν είναι καθολική.

Κβαντοποίηση και CPU

Τα αποτελέσματα της εκτέλεσης των κβαντισμένων μοντέλων στη CPU δείχνουν τα εξής:

- **Τα FP16 μοντέλα** επιτυγχάνουν παρόμοια καθυστέρηση με τα πλήρους ακρίβειας μοντέλα, όμως αυξάνουν τη χρήση μνήμης κατά περίπου 70%, γεγονός που μπορεί να επηρεάσει τη βιωσιμότητα σε περιβάλλοντα περιορισμένων πόρων.
- **Αντιθέτως, τα ακέραια μοντέλα (DR8 και FFX8)** παρουσιάζουν ταυτόχρονη επιτάχυνση της εκτέλεσης και μείωση του αποτυπώματος μνήμης στις περισσότερες

περιπτώσεις. Ωστόσο, συνοδεύονται από αξιοσημείωτη μείωση της ακρίβειας σε συγκεκριμένες αρχιτεκτονικές, γεγονός που υποδεικνύει την ανάγκη για προσεκτική επιλογή και δοκιμή.

Επιταχυντές
Η ανάλυση της εκτέλεσης στους επιταχυντές ανέδειξε σημαντικά ευρήματα:

- **Ο DSP αποδείχθηκε ακατάλληλος για την εκτέλεση μετασχηματιστών**, εξαιτίας της πολύ χαμηλής συμβατότητάς του με τις απαιτούμενες λειτουργίες, γεγονός που όχι μόνο δεν προσφέρει επιτάχυνση, αλλά αντίθετα οδηγεί σε επιβράδυνση.
- **Η GPU, αντιθέτως, παρέχει ουσιαστική επιτάχυνση στα μεγαλύτερα μοντέλα**, ειδικά σε όσα περιλαμβάνουν περισσότερα από 20 εκατομμύρια παραμέτρους. Ωστόσο, αυτή η επιτάχυνση συνοδεύεται από αύξηση του αποτυπώματος μνήμης, γεγονός που ενδέχεται να επηρεάσει αρνητικά άλλες συνιστώσες του συστήματος ή να περιορίσει την ταυτόχρονη εκτέλεση πολλαπλών μοντέλων.

Δ.3 Προτεινόμενες Βελτιστοποιήσεις

Τα ευρήματα της πειραματικής αξιολόγησης αναδεικνύουν πολλαπλές προκλήσεις για την εκτέλεση μοντέλων μετασχηματιστών σε κινητές συσκευές και υπογραμμίζουν την ανάγκη για στοχευμένες βελτιστοποιήσεις. Οι προτεινόμενες παρεμβάσεις διακρίνονται σε δύο επίπεδα: συστήματος και μοντέλου, αντικατοπτρίζοντας την ανάγκη τόσο για δυναμική διαχείριση της εκτέλεσης, όσο και για αρχιτεκτονική προσαρμογή των ίδιων των δικτύων.

Επίπεδο Συστήματος
Η παρατηρούμενη έλλειψη καθολικά αποδοτικής διαμόρφωσης εκτέλεσης καθιστά σαφή την ανάγκη για προσαρμοστικά συστήματα, ικανά να παρακολουθούν συνεχώς τη διαθεσιμότητα πόρων, να αναγνωρίζουν τα χαρακτηριστικά του χρησιμοποιούμενου υλικού και να επιλέγουν δυναμικά τις καταλληλότερες διαμορφώσεις. Η ετερογένεια των συσκευών, σε συνδυασμό με τη χρονικά μεταβαλλόμενη φύση των συνθηκών εκτέλεσης (*π.χ.*, θερμοκρασία, ταυτόχρονες διεργασίες), απαιτεί την ανάπτυξη μηχανισμών λήψης αποφάσεων σε πραγματικό χρόνο, με επίγνωση τόσο των εφαρμογών, όσο και των συστημικών περιορισμών. Μελλοντικές ερευνητικές κατευθύνσεις πρέπει να εστιάσουν στη συλλογή και ανάλυση μετρικών χρόνου εκτέλεσης, με στόχο τη βελτιστοποίηση της χρήσης του υλικού χωρίς συμβιβασμούς στην ακρίβεια ή την εμπειρία του χρήστη.

Επίπεδο Μοντέλου
Η ανάλυση ανέδειξε ότι οι μετασχηματιστές εμφανίζουν περιορισμένη συμβατότητα με τις περισσότερες μονάδες επιτάχυνσης (όπως DSPs και NPUs), ενώ ακόμη και οι GPUs, παρά την επιτάχυνση που προσφέρουν στα μεγαλύτερα μοντέλα, επιφέρουν σημαντική αύξηση στη χρήση μνήμης και επιδείνωση της ακρίβειας. Με γνώμονα τη σχεδίαση του MobileBERT, το οποίο διακρίνεται για τη συμβατότητά του με κινητές συσκευές, προτείνονται δύο αρχιτεκτονικές τροποποιήσεις που ενδέχεται να βελτιώσουν τη συμβατότητα και την αποδοτικότητα:

- **(R1):** Αντικατάσταση της συνάρτησης ενεργοποίησης GELU με τη ReLU, η οποία χαρακτηρίζεται από χαμηλότερο υπολογιστικό κόστος και υψηλότερη συμβατότητα με τις μονάδες επιτάχυνσης υλικού.
- **(R2):** Αντικατάσταση της κανονικοποίησης επιπέδου (layer normalization) με κανονικοποίηση δέσμης (batch normalization).

Τα πειραματικά αποτελέσματα δείχνουν ότι η αντικατάσταση R1 βελτιώνει τη χρονική απόδοση στη CPU σε όλες τις κβαντισμένες παραλλαγές, χωρίς όμως να επιλύει τα προβλήματα ακρίβειας κατά την εκτέλεση στη GPU, γεγονός που υποδηλώνει ότι τα θέματα αριθμητικής σταθερότητας παραμένουν. Η αντικατάσταση R2, αντιθέτως, αποδείχθηκε ιδιαίτερα επωφελής, καθώς:

- **Διατηρεί την ακρίβεια στη GPU.**
- **Προσφέρει επιτάχυνση** έως 2.5× για τα πλήρως ακέραια μοντέλα και έως 1.2× για τα υπόλοιπα.

Παρά τις βελτιώσεις στη CPU και τη GPU, οι συγκεκριμένες παρεμβάσεις δεν επιλύουν τα θέματα συμβατότητας με τους DSPs και τις NPUs, γεγονός που υποδηλώνει ότι οι μονάδες αυτές δεν έχουν σχεδιαστεί για αρχιτεκτονικές μετασχηματιστών και ότι απλές αρχιτεκτονικές τροποποιήσεις δεν επαρκούν για να αξιοποιηθούν πλήρως οι δυνατότητές τους. Τα συνολικά ευρήματα καθιστούν σαφές ότι η βιώσιμη ανάπτυξη μοντέλων μετασχηματιστών σε περιβάλλοντα με περιορισμένους πόρους προϋποθέτει αρχιτεκτονικές σχεδιάσεις με επίγνωση του υλικού, οι οποίες συνδυάζουν επιδόσεις και συμβατότητα, ώστε να διασφαλίζεται η αποτελεσματική επέκταση των εφαρμογών τεχνητής νοημοσύνης σε κινητές συσκευές.

Δ.4 Συμπέρασμα

Το παρόν κεφάλαιο ανέδειξε ότι οι στρατηγικές τοπικής συμπερασματολογίας που έχουν αποδειχθεί αποτελεσματικές για τα συνελικτικά νευρωνικά δίκτυα δεν είναι άμεσα μεταβιβάσιμες στους μετασχηματιστές, εξαιτίας θεμελιωδών διαφορών στην αρχιτεκτονική τους, στη μορφή των υπολογιστικών τους εξαρτήσεων και στη συμβατότητά τους με τις υφιστάμενες υποδομές υλικού κινητών συσκευών.

Μέσα από μια συστηματική και εμπειρικά τεκμηριωμένη μελέτη, πραγματοποιήθηκε συγκριτική αξιολόγηση πολλαπλών μοντέλων μετασχηματιστών σε διαφορετικές κινητές πλατφόρμες, με στόχο τη διερεύνηση της συμπεριφοράς τους σε όρους ακρίβειας, χρονικής καθυστέρησης, χρήσης μνήμης, καθώς και της αποδοτικότητας διαφόρων τεχνικών κβαντοποίησης. Τα αποτελέσματα κατέδειξαν σημαντικές αποκλίσεις στην απόδοση ανάμεσα στις αρχιτεκτονικές, τις τεχνικές βελτιστοποίησης και τους επιταχυντές, αναδεικνύοντας τις μοναδικές προκλήσεις που εγείρει η ανάπτυξη μετασχηματιστών σε περιβάλλοντα με αυστηρούς περιορισμούς πόρων.

Οι παρατηρήσεις αυτές υπογραμμίζουν την ανάγκη για νέες στρατηγικές βελτιστοποίησης με επίγνωση του υλικού, τόσο σε επίπεδο μοντέλου όσο και σε επίπεδο συστήματος. Επιπλέον, οι πειραματικές βελτιώσεις που προτάθηκαν, όπως η αρχιτεκτονική τροποποίηση των μηχανισμών ενεργοποίησης και κανονικοποίησης, παρέχουν ένα πλαίσιο για την πρακτική και προσαρμόσιμη ανάπτυξη μοντέλων μετασχηματιστών σε κινητές συσκευές, συμβάλλοντας στην περαιτέρω διάδοση της τεχνητής νοημοσύνης στις παρυφές του δικτύου.

Ε Έγκαιρη Ανίχνευση Εισβολών σε Δίκτυα IoT

Το διαδίκτυο των πραγμάτων (Internet of Things – IoT) συνιστά ένα ταχέως αναπτυσσόμενο υπολογιστικό μοντέλο, το οποίο βασίζεται στη δικτυακή διασύνδεση ενσωματωμένων συσκευών με δυνατότητες ανίχνευσης, τοπικής επεξεργασίας και ασύρματης επικοινωνίας. Αυτές οι δυνατότητες επιτρέπουν την ανταλλαγή και επεξεργασία δεδομένων σε πραγματικό χρόνο, διευκολύνοντας προηγμένες εφαρμογές σε τομείς όπως η βιομηχανική παραγωγή, τα έξυπνα σπίτια, η υγειονομική περίθαλψη και τα συστήματα μεταφορών.

Ωστόσο, η ευρεία υιοθέτηση του IoT συνοδεύεται από σημαντικές προκλήσεις ασφάλειας, καθώς οι εγγενείς περιορισμοί σε υπολογιστική ισχύ, διαθέσιμη μνήμη και ενεργειακή αυτονομία καθιστούν τα συστήματα αυτά ιδιαίτερα ευάλωτα σε ποικίλες μορφές επιθέσεων. Για την αποτελεσματική προστασία τέτοιων περιβαλλόντων, τα συστήματα ανίχνευσης εισβολών (intrusion detection systems – IDS) εξελίσσονται σταδιακά προς την υιοθέτηση τεχνικών βαθιάς μάθησης, οι οποίες έχουν αποδείξει υπεροχή έναντι των παραδοσιακών μεθόδων, ιδίως στην ανίχνευση μη εκ των προτέρων γνωστών ή δυναμικά εξελισσόμενων επιθέσεων. Ωστόσο, παρά τις πρόσφατες εξελίξεις, η έγκαιρη και αποδοτική ανίχνευση εξακολουθεί να αποτελεί ανοικτό ερευνητικό ζήτημα, ιδιαίτερα όταν οι εισερχόμενες ροές είναι μερικές, ατελείς ή χρονικά περιορισμένες. Παράλληλα, οι περιορισμοί των διαθέσιμων συνόλων δεδομένων, σε συνδυασμό με τις έντονες ανισορροπίες μεταξύ κακόβουλης και καλοήθους κίνησης, δυσχεραίνουν σημαντικά τη διαδικασία εκπαίδευσης και γενίκευσης των συστημάτων IDS.

Σε αυτό το πλαίσιο, παρουσιάζεται το προτεινόμενο σύστημα έγκαιρης ανίχνευσης εισβολών A-THENA, το οποίο βασίζεται στην αρχιτεκτονική του μετασχηματιστή, εμπλουτισμένη με χρονικά ευαίσθητες κωδικοποιήσεις θέσης (time-aware positional encodings). Αυτές οι κωδικοποιήσεις επιτρέπουν τη μοντελοποίηση τόσο της σειριακής δομής όσο και της χρονικής δυναμικής των πακέτων εντός μιας ροής κίνησης. Επιπλέον, το σύστημα εισάγει έναν καινοτόμο αγωγό επαύξησης δεδομένων (data augmentation pipeline) για τη βελτίωση της γενίκευσης, καθώς και μια εξειδικευμένη συνάρτηση απώλειας για έγκαιρη ανίχνευση (early detection loss - EDeL), η οποία ενισχύει την ικανότητα του μοντέλου να ανιχνεύει επιθέσεις με τον ελάχιστο δυνατό χρόνο ανταπόκρισης. Το A-THENA βελτιστοποιείται για ταχύτητα, ακρίβεια και αποδοτικότητα, καθιστώντας το κατάλληλο για ανάπτυξη σε περιβάλλοντα με περιορισμένους υπολογιστικούς πόρους, όπως αυτά που χαρακτηρίζουν τα σύγχρονα IoT δίκτυα.

Ε.1 Προτεινόμενο Σύστημα

Η παρούσα ενότητα παρουσιάζει μια μέθοδο έγκαιρης ανίχνευσης εισβολών βασισμένη στην αρχιτεκτονική του μετασχηματιστή, σχεδιασμένη ειδικά για την αναγνώριση απειλών μέσω της ανάλυσης περιορισμένου αριθμού πακέτων από μια πλήρη ροή δικτυακής κίνησης. Μια ροή (flow) ορίζεται συνήθως με βάση την πεντάδα:

(διεύθυνση IP πηγής, διεύθυνση IP προορισμού, θύρα στρώματος μεταφοράς πηγής, θύρα στρώματος μεταφοράς προορισμού, πρωτόκολλο)

η οποία χρησιμοποιείται για την ομαδοποίηση των πακέτων προς επεξεργασία. Η προτεινόμενη προσέγγιση στοχεύει στην πρόωρη και αποτελεσματική ταξινόμηση κακόβουλων ροών, επιτρέποντας την έγκαιρη αντίδραση σε ενδεχόμενες επιθέσεις, ενώ παράλληλα διατηρεί χαμηλές υπολογιστικές απαιτήσεις, καθιστώντας την κατάλληλη για υλοποίηση σε συσκευές περιορισμένων πόρων. Η μεθοδολογία δίνει έμφαση σε τέσσερις βασικούς στόχους:

1. **Εξάλειψη της ανάγκης για χειροκίνητη εξαγωγή χαρακτηριστικών**, αξιοποιώντας ως είσοδο τα ακατέργαστα πακέτα.
2. **Διασφάλιση υπολογιστικής αποδοτικότητας και χαμηλής καθυστέρησης**, στοιχείων απαραίτητων για ανάπτυξη σε πραγματικό χρόνο.
3. **Έγκαιρη ανίχνευση απειλών με ελάχιστη πληροφορία εισόδου**, επιτρέποντας έγκυρες προβλέψεις με βάση μόλις λίγα πακέτα.
4. **Υποστήριξη ροών μεταβλητού μήκους**, χωρίς την απαίτηση σταθερών διαστάσεων εισόδου.

Προετοιμασία Δεδομένων

Σε αυτό το στάδιο, η ακατέργαστη δικτυακή κίνηση συλλέγεται, οργανώνεται σε ροές, φιλτράρεται και μετασχηματίζεται σε μορφή κατάλληλη για εισαγωγή στο μοντέλο. Προς αποφυγή των περιορισμών που συνοδεύουν τις συμβατικές μεθόδους χειροκίνητης εξαγωγής χαρακτηριστικών, το προτεινόμενο σύστημα υιοθετεί μια ενδογενή προσέγγιση, στην οποία ως είσοδος στο νευρωνικό δίκτυο χρησιμοποιούνται οι απευθείας τιμές των bytes των πακέτων. Η στρατηγική αυτή μειώνει σημαντικά το υπολογιστικό κόστος της προεπεξεργασίας, εξαλείφει την ανάγκη σχεδιασμού ειδικών χαρακτηριστικών ανά εφαρμογή και επιτρέπει τη διατήρηση της πληροφορίας σε επίπεδο πακέτου, η οποία είναι ουσιώδης για τη σύλληψη λεπτομερών μοτίβων κακόβουλης συμπεριφοράς.

Η διαδικασία προετοιμασίας των δεδομένων οργανώνεται σε τρία επιμέρους στάδια:

- **Αναγνώριση Ροών (Flow Identification):** Για την αποτελεσματική διαχείριση της ετερογένειας στις διάφορες επιθέσεις, το σύστημα εφαρμόζει μια ευέλικτη στρατηγική ομαδοποίησης πακέτων σε ροές. Αρχικά, η ομαδοποίηση βασίζεται στην παραδοσιακή πεντάδα, ωστόσο, όταν ο αριθμός των ενεργών ροών υπερβαίνει προκαθορισμένα όρια, γεγονός που αποτελεί συχνά ένδειξη επιθετικής δραστηριότητας, επιτρέπεται η χαλάρωση των κριτηρίων ομαδοποίησης, όπως η παράβλεψη συγκεκριμένων διευθύνσεων IP ή θυρών, με στόχο τη συγκέντρωση των πακέτων σε λειτουργικά συνεκτικές ροές που διευκολύνουν την ανάλυση.
- **Φιλτράρισμα Πακέτων (Packet Filtering):** Εντός κάθε ροής, διατηρούνται μόνο τα πακέτα που αντιστοιχούν σε υποψήφια κακόβουλη δραστηριότητα, βάσει προκαθορισμένων κανόνων που εστιάζουν σε πρωτόκολλα υψηλού κινδύνου, όπως τα HTTP, ARP και ICMP, τα οποία είναι συχνά στόχοι ή φορείς επιθέσεων. Η στοχευμένη επιλογή πακέτων συμβάλλει στην αύξηση της αποτελεσματικότητας και στη μείωση του όγκου των δεδομένων, εστιάζοντας στην πιο ευάλωτη κίνηση.
- **Προεπεξεργασία Πακέτων (Packet Preprocessing):** Στο τελικό στάδιο, αφαιρούνται μη κρίσιμα πεδία και ολόκληρες επικεφαλίδες από κάθε πακέτο, ενώ όλα τα δείγματα προσαρμόζονται σε ένα ενιαίο μήκος μέσω τεχνικών περικοπής ή παραγεμίσματος με μηδενικά bytes (zero-padding), ώστε να εξασφαλίζεται η συμβατότητα με τις απαιτήσεις της αρχιτεκτονικής μετασχηματιστή. Στη συνέχεια, τα δεδομένα κανονικοποιούνται στην κλίμακα [0, 1], επιτρέποντας την ομαλή λειτουργία των ενεργοποιήσεων του μοντέλου και συμβάλλοντας στη σταθερότητα της εκπαίδευσης. Τέλος, εξάγονται για κάθε ροή οι χρονικές σφραγίδες (timestamps), πληροφορία που ενισχύει τη χρονική ευαισθησία του συστήματος και διευκολύνει την ανίχνευση χρονικών ανωμαλιών.

Η ανωτέρω διαδικασία επιτρέπει την κατασκευή μιας χρονικά ευαίσθητης, συμπαγούς και δομημένης αναπαράστασης της κίνησης, η οποία αποτελεί κατάλληλη είσοδο για το σύστημα ανίχνευσης, με γνώμονα τη βελτιστοποίηση της απόδοσης σε συνθήκες περιορισμένων πόρων.

Εκπαίδευση και Αξιολόγηση

Η διαδικασία της εκπαίδευσης αποσκοπεί στην κατασκευή του κατάλληλου μοντέλου για την έγκαιρη και αξιόπιστη ανίχνευση εισβολών, αξιοποιώντας τόσο το περιεχόμενο των πακέτων όσο και τις χρονικές πληροφορίες που αυτά ενσωματώνουν. Αρχικά, το σύνολο δεδομένων χωρίζεται στα υποσύνολα εκπαίδευσης και ελέγχου, ενώ παράλληλα όλες οι ροές εξομοιώνονται ως ακολουθίες σταθερού μήκους μέσω παραγεμίσματος με μηδενικά πακέτα για την ενεργοποίηση της εκπαίδευσης με δέσμες. Παράλληλα, δημιουργούνται μάσκες

προσοχής, οι οποίες επιτρέπουν στον μετασχηματιστή να διακρίνει τα ουσιώδη από τα τεχνητά πακέτα, διατηρώντας τη σημασιολογική εγκυρότητα της εισόδου.

Για την ενίσχυση της γενίκευσης του μοντέλου και τη βελτίωση της ικανότητας πρώιμης ανίχνευσης, εφαρμόζεται ένα πολυεπίπεδο σχήμα επαύξησης δεδομένων, το οποίο περιλαμβάνει τεχνικές τόσο σε στατικό όσο και σε δυναμικό επίπεδο. Οι στατικές τεχνικές περιλαμβάνουν τις ακόλουθες δύο λειτουργίες:

- **Δημιουργία Υπορροών (Subflow Generation):** Πρόκειται για επιμέρους ακολουθίες πακέτων από κάθε ροή, ώστε το μοντέλο να εκπαιδευτεί χρησιμοποιώντας ελλιπή πληροφορία.
- **Ντετερμινιστική Υπερδειγματοληψία (Deterministic Oversampling):** Στόχος είναι η αύξηση του μεγέθους του συνόλου δεδομένων εκπαίδευσης.

Οι δυναμικές τεχνικές επαύξησης, οι οποίες εφαρμόζονται κατά τη διάρκεια της εκπαίδευσης ανά εποχή και ανά ροή, περιλαμβάνουν:

1. **Τυχαία παραμόρφωση των χρονικών σφραγίδων** (jitter injection), ώστε να ενισχυθεί η ανθεκτικότητα του μοντέλου σε ασυνέπειες χρονισμού.
2. **Μεταβολή της έντασης της κίνησης** (traffic scaling), προσομοιώνοντας διαφορετικές συνθήκες φόρτου και εύρους ζώνης.
3. **Απομάκρυνση πακέτων** (packet drop), προκειμένου να αναπαρασταθούν σενάρια απώλειας πακέτων.
4. **Εισαγωγή μηδενικών πακέτων** (packet insertion), ώστε να προσομοιωθούν θόρυβος ή παρεμβολές στη ροή.
5. **Τυχαίες μεταβολές στις τιμές των bytes των πακέτων** (noise injection), ενισχύοντας την ανοχή σε σφάλματα ή παραποιήσεις του ωφέλιμου φορτίου.

Ιδιαίτερης σημασίας στην παρούσα προσέγγιση είναι η υιοθέτηση της συνάρτησης απώλειας έγκαιρης ανίχνευσης. Η συνάρτηση αυτή σχεδιάστηκε ώστε να εφαρμόζει αυστηρότερες ποινές στις εσφαλμένες προβλέψεις σύντομων ροών, καθοδηγώντας έτσι το μοντέλο να επιτυγχάνει ταχεία και ακριβή ταξινόμηση με ελάχιστο αριθμό πακέτων. Η ιδιότητα αυτή είναι κρίσιμη σε περιβάλλοντα πραγματικού χρόνου, όπου η καθυστέρηση στην ανίχνευση ενδέχεται να έχει ουσιώδεις επιπτώσεις στην ασφάλεια και τη σταθερότητα του συστήματος.

Σύστημα

Το προτεινόμενο σύστημα θεμελιώνεται σε μια ελαφριά και αποδοτική παραλλαγή της αρχιτεκτονικής του κωδικοποιητή του μετασχηματιστή, ειδικά σχεδιασμένη για την ανάλυση ροών δικτυακής κίνησης σε περιβάλλοντα με περιορισμένους υπολογιστικούς πόρους. Η αρχιτεκτονική του μοντέλου ακολουθεί μια μινιμαλιστική προσέγγιση και αποτελείται από ένα μόνο μπλοκ μετασχηματιστή, το οποίο ακολουθείται από ένα επίπεδο μέσης συγκέντρωσης (average pooling) και ένα πλήρως συνδεδεμένο επίπεδο ταξινόμησης. Ο συνολικός αριθμός παραμέτρων εκπαίδευσης ανέρχεται σε περίπου 5100, γεγονός που καταδεικνύει την υπολογιστική αποδοτικότητα και πρακτική αξιοποιησιμότητα του μοντέλου σε συστήματα IoT και ενσωματωμένες πλατφόρμες.

Ένα θεμελιώδες ζήτημα στις αρχιτεκτονικές μετασχηματιστών είναι η ανάγκη για κωδικοποίηση της θέσης των στοιχείων εισόδου, λόγω της απουσίας εγγενούς χρονικής ή σειριακής πληροφορίας. Η παρούσα εργασία εξετάζει τρεις καθιερωμένες τεχνικές κωδικοποίησης θέσης:

1. **Ημιτονοειδής κωδικοποίηση** (sinusoidal positional encoding).
2. **Κωδικοποίηση βασισμένη στον μετασχηματισμό Fourier** (Fourier-based positional encoding).
3. **Περιστροφική κωδικοποίηση** (rotary positional encoding – RoPE).

Κάθε μία από αυτές διακρίνεται ως προς τον τρόπο με τον οποίο ενσωματώνει χωρική πληροφορία στις αναπαραστάσεις των πακέτων. Για να καταστεί το μοντέλο ευαίσθητο στη χρονική δομή των δεδομένων, οι παραδοσιακές αυτές τεχνικές επεκτείνονται με την άμεση ενσωμάτωση των χρονικών σφραγίδων των πακέτων, με σκοπό την αξιοποίηση της ανομοιομορφίας στους χρόνους μεταξύ αφίξεων (inter-arrival times). Οι προτεινόμενες χρονικά ευαίσθητες παραλλαγές ενισχύουν την ικανότητα του μοντέλου να ανιχνεύει επιθέσεις που εκδηλώνονται ως χρονικά εξαρτώμενα μοτίβα.

Το τελικό σύστημα, με την ονομασία **A-THENA**, συνδυάζει την καταλληλότερη χρονικά ευαίσθητη κωδικοποίηση θέσης με τις τεχνικές επαύξησης δεδομένων που περιεγράφηκαν προηγουμένως. Ο υβριδικός αυτός σχεδιασμός, ο οποίος συνίσταται στην Χρονικά Ευαίσθητη Υβριδική Κωδικοποίηση (**T**ime-Aware **H**ybrid **E**ncoding – **THE**) και στην Επαύξηση για Δικτυακά Δεδομένα (**N**etwork-Specific **A**ugmentation – **NA**), καθιστά εφικτή την έγκαιρη, ακριβή και εύρωστη ανίχνευση εισβολών σε ρεαλιστικά περιβάλλοντα με περιορισμένη υπολογιστική ισχύ, όπως αυτά που χαρακτηρίζουν τα σύγχρονα δίκτυα IoT.

Ε.2 Πειραματική Μεθοδολογία

Η αξιολόγηση του A-THENA βασίζεται σε μια αυστηρή πειραματική μεθοδολογία, η οποία αξιοποιεί τρία πρότυπα σύνολα δεδομένων που αντανακλούν διαφορετικές διαστάσεις του προβλήματος ανίχνευσης εισβολών σε περιβάλλοντα IoT: CICIoT23-WEB, MQTT-IoT-IDS2020 και IoTID20. Τα εν λόγω σύνολα περιλαμβάνουν ένα ευρύ φάσμα επιθέσεων—από την εκμετάλλευση ευπαθειών σε εφαρμογές ιστού έως σενάρια κατανεμημένης άρνησης υπηρεσίας—παρέχοντας έτσι μία πολυδιάστατη αξιολόγηση της ικανότητας του μοντέλου να γενικεύει και να προσαρμόζεται απέναντι σε ετερογενείς και εξελισσόμενες απειλές.

Οι μετρικές αξιολόγησης επιλέχθηκαν ώστε να προσεγγίζουν ρεαλιστικά σενάρια εφαρμογών σε περιβάλλοντα IoT, εστιάζοντας τόσο στην ακρίβεια όσο και στην ταχύτητα ανίχνευσης. Οι βασικοί δείκτες επίδοσης βασίζονται σε ένα προκαθορισμένο κατώφλι πεποίθησης, το οποίο εφαρμόζεται στη μέγιστη τιμή πεποίθησης του μοντέλου, και περιλαμβάνουν:

- **Πρωιμότητα (earliness):** Ο αριθμός των πακέτων που απαιτούνται έως την πρώτη βέβαιη και ορθή πρόβλεψη.
- **Ακρίβεια (top-1 accuracy):** Η συνολική ακρίβεια με βάση την επικρατέστερη πρόβλεψη ταξινόμησης.
- **Ρυθμός ψευδώς αρνητικών δειγμάτων (false negative rate – FNR):** Το ποσοστό επιθέσεων που δεν εντοπίστηκαν και συνεπώς ταξινομήθηκαν εσφαλμένα ως καλοήθης κίνηση.
- **Ρυθμός ψευδώς συναγερμών (false alarm rate – FAR):** Το ποσοστό καλοήθους κίνησης που λανθασμένα αναγνωρίστηκε ως κακόβουλη.
- **Σφάλμα έγκαιρης ανίχνευσης κινδύνου (Early Risk Detection Error – ERDE):** Συνδυαστικός δείκτης που αποτιμά την απόδοση του συστήματος ως προς την ακρίβεια και την ταχύτητα λήψης σωστών αποφάσεων.

Προκειμένου να τεκμηριωθεί η πρακτική αξιοποιησιμότητα και καταλληλότητα του συστήματος για ανάπτυξη σε περιβάλλοντα με αυστηρούς περιορισμούς, το A-THENA

αναπτύσσεται και αξιολογείται στην πλατφόρμα Raspberry Pi Zero 2 W. Στο πλαίσιο αυτό, διενεργούνται μετρήσεις του χρόνου αναγνώρισης και της κατανάλωσης μνήμης, υπό πραγματικές συνθήκες λειτουργίας. Η πειραματική αυτή διάταξη παρέχει ισχυρά εμπειρικά τεκμήρια υπέρ της αποδοτικότητας και της βιωσιμότητας του συστήματος για χρήση σε εφαρμογές πραγματικού χρόνου σε ενσωματωμένα IoT περιβάλλοντα.

Ε.3 Αποτελέσματα

Το προτεινόμενο σύστημα A−THENA αξιολογείται συγκριτικά με επτά διαφορετικές στρατηγικές κωδικοποίησης θέσης, καθώς και με τέσσερα καθιερωμένα μοντέλα για έγκαιρη ανίχνευση εισβολών, γνωστά ως eRNN, eTransformer, eAtt και eGlo. Οι εξεταζόμενες τεχνικές κωδικοποίησης καλύπτουν ένα ευρύ φάσμα προσεγγίσεων, περιλαμβάνοντας τόσο παραδοσιακές, όσο και παραμετρικές/μαθησιακές τεχνικές, όπως η κωδικοποίηση με βάση διανυσματικές αναπαραστάσεις (embedding-based), η συνελικτική (convolutional) και η καθολικά σχετική (relative global) κωδικοποίηση. Σε αντίθεση με τα συγκρινόμενα μοντέλα της βιβλιογραφίας, το A−THENA είναι το μοναδικό μοντέλο που ενσωματώνει ρητά τη χρονική πληροφορία στη διαδικασία μάθησης, επιτρέποντας έτσι μια συστηματική και τεκμηριωμένη αξιολόγηση της επίδρασης των χρονικά ευαίσθητων κωδικοποιήσεων στην απόδοση.

Πρωιμότητα και Ακρίβεια

Η αξιολόγηση του συστήματος υπό κατώφλι εμπιστοσύνης 95% αναδεικνύει την υπεροχή του έναντι όλων των συγκρινόμενων μοντέλων και σε όλα τα χρησιμοποιούμενα σύνολα δεδομένων—CICIoT23-WEB, MQTT-IoT-IDS2020 και IoTID20. Συγκεκριμένα, η χρονικά ευαίσθητη υβριδική κωδικοποίηση θέσης που υιοθετεί το A−THENA οδηγεί σε βελτίωση της ακρίβειας έως και 18.57 μονάδες, σε σύγκριση με τις παραδοσιακές τεχνικές κωδικοποίησης. Η σημαντική αυτή απόδοση αποδίδεται στην ικανότητα του συστήματος να αποτυπώνει τις χρονικές συσχετίσεις στον ρυθμό άφιξης πακέτων, τις οποίες οι συμβατικές κωδικοποιήσεις αδυνατούν να συλλάβουν. Πέραν των κλασικών στρατηγικών, το A−THENA υπερέχει και έναντι των πλέον καθιερωμένων μοντέλων έγκαιρης ανίχνευσης εισβολών, παρουσιάζοντας μέση αύξηση ακρίβειας μεγαλύτερη των 6 μονάδων ανά σύνολο δεδομένων.

Ιδιαίτερο ενδιαφέρον παρουσιάζει η θετική συσχέτιση μεταξύ υψηλής ακρίβειας και μειωμένου αριθμού πακέτων που απαιτούνται για τη λήψη απόφασης (δείκτης πρωιμότητας), στοιχείο που καταδεικνύει την ικανότητα του μοντέλου να παρέχει πρώιμες και αξιόπιστες προβλέψεις. Παράλληλα, οι τιμές των δεικτών FAR και FNR διατηρούνται εξαιρετικά χαμηλές ή και μηδενικές, ιδίως στα σύνολα CICIoT23-WEB και MQTT-IoT-IDS2020, γεγονός που επιβεβαιώνει τον υψηλό βαθμό αξιοπιστίας του συστήματος.

Η τελική επιλογή κωδικοποίησης για κάθε σύνολο δεδομένων, βασισμένη στη χαμηλότερη απώλεια στο σύνολο επικύρωσης, υπογραμμίζει την αποτελεσματικότητα των χρονικά ευαίσθητων παραλλαγών. Ειδικότερα:

- **Για το CICIoT23-WEB**, η χρονικά ευαίσθητη ημιτονοειδής κωδικοποίηση οδήγησε σε 100% ακρίβεια με πρόβλεψη από το πρώτο πακέτο.
- **Για το MQTT-IoT-IDS2020**, η παραλλαγή τύπου Fourier επίσης παρείχε 100% ακρίβεια με ανίχνευση ήδη από το πρώτο πακέτο.
- **Για το IoTID20**, η παραλλαγή RoPE σημείωσε ακρίβεια 93.83%, διατηρώντας ταυτόχρονα χαμηλούς δείκτες σφαλμάτων.

Καθυστέρηση και Αποτύπωμα Μνήμης

Η ανάπτυξη του συστήματος A−THENA στο Raspberry Pi Zero 2 W ανέδειξε την ιδιαίτερα υψηλή αποδοτικότητά του, επιβεβαιώνοντας τη συμβατότητά του με περιβάλλοντα

περιορισμένων πόρων. Συγκεκριμένα, το μοντέλο απαιτεί μόλις 1.42 ms για την επεξεργασία ροών αποτελούμενων από 30 πακέτα, ενώ η συνολική κατανάλωση μνήμης παραμένει κάτω από τα 4 MB. Η καθυστέρηση παρουσιάζει μη γραμμική κλιμάκωση ως προς το μήκος της ροής, γεγονός που επιτρέπει τη χρήση του συστήματος ακόμη και σε δυναμικά περιβάλλοντα με μεταβαλλόμενο ρυθμό κίνησης. Αξιοσημείωτο είναι ότι η επιλογή στρατηγικής κωδικοποίησης θέσης έχει αμελητέο αντίκτυπο στους απαιτούμενους πόρους, στοιχείο που επιβεβαιώνει τη συμπαγή και αποδοτική σχεδίαση της αρχιτεκτονικής μετασχηματιστή που χρησιμοποιείται.

Σε σύγκριση με τα συναφή μοντέλα έγκαιρης ανίχνευσης εισβολών, το A−THENA επιτυγχάνει μια ιδανική ισορροπία ανάμεσα στον χρόνο απόκρισης, την ακρίβεια και τον αριθμό παραμέτρων. Μοντέλα όπως τα eAtt και eGlo, τα οποία βασίζονται σε συνελικτικές αρχιτεκτονικές, επιτυγχάνουν αντίστοιχους χρόνους απόκρισης, ωστόσο χρησιμοποιούν πολλαπλάσιο αριθμό παραμέτρων και παρουσιάζουν υποδεέστερη ακρίβεια ανίχνευσης. Αντιθέτως, πιο σύνθετα μοντέλα όπως τα eRNN και eTransformer εμφανίζουν υψηλή καθυστέρηση και υψηλές απαιτήσεις σε μνήμη, περιορίζοντας τη δυνατότητα υλοποίησής τους σε πραγματικές συσκευές IoT.

Αξιολόγηση Βασικών Συνιστωσών του A−THENA

Οι μελέτες απάλειψης (ablation studies) που πραγματοποιήθηκαν τεκμηριώνουν με σαφήνεια τη συμβολή κάθε βασικής συνιστώσας του συστήματος A−THENA στην εν γένει απόδοσή του. Συγκεκριμένα, η αφαίρεση των τεχνικών επαύξησης που βασίζονται σε χρονικές σφραγίδες, καθώς και η αντικατάσταση της προτεινόμενης συνάρτησης απώλειας EDeL με την κλασική διασταυρούμενη εντροπία (cross-entropy), οδηγούν σε σημαντική μείωση της ακρίβειας ανίχνευσης σε όλα τα αξιολογούμενα σύνολα δεδομένων. Το εύρημα αυτό υπογραμμίζει τη λειτουργική αναγκαιότητα των δύο αυτών μηχανισμών για την επίτευξη πρώιμης, αξιόπιστης και γενικεύσιμης ανίχνευσης εισβολών.

Η κβαντοποίηση του μοντέλου, η οποία εφαρμόστηκε μετά την εκπαίδευση βάσει του σχήματος INT8, αξιολογήθηκε ως προς την καθυστέρηση και την κατανάλωση μνήμης. Στην περίπτωση του A−THENA, η κβαντοποίηση προσέφερε επιτάχυνση κατά 1.37× στον χρόνο εκτέλεσης, χωρίς όμως να επιφέρει σημαντική μείωση στο αποτύπωμα μνήμης. Αντιθέτως, σε μεγαλύτερης κλίμακας αρχιτεκτονικές όπως το eTransformer, η κβαντοποίηση απέφερε 1.52× επιτάχυνση και 1.77× μείωση στη χρήση μνήμης, υποδηλώνοντας ότι τα οφέλη της τεχνικής αυτής είναι εντονότερα σε πιο απαιτητικά μοντέλα.

Συνολικά, τα πειραματικά αποτελέσματα αποδεικνύουν ότι το A−THENA είναι ήδη βελτιστοποιημένο τόσο σε επίπεδο αρχιτεκτονικής όσο και υπολογιστικά, παρουσιάζοντας περιορισμένο περιθώριο για περαιτέρω επιτάχυνση μέσω μεταγενέστερων τεχνικών συμπίεσης. Η συμπαγής αρχιτεκτονική και οι στοχευμένες σχεδιαστικές επιλογές καθιστούν το σύστημα ιδιαίτερα κατάλληλο για χρήση σε σενάρια πραγματικού χρόνου και για περιβάλλοντα με αυστηρούς ενεργειακούς περιορισμούς, όπως αυτά που συναντώνται στα σύγχρονα IoT συστήματα ασφάλειας.

Ε.4 Συμπέρασμα

Μέσω της ενσωμάτωσης της χρονικά ευαίσθητης υβριδικής κωδικοποίησης, της στοχευμένης επαύξησης δεδομένων προσαρμοσμένης σε δικτυακές ροές, και της εξειδικευμένης συνάρτησης απώλειας, το A−THENA καταφέρνει να αποτυπώνει με ακρίβεια τα χρονικά μοτίβα κακόβουλης δραστηριότητας, επιτυγχάνοντας υψηλή ακρίβεια με ελαχιστοποιημένα ποσοστά ψευδών θετικών και ψευδών αρνητικών δειγμάτων. Η πειραματική αξιολόγηση του συστήματος σε πολλαπλά σύνολα δεδομένων IoT, καθώς και η ανάπτυξή του σε υπολογιστικά

περιορισμένο ενσωματωμένο υλικό τεκμηριώνουν την πρακτική αξιοποιησιμότητα του σε ρεαλιστικά περιβάλλοντα. Η χαμηλή καθυστέρηση και η εξαιρετικά μικρή κατανάλωση μνήμης ενισχύουν περαιτέρω την καταλληλότητά του για λειτουργία σε πραγματικό χρόνο. Συνολικά, το A-THENA θέτει τις βάσεις για αξιόπιστη, αποδοτική και έγκαιρη ανίχνευση απειλών σε συνθήκες υψηλής μεταβλητότητας και περιορισμένων πόρων, όπως αυτές που χαρακτηρίζουν τον αναδυόμενο χώρο των ευφυών κατανεμημένων συστημάτων παρυφών.

ΣΤ Επίλογος

Η παρούσα διατριβή πραγματεύεται την επιτακτική ανάγκη καθιέρωσης της βαθιάς μάθησης ως βιώσιμης τεχνολογικής επιλογής σε κινητά και ενσωματωμένα υπολογιστικά περιβάλλοντα, όπου η υπολογιστική αποδοτικότητα και η προσαρμοστικότητα στους περιορισμούς πόρων συνιστούν αδιαπραγμάτευτες απαιτήσεις. Αναθεωρώντας τις υφιστάμενες πρακτικές ανάπτυξης και εκτέλεσης μοντέλων πέραν των παραδοσιακών υποδομών υπολογιστικού νέφους, η διατριβή εισάγει μια σειρά από καινοτόμες προσεγγίσεις, μεταξύ των οποίων: (α) το CARIn, ένα πλαίσιο για προσαρμοστική και πολυκριτηριακά βελτιστοποιημένη εκτέλεση συμπερασματολογίας σε κινητές συσκευές, (β) εξερεύνηση και βελτιστοποιήσεις σε μοντέλα τύπου μετασχηματιστή με στόχο τη λειτουργία τους στις παρυφές του δικτύου, καθώς και (γ) χρονικά ευαίσθητες κωδικοποιήσεις θέσης για την ενίσχυση της ασφάλειας σε δικτυακά περιβάλλοντα IoT.

ΣΤ.1 Τελικά Συμπεράσματα

Η παρούσα διατριβή αναδεικνύει ότι η αποδοτικότητα συνιστά θεμελιώδη προϋπόθεση για την πρόοδο της τεχνητής νοημοσύνης σε ρεαλιστικά περιβάλλοντα με περιορισμένους πόρους. Τα βασικά ευρήματα καταδεικνύουν την ανάγκη για ολιστικές στρατηγικές βελτιστοποίησης σε επίπεδο συστήματος, την επιτακτική απαίτηση ανασχεδιασμού των μετασχηματιστών ώστε να συμβαδίζουν με τους περιορισμούς των παρυφών, καθώς και τη σημασία της αποδοτικής βαθιάς μάθησης για την ενίσχυση της ταχύτητας και της αξιοπιστίας συστημάτων ασφάλειας σε εφαρμογές πραγματικού χρόνου. Συνολικά, τα συμπεράσματα της μελέτης υπογραμμίζουν ότι η βιώσιμη ανάπτυξη της τεχνητής νοημοσύνης απαιτεί έναν ριζικό επαναπροσδιορισμό τόσο των υποκείμενων αρχιτεκτονικών όσο και του τρόπου εκτέλεσής τους, ώστε να ανταποκρίνονται στις απαιτήσεις των σύγχρονων, δυναμικών και ετερογενών υπολογιστικών περιβαλλόντων.

ΣΤ.2 Μελλοντικές Επεκτάσεις

Η παρούσα διατριβή αναδεικνύει πολλαπλές ερευνητικές κατευθύνσεις με ιδιαίτερη δυναμική για την περαιτέρω προώθηση της αποδοτικής βαθιάς μάθησης σε κινητά και ενσωματωμένα συστήματα.

- **Μελλοντικές προσπάθειες θα πρέπει να επικεντρωθούν στην τοπική εκπαίδευση** (on-device training), επιδιώκοντας ελαφριά, εξατομικευμένη και προσαρμόσιμη μάθηση σε πραγματικό χρόνο. Μια τέτοια κατεύθυνση ενισχύει την προστασία της ιδιωτικότητας και τη δυνατότητα λειτουργίας υπό μεταβαλλόμενες συνθήκες, καθιστώντας τα συστήματα τεχνητής νοημοσύνης πιο ανταποκρίσιμα και ασφαλή.
- **Η ακριβής πρόβλεψη της απόδοσης νευρωνικών δικτύων** μέσω μαθησιακών μοντέλων αποτελεί άλλο ένα υποσχόμενο πεδίο. Τέτοιες προσεγγίσεις μπορούν να μειώσουν δραστικά το κόστος ανάπτυξης, να επιταχύνουν τη διαδικασία βελτιστοποίησης και να υποστηρίξουν την αποτελεσματική λήψη αποφάσεων χωρίς την ανάγκη για εξαντλητικές εκτελέσεις ή μετρήσεις.

- **Το ταχέως εξελισσόμενο πεδίο της παραγωγικής τεχνητής νοημοσύνης** (generative AI) σε κινητές και ενσωματωμένες πλατφόρμες προσφέρει νέες δυνατότητες για ιδιωτικές και άκρως προσωποποιημένες εφαρμογές. Ωστόσο, εξακολουθεί να αντιμετωπίζει σημαντικές προκλήσεις που σχετίζονται με τους περιορισμένους πόρους, την καθυστέρηση, την προσαρμοστικότητα και τη διαχείριση πολλαπλών μορφών πληροφορίας.
- **Εξίσου κρίσιμος αναδεικνύεται ο ρόλος της τεχνητής νοημοσύνης στα δικτυακά συστήματα**, η οποία διαμορφώνει τις βάσεις για ευφυείς, προσαρμοστικές και αυτοβελτιούμενες επικοινωνιακές υποδομές. Παρά τη δυναμική του, το πεδίο αυτό υπολείπεται κατάλληλων συνόλων δεδομένων, παραστατικών αναπαραστάσεων και εξειδικευμένων αρχιτεκτονικών. Η ανάγκη για εκμάθηση σε πραγματικό χρόνο και η έλλειψη εδραιωμένων μεθόδων συστηματικής συγκριτικής αξιολόγησης (benchmarking) καταδεικνύουν την ανάγκη για εντατικότερη έρευνα.
- **Περαιτέρω κατευθύνσεις** περιλαμβάνουν την ανάπτυξη υβριδικών υπολογιστικών σχημάτων που γεφυρώνουν τις παρυφές με το νέφος, τη βελτιστοποίηση με γνώμονα την ενεργειακή απόδοση, τη διασφάλιση της ανθεκτικότητας των μοντέλων επί της συσκευής, και την εμβάθυνση σε παραδείγματα αυτοεπιβλεπόμενης και συνεχούς μάθησης.

Συνολικά, οι παραπάνω κατευθύνσεις διευρύνουν το όραμα για κλιμακούμενη και αποδοτική τεχνητή νοημοσύνη στις παρυφές, συμβάλλοντας αποφασιστικά στην ανάπτυξη πραγματικά ευφυών και εφαρμόσιμων συστημάτων σε περιβάλλοντα με ρεαλιστικούς περιορισμούς.

Acknowledgements

The completion of this dissertation marks the end of an academically enriching journey—a journey not undertaken alone, but made possible by the invaluable contributions and unwavering support of many individuals and institutions.

First and foremost, I extend my deepest gratitude to my advisor, Professor Iakovos S. Venieris, whose exceptional guidance, profound insights, and constant encouragement were instrumental in shaping this research. His mentorship went beyond mere supervision, inspiring me to strive for rigor, originality, and practical relevance in every aspect of my work. His trust and understanding throughout this journey fostered a supportive environment that enabled me to grow both personally and professionally, encouraging me to confidently pursue ambitious goals. I am equally indebted to my co-advisor, Professor Dimitra Kaklamani, for her invaluable mentorship, intellectual generosity, and continuous encouragement throughout this journey. Her insightful discussions and constructive feedback significantly enriched my research experience and played a vital role in the successful completion of this dissertation.

I would also like to extend my sincere appreciation to the members of my dissertation committee for their meticulous feedback and constructive suggestions, which considerably improved the quality of my research and writing. Special thanks are due to my collaborators and colleagues, especially those with whom I co-authored research papers during my PhD. In particular, I would like to extend my heartfelt thanks to Dr. Stylianos Venieris, Senior Research Scientist at the Samsung AI Center in Cambridge, UK, for his invaluable guidance and support throughout the often-chaotic journey of research. His mentorship has played a pivotal role in shaping me into the researcher I am today.

This research would not have been possible without the support of the Intelligent Communications and Broadband Networks (ICBNet) Laboratory, which provided the resources, facilities, and academic environment necessary for pursuing innovative and impactful research. I am grateful to all the technical and administrative staff who facilitated my work with efficiency and kindness. Furthermore, I warmly thank my fellow graduate students, friends, and family for their intellectual companionship, patience, and emotional support throughout this journey. Their shared experiences, stimulating conversations, and unwavering encouragement have rendered my academic path genuinely enjoyable and memorable.

Lastly, I wish to recognize the broader research community whose collective efforts continue to drive forward the fields of deep learning, mobile computing, and embedded systems. This dissertation represents a modest contribution within a larger, collaborative endeavor, standing upon the foundation laid by numerous researchers who have pioneered the way for continuous innovation and discovery. I am deeply grateful for their contributions, as they provided invaluable inspiration and context for my own work and highlighted the essential role of collaborative progress in achieving meaningful scientific advancements.

Ioannis Panopoulos
Athens, April 2025

This research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI PhD Fellowships (Fellowship Number: 5578).

Table of Contents

Περίληψη **7**
Abstract **9**
Εκτεταμένη Περίληψη **11**
Α Εισαγωγή 11
Β Θεωρητικό Υπόβαθρο 11
Γ Τοπική Συμπερασματολογία σε Κινητές Συσκευές 16
Δ Εκτέλεση Μοντέλων Μετασχηματιστών σε Κινητές Συσκευές 21
Ε Έγκαιρη Ανίχνευση Εισβολών σε Δίκτυα IoT 25
ΣΤ Επίλογος 32
Acknowledgements **35**
Table of Contents **37**
List of Figures **41**
List of Tables **43**
1 Introduction **45**
1.1 Limitations of Existing Work 46
1.2 Motivation and Goals 47
1.3 Contributions 48
1.4 Dissertation Overview 48
2 Theoretical Background **51**
2.1 Deep Learning Fundamentals 51
2.1.1 Evolution 52
2.1.2 Core Architectures 53
2.1.3 Transformers 54
2.2 Edge Computing 58
2.2.1 Evolution 59
2.2.2 Mobile Computing 60
2.2.3 Embedded Computing 61
2.3 Edge Intelligence 63
2.3.1 Defining Characteristics 63
2.3.2 Types of Applications 65
2.3.2 Device Hardware 66
2.4 On-Device Inference 70
2.4.1 Benefits 71
2.4.2 Challenges 72
2.5 Efficient Deep Learning 74
2.5.1 Metrics of Interest 74

2.5.2 Compression Methods 75
3 Optimized On-Device Inference for Mobile Devices 79
3.1 Related Work 79
3.1.1 Limited Resources 80
3.1.2 Device Heterogeneity 81
3.1.3 Dynamic Environment 81
3.1.4 DNN Diversity 82
3.1.5 DNN Innovations 82
3.1.6 Diverse Application SLOs 82
3.1.7 Multi-DNN Inference 83
3.2 System Overview 83
3.2.1 Workflow 84
3.3 Multi-Objective Optimization Framework 86
3.3.1 MOO Problem Formulation 86
3.3.2 Objective Function Evaluation 88
3.3.3 MOO Problem Solver 89
3.4 Implementation 92
3.5 Experimental Methodology 93
3.5.1 Quantization 93
3.5.2 Application Scenarios, Models and Tasks 94
3.5.3 Mobile Devices 97
3.5.4 Profiling Details 98
3.6 Results 98
3.6.1 Designs 99
3.6.2 Runtime Adaptation 101
3.7 Limitations and Future Directions 104
3.8 Conclusion 105
4 Deploying Transformer-Based Models on Mobile Devices 107
4.1 Related Work 108
4.2 Experimental Methodology 108
4.2.1 Tasks and Models 109
4.2.2 Mobile Devices 110
4.2.3 Benchmarking Details 112
4.3 Results 113
4.3.1 On-Device Accuracy 113
4.3.2 CPU Performance 113
4.3.3 Accelerators 115
4.4 Discussion and Future Work 116
4.4.1 System Optimizations 116

4.4.2 Model Optimizations 117
4.5 Conclusion 119
5 Advancing Early Intrusion Detection for the Internet of Things 121
5.1 Preliminaries and Related Work 122
5.1.1 DL-Based Intrusion Detection 122
5.1.2 Augmentation Strategies for Cybersecurity Datasets 123
5.1.3 Transformer Positional Encodings 123
5.2 Proposed Approach 125
5.2.1 Data Preparation 126
5.2.2 Training 128
5.2.3 System 130
5.3 Implementation 133
5.4 Experimental Methodology 133
5.4.1 Datasets 133
5.4.2 Training Configuration 134
5.4.3 Evaluation Metrics 134
5.5 Results 135
5.5.1 Comparison Methods 135
5.5.2 Earliness and Accuracy 137
5.5.3 Latency and Memory Footprint 138
5.5.4 Evaluating Core Components of A-THENA 139
5.6 Conclusion 140
6 Discussion and Concluding Thoughts 143
6.1 Key Findings 143
6.2 Looking into the Future 144
6.2.1 On-Device Training 144
6.2.2 DNN Performance Prediction in Resource-Constrained Devices 148
6.2.3 Generative AI on Mobile and Embedded Platforms 151
6.2.4 AI-Enabled Networking 153
6.2.5 Additional Directions 154
6.3 Final Thoughts 155
6.4 Publications 155
6.4.1 Journal Articles 155
6.4.2 Peer-Reviewed Conference/Workshop Papers 155
6.4.3 Under Review/Accepted 156
Glossary 157
References 161

List of Figures

Figure 2.1 The perceptron 52
Figure 2.2 The Transformer architecture [28] 55
Figure 3.1 High-level workflow of CARIn 84
Figure 3.2 Toolflow for the evaluation of CARIn 93
Figure 3.3 UC1 evaluation 100
Figure 3.4 UC2 evaluation 100
Figure 3.5 UC3 evaluation 101
Figure 3.6 UC4 evaluation 101
Figure 3.7 CARIn's runtime behavior targeting the single-DNN UC1 scenario on S20 102
Figure 3.8 CARIn's runtime behavior targeting the multi-DNN UC3 scenario on A71 103
Figure 4.1 CPU throughput for Samsung A71 114
Figure 4.2 CPU throughput for Samsung S20 FE 114
Figure 5.1 The proposed A-THENA system 126
Figure 5.2 Time-aware positional encodings 132
Figure 5.3 A-THENA's confidence-based evaluation 137
Figure 5.4 Impact of augmentation, EDeL, and quantization on A-THENA's performance 139

List of Tables

Table 2.1 Quantization Schemes 76
Table 3.1 UC1 Models 95
Table 3.2 UC2 Models 95
Table 3.3 UC3 Models 96
Table 3.4 UC4 Models 97
Table 3.5 Target Devices 98
Table 3.6 Selected Designs and SP for the Single-DNN UC1 Scenario on S20 102
Table 3.7 Selected Designs and SP for the Multi-DNN UC3 Scenario on A71 103
Table 3.8 `OODIn`'s Solving Time in Milliseconds 104
Table 3.9 Storage Requirements of `CARIn` and `OODIn` in MB 104
Table 4.1 Transformer Architectural Parameters 109
Table 4.2 Transformer Models 109
Table 4.3 Target Smartphones 110
Table 4.4 Quantization-Delegate Compatibility 112
Table 4.5 Impact of Quantization on CPU 115
Table 4.6 Accelerator Latency Speedup 115
Table 4.7 Accelerator Memory Increase 116
Table 4.8 Model Optimizations 118
Table 5.1 System Hyperparameters and their Corresponding Values 131
Table 5.2 Attack Categories in the Selected Datasets 134
Table 5.3 Summary of Considered Positional Encodings 136
Table 5.4 Model Complexity and Efficiency Comparison 138
Table 5.5 Quantization Benefits 140
Table 6.1 Processor-Specific Latency Prediction Models 149

1 Introduction

Technology has always evolved in response to the limitations of its time, pushing against the boundaries of what was once thought impossible. The history of innovation is a constant interplay between ambition and constraint, where every leap forward in capability is met with new barriers to overcome. Nowhere is this dynamic more apparent than in artificial intelligence, a field that has transformed from theoretical speculation into a defining force of the modern era [1]. The pursuit of machine intelligence has always been a balance between power and efficiency, between complexity and usability. Each breakthrough—whether in neural networks, statistical learning, or modern deep learning—has introduced new opportunities while exposing new limitations, driving the need for further refinement.

Deep learning, once a niche research domain, has now become the backbone of contemporary AI, fueling advancements in natural language processing, computer vision, and autonomous decision-making. It has reshaped industries, powered scientific discoveries, and even redefined how humans interact with technology. However, the progress of deep learning has been accompanied by an insatiable hunger for computational resources. Modern deep learning models are larger than ever [2], demanding massive amounts of memory, processing power, and energy. As they grow in complexity, they increasingly rely on high-performance cloud infrastructures, where vast computing resources can accommodate their requirements. While this has enabled impressive feats of artificial intelligence, it has also raised critical concerns about accessibility, latency, privacy, and sustainability. Many real-world applications require AI to function in real time, closer to the user, and without excessive reliance on centralized computing resources. The need for intelligence at the edge has never been greater.

Mobile and embedded computing present a unique frontier for deep learning [3], one that demands a shift in focus—from sheer power to efficiency, from abstraction to real-world adaptability. These devices are ubiquitous, integrated into everyday life through smartphones, IoT sensors, wearable devices, autonomous systems, and specialized edge hardware. They serve as the interface between digital intelligence and the physical world, enabling AI-powered applications that range from personal assistants and healthcare monitoring to industrial automation and intelligent surveillance. The potential is enormous, but so are the challenges. Deploying deep learning in these environments requires overcoming severe hardware constraints, including limited computational resources, energy consumption restrictions, and device heterogeneity [4]. Unlike cloud-based AI, where computational power can be scaled almost indefinitely, edge AI must function within strict operational limits, balancing performance with efficiency in ways that traditional deep learning paradigms were never designed to handle.

Bringing intelligence to these constrained environments—on-device inference—offers transformative possibilities [5]. It enables real-time processing without reliance on external servers, reducing latency and ensuring privacy. It creates AI systems that are more resilient and independent, capable of functioning even in disconnected or bandwidth-limited settings. Yet, achieving this vision is not a trivial task. The challenges of deploying deep learning in mobile and embedded computing systems are numerous: how to reduce model size without sacrificing accuracy, how to optimize execution for diverse and often unpredictable hardware, how to

manage power consumption while maintaining responsiveness, and how to enable adaptive AI systems that can learn and evolve within limited computational budgets. The question is no longer just how to make AI more intelligent, but how to make it efficient, scalable, and adaptable to the real-world constraints of mobile and embedded systems. As deep learning continues to evolve, its future will not be defined solely by ever-larger models trained on massive datasets in the cloud, but by its ability to operate effectively in diverse, distributed, and resource-limited environments. Efficiency is not merely an optimization goal—it is the key to unlocking the full potential of AI, ensuring that intelligence is not confined to powerful servers but integrated seamlessly into the fabric of everyday technology.

This dissertation explores the intersection of deep learning and efficiency, focusing on how advanced AI models can be optimized, adapted, and deployed in mobile and embedded computing environments. It delves into the technical and theoretical challenges that arise when bringing state-of-the-art AI beyond the data center, proposing novel solutions that bridge the gap between cutting-edge deep learning research and real-world deployment. By addressing these challenges, this research aims to contribute to a future where AI is not just powerful, but practical—where intelligence is embedded into the devices that shape our lives, operating seamlessly, efficiently, and sustainably.

1.1 Limitations of Existing Work

Although deep learning has seen rapid progress and increasing integration into mobile and embedded systems, several critical gaps remain unaddressed in the current research landscape. While numerous studies have focused on optimizing individual models or specific tasks, they often overlook broader, system-level challenges. These limitations become especially apparent when considering the practical demands of deploying advanced deep learning architectures on resource-constrained devices, supporting a wide range of applications, and maintaining efficient, scalable performance in real-world environments. The following highlights three major limitations in existing work that motivate the direction of this dissertation:

- **Lack of a unified system for real-world mobile inference:** Existing research often targets isolated optimization problems, such as improving the efficiency of single models or specific tasks. However, the broader need for integrated solutions that can jointly handle multi-DNN execution, adapt to heterogeneous hardware, and manage dynamic resource allocation in mobile environments remains largely unaddressed [6], [7]. Real-world edge computing scenarios frequently require multiple deep learning models to run simultaneously, each with different computational requirements and execution priorities. Furthermore, mobile and embedded devices operate under varying power, memory, and processing constraints, often requiring real-time adaptation to fluctuating hardware availability and environmental conditions. Current research lacks integrated frameworks that can dynamically manage these factors, ensuring optimal model execution without compromising efficiency or performance.
- **Limited evaluation of state-of-the-art architectures in resource-constrained settings:** Although Transformer models have become central to many recent advances in modern AI, efforts to tailor them effectively for mobile and embedded inference are still emerging. While extensive research has explored their use in cloud-based applications [8], [9], comparatively less attention has been given to their practical deployment on resource-constrained devices. Transformers present unique challenges due to their high memory and computational demands, which can complicate their fit with current mobile AI accelerators. Although techniques like quantization and

hardware acceleration offer promising avenues for improving efficiency, their application to Transformer-based architectures in mobile settings has yet to be thoroughly explored. Consequently, the empirical understanding of how state-of-the-art deep learning models perform under mobile constraints remains limited, potentially slowing broader adoption in real-world applications.

- **Underexplored domains such as networking applications:** While deep learning has seen extensive application in computer vision, speech processing, and natural language understanding, its use in domains such as networking and cybersecurity has so far received comparatively less attention [10]. In particular, real-time intrusion detection in IoT environments—where deep learning could provide adaptive, intelligent security solutions—has received significantly less attention compared to other AI-driven tasks. Securing IoT networks calls for low-latency, resource-efficient models capable of detecting anomalous behavior in real time; however, many existing approaches only partially address the stringent resource constraints characteristic of IoT hardware.

These persistent gaps in research and implementation hinder the widespread adoption of deep learning in mobile and embedded systems. Overcoming these limitations requires a shift from isolated model-level optimizations to a broader, system-level perspective—one that considers the full execution pipeline, from model design and hardware compatibility to adaptive resource management and multi-model integration. By addressing these challenges, deep learning can move beyond the confines of high-performance computing centers and become a truly ubiquitous, efficient, and adaptable technology in real-world edge applications.

1.2 Motivation and Goals

This research is motivated by the pressing need to address the broader set of challenges surrounding the deployment of deep learning in edge environments. As outlined in Section 1.1, these challenges span multiple dimensions—from system-level efficiency and architectural adaptation to the expansion of deep learning into underexplored application domains. Meeting these demands calls for a comprehensive approach that goes beyond isolated solutions, aiming instead to develop strategies that enhance performance, scalability, and applicability across diverse edge computing scenarios. This work is shaped by three key objectives:

- **Designing adaptive inference frameworks for heterogeneous mobile platforms**: This dissertation targets the development of adaptive inference frameworks that go beyond individual model optimizations to address multi-DNN execution, dynamic scheduling, and real-time resource management, aiming to enable seamless and efficient AI deployment across diverse mobile and embedded systems.
- **Bridging cutting-edge architectures with real-world constraints**: While deep learning architectures such as Transformers continue to evolve, their practical deployment on resource-limited devices hinges on the development of targeted optimization and acceleration strategies—an area this dissertation aims to investigate in the context of mobile execution.
- **Expanding efficient deep learning to underexplored application domains:** Real-time security applications, such as intrusion detection in IoT environments, demand AI solutions that balance computational efficiency with responsiveness. This research aims to explore how efficient deep learning techniques can be adapted to enhance security in networked systems while maintaining low computational overhead.

Beyond edge computing, the scope of efficient deep learning has expanded to encompass cloud-based optimization, hybrid cloud-edge architectures, and energy-efficient AI systems. As deep learning models continue to grow in complexity and their deployment scenarios diversify, efficiency remains a central concern. Whether in resource-constrained edge environments, large-scale cloud infrastructures, or distributed AI frameworks, the ability to balance computational cost, performance, and adaptability is crucial for ensuring the sustainability and accessibility of AI-driven technologies. This field will continue to be a vital and transformative area of research, shaping the future of AI-powered applications across industries. The work presented in this dissertation is driven by the opportunity to contribute to this ongoing transformation—by advancing efficient deep learning methodologies, addressing real-world constraints, and enabling next-generation intelligent systems to operate effectively across a wide range of computing environments.

1.3 Contributions

This dissertation is built upon three research studies, each addressing a critical aspect of efficient deep learning in mobile and embedded computing:

1. The first study, ***CARIn: Constraint-Aware and Responsive Inference on Heterogeneous Devices for Single- and Multi-DNN Workloads*** [11], proposes a unified system for deep learning inference in dynamic, heterogeneous mobile environments. It introduces an approach that supports multi-DNN execution, adapting in real time to hardware and resource constraints, ensuring efficient utilization of available computational power.
2. The second study, ***Exploring the Performance and Efficiency of Transformer Models for NLP on Mobile Devices*** [12], evaluates the real-world feasibility of Transformer architectures on mobile hardware. It examines their execution efficiency, uncovering key quantization and hardware compatibility bottlenecks, and highlights areas where improvements are needed to make these models more practical for mobile deployment.
3. The third study, ***A-THENA: Early Intrusion Detection for IoT with Time-Aware Hybrid Encoding and Network-Specific Augmentation***, represents a research effort that, at the time of writing, is under peer review. It introduces novel time-aware positional encodings that enhance the efficiency and effectiveness of deep learning models in security-related applications. By optimizing Transformer-based architectures, this research demonstrates how real-time intrusion detection can be improved in edge environments without excessive computational overhead.

Together, these contributions establish a comprehensive approach to efficient deep learning, spanning system-level optimization, architectural evaluation, and application-driven advancements. By integrating insights from these studies, this dissertation aims to bridge the gap between cutting-edge AI research and real-world deployment in mobile and embedded systems, ensuring that deep learning can operate effectively in constrained and dynamic computing environments.

1.4 Dissertation Overview

The structure of this dissertation is designed to provide a comprehensive exploration of efficient deep learning for on-device inference, presenting the conducted research, key findings, and broader implications for future advancements in the field. The chapters are organized to guide the reader through the theoretical foundations, methodological innovations, and experimental

results, ultimately building a cohesive narrative that underscores the importance of optimizing deep learning for mobile and embedded computing environments. The dissertation is structured as follows:

- **Chapter 1** serves as an introduction, establishing the research problem, motivation, and contributions. It presents the broader context of the study, emphasizing the necessity of efficient deep learning techniques and outlining the specific research questions addressed in the subsequent chapters.
- **Chapter 2** provides the foundational background necessary to understand the field of efficient deep learning. It highlights the significance of deploying deep learning models in resource-constrained environments, identifies the key challenges associated with on-device inference, and discusses state-of-the-art techniques used to improve model efficiency.
- **Chapter 3** introduces `CARIn`, an inference framework designed to address the multifaceted challenges of deep learning deployment on mobile devices. This chapter details the design, implementation, and evaluation of `CARIn`, demonstrating its ability to manage heterogeneous hardware constraints, dynamic resource availability, and multi-DNN execution. The framework represents a system-level approach to efficient deep learning, ensuring that mobile devices can execute deep learning workloads adaptively and optimally under varying operational conditions.
- **Chapter 4** presents early-stage research on optimizing Transformer architectures for edge deployment. Given their computational complexity and memory-intensive nature, Transformers present significant challenges when applied to mobile and embedded devices. This chapter examines these limitations, evaluates their impact on model deployment, and presents preliminary optimization strategies aimed at making Transformers more compatible with constrained environments.
- **Chapter 5** proposes `A-THENA`, an early intrusion detection system designed specifically for resource-constrained IoT devices. The research presented in this chapter demonstrates how specialized time-aware positional encodings, as well as network-specific augmentation can improve both the accuracy and efficiency of deep learning models in security applications, advancing the broader use of Transformers in IoT-based intrusion detection systems.
- **Chapter 6** concludes the dissertation by summarizing the key insights gained from the research, discussing its broader impact, and proposing future research directions. This final chapter reflects on the challenges that remain in the field of efficient deep learning and considers potential advancements in hardware acceleration, algorithmic efficiency, and application-specific optimizations that could further enhance deep learning's viability in real-world edge deployments.

Together, these chapters provide a structured and in-depth examination of efficient deep learning, bridging theoretical foundations with experimental findings and practical applications. By addressing both fundamental challenges and emerging opportunities, this dissertation contributes to the advancement of deep learning beyond cloud computing, ensuring that AI can function efficiently across diverse, resource-limited environments.

2 Theoretical Background

This chapter provides the theoretical foundations for deploying deep learning in resource-constrained environments. It begins with an overview of deep learning, covering its evolution and the rise of Transformers in modern AI applications. The discussion then shifts to edge computing, exploring its role in enabling real-time AI on mobile and embedded devices. Building on this, edge intelligence is introduced as the convergence of edge computing and AI, highlighting key applications and specialized hardware. The chapter then examines on-device inference, outlining its benefits—low latency, privacy, and reduced cloud dependency—alongside challenges like hardware limitations, energy constraints, and dynamic resource availability. Finally, the focus turns to efficient deep learning, emphasizing key metrics such as accuracy, latency, and power efficiency, along with compression techniques, such as quantization, which are essential for deploying AI in constrained environments.

2.1 Deep Learning Fundamentals

Artificial intelligence (AI) has long been driven by the ambition to create computational systems capable of learning from data and making intelligent decisions. Among the various AI methodologies, deep learning (DL) has emerged as the most transformative, revolutionizing fields such as computer vision [13], natural language processing (NLP) [14], autonomous systems [15], and healthcare [16]. At its core, DL is a specialized subset of machine learning (ML) that utilizes artificial neural networks with multiple layers to extract and represent complex patterns from data. The hierarchical nature of DL models allows them to automatically discover intricate structures within raw input, distinguishing them from traditional ML approaches that typically depend on manually engineered features and predefined statistical assumptions. Unlike conventional ML models, which require extensive human intervention to define relevant features, DL introduces several key advantages that have contributed to its dominance in modern AI:

- **Automated feature extraction:** Deep networks learn to identify relevant patterns directly from raw data, reducing reliance on domain expertise for feature engineering.
- **Scalability:** As datasets and computational power grow, DL models continue to improve in performance, whereas traditional models often reach a plateau.
- **End-to-end learning:** Many DL architectures can process raw input and generate predictions without requiring complex, intermediate manual steps.
- **Generalization capability:** When trained effectively, deep networks can adapt across diverse tasks, often outperforming conventional methods with minimal human intervention.
- **Handling high-dimensional data:** DL models can efficiently process very large and complex inputs that would overwhelm or underperform with traditional models.

These characteristics have positioned DL as the cornerstone of modern AI, enabling models to achieve or surpass human-level performance in tasks such as image recognition, language modeling, and strategic decision-making.

2.1.1 Evolution

The foundations of deep learning can be traced back to the 1950s, when researchers began exploring computational models inspired by the functioning of biological neurons. Early artificial neural networks were designed to mimic the way the human brain processes information, relying on interconnected units, or "neurons," that perform simple mathematical operations to recognize patterns in data. Among the earliest models was the Perceptron [17], introduced by Frank Rosenblatt in 1958, which was capable of learning basic relationships between inputs and outputs through an adaptive learning process. The perceptron, as illustrated in Figure 2.1, was a single-layer neural network that used a weighted sum of inputs followed by an activation function f to determine class predictions. It demonstrated that computational systems could learn from experience, a fundamental principle that would later define ML. Despite its initial promise, the perceptron had fundamental limitations. Most notably, it was incapable of handling non-linearly separable problems, as famously demonstrated by Minsky and Papert (1969) [18] in their proof that a perceptron could not solve the XOR problem—a simple logical function requiring a non-linear decision boundary. This realization led to a decline in research interest in neural networks, as many viewed them as inadequate for solving complex learning tasks.

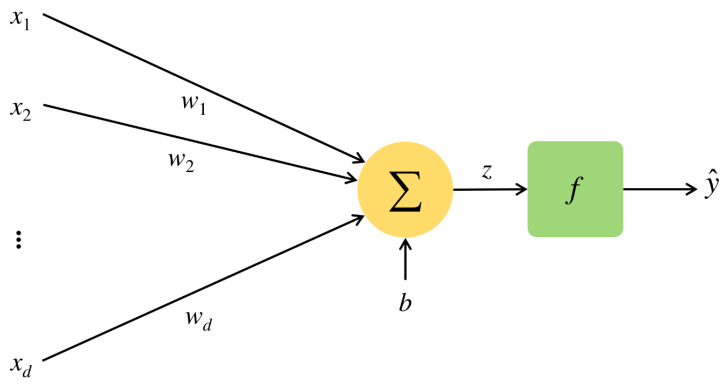

Figure 2.1 The perceptron

The field experienced a resurgence in the 1980s with the development of multilayer perceptrons (MLPs) and the backpropagation algorithm. MLPs introduced hidden layers between the input and output layers, allowing the network to learn hierarchical representations and model more complex functions. However, the key breakthrough that made MLPs practical was the backpropagation algorithm, which enabled efficient gradient-based learning. This technique, first formalized by Rumelhart, Hinton, and Williams (1986) [19], allowed deep networks to adjust their weights through error propagation, significantly improving their ability to learn from data.

Despite these advancements, training deep neural networks (DNNs) remained computationally prohibitive due to hardware limitations. The lack of high-performance computing resources and the inefficiency of early optimization techniques made it difficult to train deep models on large datasets. Consequently, neural networks were often outperformed by other ML techniques, such as support vector machines (SVMs) [20] and decision trees [21], which were computationally more efficient at the time. It was not until the early 2000s and 2010s, with the availability of large-scale datasets, GPU-accelerated training, and significant architectural advancements, that DL was able to achieve its full potential and become the dominant paradigm in AI.

2.1.2 Core Architectures

Over the decades, DL architectures have evolved to address different types of data and computational challenges. While early models such as multilayer perceptrons laid the groundwork for DL, they struggled with scalability and complex data structures. Later advancements introduced specialized architectures like convolutional neural networks for spatial data processing and recurrent neural networks for sequential data, significantly improving the ability of neural networks to handle image and time-dependent tasks. More recently, the advent of Transformer models has redefined state-of-the-art performance across multiple domains, particularly in NLP and computer vision.

2.1.2.1 Multilayer Perceptrons

Multilayer perceptrons (MLPs) represent the most basic form of DNNs, consisting of multiple layers of neurons connected in a fully dense fashion. Each neuron in an MLP applies a weighted sum of its inputs followed by a non-linear activation function such as ReLU (Rectified Linear Unit) or sigmoid. MLPs are capable of learning complex functions by stacking multiple layers, where each layer extracts increasingly abstract representations of the data. Despite their foundational role in DL, MLPs suffer from several limitations:

- **Lack of spatial awareness**: MLPs treat all input features equally, making them inefficient for structured data such as images, where spatial relationships are crucial.
- **Computational inefficiency:** Fully connected layers require a large number of parameters, leading to increased memory consumption and training time.
- **Poor scalability:** As the network depth increases, training deep MLPs becomes difficult due to issues like vanishing and exploding gradients, making optimization challenging.

Due to these constraints, MLPs are rarely used in isolation for modern DL tasks. Instead, specialized architectures have been developed to handle specific types of structured data more efficiently.

2.1.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) were designed to overcome the limitations of MLPs in processing spatial data, particularly images and videos. Instead of treating each pixel independently, CNNs leverage local connectivity and weight sharing through convolutional layers, allowing them to capture spatial hierarchies in data more efficiently. Key components of CNNs include:

- **Convolutional layers:** Perform localized feature extraction by applying filters (kernels) that detect edges, textures, and higher-level patterns.
- **Pooling layers:** Downsample feature maps to reduce computational cost and increase spatial invariance.
- **Fully connected layers:** Process high-level extracted features for classification or decision-making.

CNNs have been instrumental in computer vision, enabling breakthroughs in image recognition, object detection, medical imaging, and facial recognition. Architectures such as LeNet [22], AlexNet [23], VGG [24], ResNet [25], and EfficientNet [26] have set new benchmarks for performance and efficiency. However, CNNs still face challenges in handling sequential and long-range dependencies, necessitating architectures better suited for temporal and linguistic data.

2.1.2.3 Recurrent Neural Networks

While CNNs excel in spatial feature extraction, they struggle with sequential dependencies, making them unsuitable for tasks such as speech recognition, language modeling, and time-series prediction. Recurrent neural networks (RNNs) were introduced to address this issue by incorporating memory mechanisms, allowing them to retain information from previous time steps when processing sequential data. However, standard RNNs suffer from the vanishing gradient problem, which makes it difficult for them to capture long-term dependencies in sequences. To mitigate this issue, long short-term memory (LSTM) networks were introduced [27]. LSTMs utilize gated mechanisms—such as the forget, input, and output gates—to regulate information flow, enabling them to learn longer-term dependencies more effectively than traditional RNNs. Despite their success, RNNs and LSTMs are inherently sequential in nature, making them computationally inefficient for large-scale training and inference. The inability to parallelize computations effectively led to the search for more scalable architectures, culminating in the development of Transformers.

2.1.3 Transformers

Deep learning has undergone a series of architectural shifts, each designed to overcome limitations in previous models. While RNNs and LSTM networks were initially dominant in sequence modeling tasks, they suffered from fundamental inefficiencies that limited their scalability. The Transformer architecture, introduced by Vaswani et al. (2017) in the paper *"Attention is All you Need"* [28], revolutionized the field by eliminating recurrence altogether, enabling parallel computation and significantly improving the ability to model long-range dependencies. Since their introduction, Transformers have become the backbone of modern DL, powering state-of-the-art models in NLP, computer vision, and multimodal AI.

2.1.3.1 Motivation

Prior to Transformers, RNNs and LSTMs were the standard architectures for processing sequential data. These models operated recursively, processing input sequences one step at a time while maintaining an internal memory of past elements. However, they faced significant drawbacks:

- **Sequential computation bottleneck:** Since RNNs process input tokens sequentially, they cannot be parallelized effectively, leading to computationally intensive and slow training and inference.
- **Difficulty in capturing long-range dependencies:** Despite LSTMs improving memory retention, they still struggle with learning dependencies in very long sequences, especially when context is separated by hundreds of tokens.
- **Gradient vanishing and explosion:** As the number of time steps increases, gradients may diminish or explode, making training unstable and less effective.

Transformers overcame these challenges by introducing self-attention, which allows the model to process all input tokens simultaneously, rather than sequentially. This design eliminates the need for recurrence, making training significantly faster and enabling better modeling of long-range relationships within data.

2.1.3.2 Key Components

The Transformer architecture departs from the traditional recurrent design and instead relies on a fully attention-based mechanism. Its core innovations—self-attention, positional encoding, multi-head attention, and feedforward networks—work together to provide highly efficient and

scalable sequence modeling. As illustrated in Figure 2.2, the Transformer consists of two primary components: the encoder and the decoder. The encoder processes the input sequence, transforming it into a fixed-length vector representation, often referred to as the context or memory, which encapsulates the essential information of the input. The decoder, on the other hand, generates the output sequence one token at a time. It does so by leveraging the encoder's representation as context to make informed predictions for each subsequent token in the output sequence.

Renowned for its sequence-to-sequence capabilities, the Transformer effectively maps input sequences to output sequences, making it particularly well-suited for tasks such as machine translation. In such applications, the model translates entire sequences (*e.g.*, sentences) from one language to another. Typically, a Transformer undergoes pre-training on large-scale tasks like language modeling or next-sentence prediction, where it learns to generate or infer missing information based on the surrounding context. Following pre-training, the encoder can be fine-tuned for various downstream tasks, including text classification, sentiment analysis, and even computer vision applications when adapted appropriately. This ability to first pre-train on general tasks and later fine-tune for specific applications is a defining strength of the Transformer, contributing to its versatility across a wide range of domains.

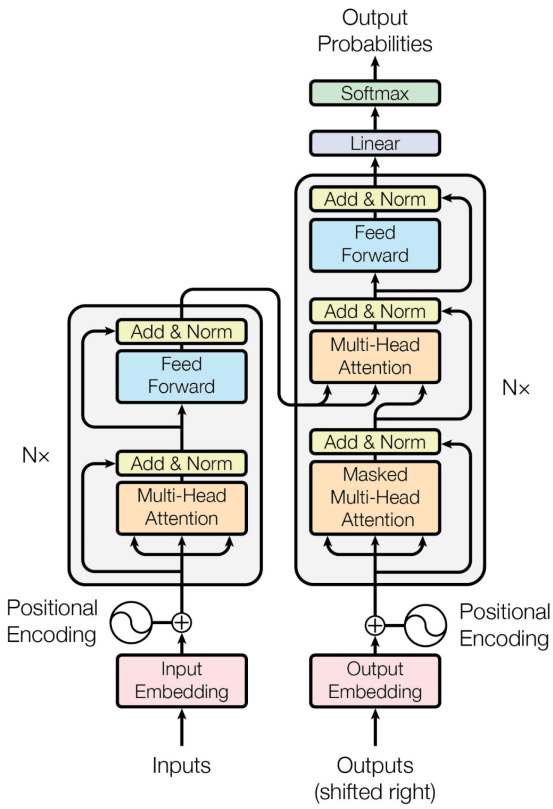

Figure 2.2 The Transformer architecture [28]

Encoder. This work focuses on representation learning, specifically within the context of classification tasks, making the encoder of the Transformer architecture the primary area of interest. The encoder is designed to process input sequences and transform them into meaningful, context-aware representations that can be leveraged for various downstream tasks. Unlike recurrent architectures, which process sequences step by step, the Transformer encoder utilizes a fully parallelizable attention mechanism, enabling more efficient and scalable learning.

The first component of the encoder is the *Input Embedding* layer, which maps discrete input tokens into dense vector representations of dimension d_{m} within a continuous space. This transformation allows the model to capture semantic similarities between tokens, as words with similar meanings will have embeddings that are closer in this learned space. Since Transformers inherently lack a sequential structure, they require an explicit way to encode positional information. This is achieved through *Positional Encodings*, which are typically added to the input embeddings. These encodings enable the model to differentiate the order of tokens while preserving their contextual relationships. Unlike recurrent models that process inputs sequentially, positional encodings allow the Transformer to generalize better to variable-length sequences, making it suitable for a wide range of applications. A more detailed discussion of positional encodings is provided in Subsection 5.1.3.

After positional encoding, the input sequence is passed through L Transformer encoder layers, each of which consists of the following key components:

- **Multi-Head Self-Attention Mechanism**
- **Position-Wise Feedforward Network (FFN)**
- **Residual Connections, Dropout, and Layer Normalization**

At the core of the Transformer encoder is the *Multi-Head Self-Attention* mechanism, which allows the model to compute dependencies between tokens in parallel. Self-attention enables direct interactions between all tokens, improving efficiency and long-range dependency modeling. Mathematically, self-attention is computed using three learnable matrices:

- **Query (*Q*):** Represents the current token's request for attention.
- **Key (*K*):** Encodes information about all tokens in the sequence.
- **Value (*V*):** Contains the actual token information that will be weighted and aggregated.

The attention scores are computed using the Scaled Dot-Product Attention formula:

$$Attention(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_{\mathrm{k}}}}\right)V \tag{2.1}$$

where d_{k} is the dimensionality of the key vectors, and the softmax function ensures that the attention weights sum to 1. This mechanism allows the model to dynamically focus on the most relevant tokens, irrespective of their position in the sequence.

To further enhance its learning capacity, the Transformer employs multi-head attention, where h self-attention mechanisms, each with a dimensionality of d_{h}, operate in parallel. Each attention head learns a different aspect of the input relationships, enabling a more comprehensive understanding of the sequence. The outputs from all heads are then concatenated and projected to a final representation:

$$MultiHead(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)W^{\mathrm{O}} \tag{2.2}$$

where W^{O} is a learned projection matrix. This mechanism improves the robustness of learned representations, allowing the model to capture both local and global dependencies simultaneously.

Following the self-attention mechanism, each encoder layer includes a position-wise *Feed Forward Network (FFN)*. This network consists of two fully connected layers with hidden dimension d_{ff}, separated by a non-linear activation function, typically GELU (Gaussian Error Linear Unit). The purpose of this component is to enhance the model's representational power

by introducing element-wise transformations. Unlike self-attention, which captures relationships between tokens, the feedforward network applies transformations independently to each token in the sequence, increasing the model's expressiveness.

To facilitate stable training and efficient gradient propagation, Transformers utilize *Residual Connections* around both the multi-head attention and feedforward network layers. The outputs of these sublayers are added to their respective inputs before being passed forward:

$$Output = \text{LayerNorm}\big(X + \text{SubLayer}(X)\big) \tag{2.3}$$

where LayerNorm refers to *Layer Normalization*, which ensures stabilized activations and helps prevent vanishing or exploding gradients. Additionally, dropout is applied after each residual connection to mitigate overfitting.

After passing through L Transformer encoder layers, the model produces a final set of token representations. The way these outputs are utilized depends on the specific task at hand:

- **For sequence classification**, the token-level representations need to be aggregated into a single fixed-length vector. This can be done via:
  - Global average pooling, where the embeddings of all tokens are averaged.
  - A special [CLS] token, which is introduced at the start (or end) of the sequence and learns a global representation for classification.
- **For token-level tasks** (*e.g.*, named entity recognition, part-of-speech tagging), the final output representations are used directly, with each token mapped to its respective label.
- **For other downstream tasks**, the encoder outputs can be fed into additional layers (*e.g.*, fully connected layers, attention mechanisms) to suit specific applications.

For classification, the final representation is passed through a fully connected layer with c output neurons, followed by a softmax activation function, which generates confidence scores for each class:

$$\hat{y} = \text{softmax}(W_{\text{out}}x + b) \tag{2.4}$$

where W_{out} and b are learnable parameters.

2.1.3.3 Impact

Since their introduction, Transformers have revolutionized AI, setting new benchmarks across multiple domains. Their impact extends far beyond NLP, influencing fields such as computer vision, multimodal AI, and even scientific research.

Natural Language Processing. Transformers have become the backbone of state-of-the-art NLP models, revolutionizing language understanding and generation at an unprecedented scale. Key advancements in this domain include BERT (Bidirectional Encoder Representations from Transformers) [29], which introduced context-aware word embeddings, significantly enhancing tasks such as sentiment analysis, question answering, and named entity recognition. GPT-4 [30] and T5 [31] further pushed the boundaries of natural language generation, enabling AI to produce human-like text, summarize information, and engage in coherent, contextually relevant conversations. Meanwhile, XLNet [32] introduced permutation-based pretraining, improving language modeling by capturing bidirectional context without relying on masking techniques. These innovations have had profound implications for chatbots, translation systems, search engines, and automated content creation, enabling AI-driven applications that closely mimic human language understanding and interaction.

Computer Vision. Traditionally, convolutional neural networks dominated computer vision tasks, excelling in image classification, object detection, and segmentation. However, the emergence of Vision Transformers has demonstrated that self-attention mechanisms can outperform CNNs, particularly in large-scale image recognition and generative tasks. ViT (Vision Transformer) [33] introduced self-attention to image patches, achieving state-of-the-art performance in image classification. DETR (DEtection TRansformer) [34] leveraged end-to-end self-attention for object detection, eliminating the need for traditional region proposal-based methods. Additionally, models like Stable Diffusion [35] and DALL·E [36] combined Transformers with diffusion models, enabling the generation of highly realistic images from textual descriptions. This shift underscores the transformative power of attention mechanisms in visual understanding, allowing models to capture long-range dependencies and global context more effectively than CNNs.

Multimodal AI. Beyond text and vision, Transformers have paved the way for multimodal AI, enabling models to process and generate information across multiple data types simultaneously. CLIP (Contrastive Language–Image Pretraining) [37] was trained on large-scale image-text pairs, allowing for zero-shot classification and cross-modal understanding, making it a breakthrough in aligning visual and linguistic representations. DALL·E [36] expanded on this capability by integrating text and image generation, producing highly detailed visuals from natural language descriptions. Meanwhile, Flamingo [38] advanced video-language modeling, enabling AI to analyze and generate video content based on textual prompts. By integrating multiple modalities, these Transformer-based systems bridge the gap between human cognition and machine intelligence, allowing AI to understand, generate, and interact with data in a far more contextually rich and meaningful way.

2.2 Edge Computing

The evolution of modern computing has been shaped by the need to balance processing power, data transfer efficiency, and latency. Traditionally, computing paradigms have relied on centralized architectures, where data are processed in large, remote data centers or cloud infrastructures. While cloud computing has revolutionized how data are stored, accessed, and processed, its dependence on high-speed network connectivity and centralized processing power introduces significant limitations in scenarios requiring real-time responsiveness, low latency, and high reliability. This has led to the emergence of edge computing, a paradigm that shifts computational resources closer to the source of data generation, reducing reliance on cloud servers and enabling faster, more localized processing.

Edge computing [39] represents a distributed computing model in which data processing and computational tasks occur at or near the data source, such as sensors, mobile devices, or edge servers. By minimizing the need to transmit large volumes of data to distant cloud infrastructures, edge computing reduces network congestion, bandwidth usage, and response times, making it an ideal solution for applications that require low latency and real-time decision-making. The demand for edge computing has been driven by several key technological and infrastructural challenges that traditional cloud-based models struggle to address:

- **Latency sensitivity:** Many applications, such as autonomous vehicles, industrial automation, and healthcare monitoring, require near-instantaneous responses. The delays introduced by data transmission to cloud servers can be detrimental in critical real-time applications.

- **Bandwidth optimization:** The exponential growth of Internet of Things devices has led to massive amounts of data being generated at the edge of networks. Transmitting all these data to centralized cloud infrastructures is inefficient and unsustainable. Processing data locally reduces unnecessary bandwidth consumption.
- **Data privacy and security:** Certain industries, such as finance, healthcare, and defense, require localized data processing to ensure privacy compliance and reduce risks associated with transmitting sensitive information over external networks.
- **Intermittent connectivity and reliability:** Applications deployed in remote locations, disaster recovery scenarios, or mobile environments cannot always rely on stable, high-speed network connections. Edge computing ensures that core processing capabilities remain operational even in limited-connectivity scenarios.
- **Scalability in a hyperconnected world:** As the number of connected devices increases with the expansion of 5G, IoT, and future 6G networks, relying solely on centralized cloud servers will become infeasible. Edge computing distributes processing power across the network, making it more scalable and resilient.

By addressing these challenges, edge computing bridges the gap between centralized cloud processing and real-world application needs, offering a hybrid approach where computation is dynamically distributed between cloud and edge nodes based on resource availability, network conditions, and application requirements.

2.2.1 Evolution

In the early days of computing, mainframe architectures dominated the landscape. Large, centralized computers housed all processing power, with users accessing them through "dumb" terminals—devices with no independent processing capability. This centralized approach ensured tight control over computation and data management but suffered from high latency and limited scalability, as all processing occurred in a single location. As computing technology advanced, the client-server model emerged, enabling distributed processing where clients (user devices) could perform some computations while relying on central servers for complex tasks. This model became the backbone of enterprise computing, allowing for greater scalability and flexibility. However, as internet connectivity expanded and more applications moved online, the need for a more scalable, remotely accessible computing model led to the rise of cloud computing.

Cloud computing revolutionized data storage and processing by centralizing resources in large-scale data centers, enabling users to access vast computational power on demand. The cloud model offered numerous benefits, including cost efficiency, flexibility, and high availability. However, it also introduced several critical limitations, particularly in scenarios requiring low-latency responses, high bandwidth availability, and continuous operation in environments with unreliable network access. The limitations of cloud computing led to the emergence of edge computing; a paradigm shift toward decentralization that enables devices to process data locally rather than constantly relying on cloud-based processing.

5G Networks. The advent of 5G technology has been a major catalyst for the widespread adoption of edge computing, providing the high-speed connectivity and low-latency performance necessary for real-time processing at the network edge. Unlike previous generations of wireless communication, which primarily focused on increasing data speeds, 5G is designed to support a vast and diverse ecosystem of connected devices, making it an essential enabler for smart cities, industrial automation, and next-generation mobile applications. Key

advantages of 5G in edge computing include (a) ultra-low latency (~1 millisecond), which is crucial for applications such as autonomous driving and remote robotics, (b) higher bandwidth, allowing seamless connectivity for a large number of distributed edge devices, and (c) network slicing, which enables customized network performance based on application needs, ensuring reliable real-time computation. The high-speed, low-latency capabilities of 5G networks make them ideal for edge computing applications that demand instant responsiveness, including smart factories, augmented reality (AR), and mission-critical healthcare systems, where real-time decision-making is essential.

IoT Growth. The Internet of Things (IoT) encompasses a vast network of connected sensors, embedded systems, and smart devices that continuously collect and transmit data. With billions of IoT devices deployed worldwide across industries such as smart cities, industrial automation, logistics, agriculture, and consumer electronics, the sheer volume of generated data presents a computational bottleneck if all processing were to rely solely on centralized cloud servers. By integrating edge computing with IoT, data processing is distributed across local devices, significantly reducing the need for excessive cloud communication. This approach optimizes bandwidth usage by transmitting only relevant data to the cloud, enhances response times for critical applications such as predictive maintenance and emergency response, and increases device autonomy, enabling real-time analytics and decision-making without constant connectivity. As IoT adoption continues to grow, edge computing plays a crucial role in ensuring the scalability, efficiency, and real-time responsiveness of connected systems.

2.2.2 Mobile Computing

Mobile computing refers to a computational paradigm that enables wireless, portable, and real-time access to computing resources, regardless of a user's physical location. Unlike traditional computing systems, which rely on fixed infrastructure, mobile computing provides ubiquitous access to networks, data, and applications while supporting seamless mobility. The defining characteristics of mobile computing include:

- **Portability:** Devices are designed to be lightweight and battery-powered, allowing users to perform computational tasks while on the move.
- **Wireless connectivity:** Mobile computing relies on technologies such as Wi-Fi, cellular networks (3G, 4G, 5G), and Bluetooth to facilitate communication and access to remote resources.
- **Real-time accessibility:** Users can interact with applications and services instantly, enabling real-time collaboration, data exchange, and decision-making.
- **Context awareness:** Many mobile applications leverage network connectivity, sensors, GPS, accelerometers, and environmental data to provide location-based and context-aware services.

The scope of mobile computing extends beyond personal devices such as smartphones and tablets. It includes wearable technology, vehicle-based computing, and mobile IoT devices, all of which rely on efficient computing to deliver real-time insights and seamless connectivity.

2.2.2.1 Key Components

Mobile computing encompasses a complex ecosystem consisting of devices, network infrastructure, and digital services that collectively enable a seamless mobile experience. The key components are described below.

Mobile Devices. Mobile computing revolves around portable electronic devices that provide computational capabilities while remaining lightweight and battery-powered. These devices range from consumer electronics to specialized industrial tools, enabling seamless connectivity and real-time data processing. Smartphones and tablets are the most widely used mobile computing devices, supporting communication, web browsing, and diverse applications. Wearable devices, such as smartwatches, fitness trackers, and AR headsets, enhance mobility by collecting and transmitting real-time health, fitness, and environmental data. Laptops and ultra-mobile PCs (UMPCs) offer greater computational power and versatility while maintaining portability, making them essential for mobile professionals and fieldwork applications. Additionally, vehicle-mounted and handheld industrial devices are extensively used in logistics, transportation, and remote operations, allowing workers to track inventory, navigate routes, and manage industrial workflows on the go. As mobile computing continues to evolve, advancements in battery life, connectivity, and processing power are expanding the potential of these devices across multiple industries.

Mobile Networks. Mobile computing depends on wireless networks to enable uninterrupted communication and seamless data transfer, ensuring that users and devices remain connected regardless of location. Cellular networks (3G, 4G, and 5G) provide wide-area coverage and support high-speed data transmission, making them essential for mobile communication and internet access. Wi-Fi and wireless local area networks (WLANs) offer high-speed connectivity in localized environments such as homes, offices, and public spaces, reducing reliance on cellular networks for data-intensive applications. Additionally, Bluetooth and Near Field Communication (NFC) facilitate short-range communication, enabling device pairing, contactless payments, and efficient data exchange between connected devices. As wireless technologies continue to advance, the increasing integration of 5G, next-generation Wi-Fi standards, and low-power wireless protocols is further enhancing the speed, reliability, and efficiency of mobile computing ecosystems.

Mobile Services and Applications. Mobile computing is powered by a vast ecosystem of applications, software platforms, and cloud-based services that enable users to access, process, and store data remotely. One key advancement is mobile cloud computing (MCC), which allows mobile devices to offload computationally intensive tasks to cloud servers, enhancing performance and extending battery life. Additionally, edge-assisted mobile applications leverage edge computing resources to reduce latency and network congestion, making real-time applications—such as gaming, AR/VR, and autonomous systems—more responsive and efficient. Mobile payment systems, including contactless payments, digital wallets, and mobile banking, have revolutionized financial transactions by offering secure, fast, and convenient payment methods. Furthermore, location-based services (LBS), such as real-time navigation, geofencing, and personalized content recommendations, enhance user experiences by utilizing GPS and environmental data to deliver context-aware services. These advancements are shaping a seamless, intelligent, and highly connected mobile computing landscape, driving ubiquitous access to digital resources and real-time decision-making.

2.2.3 Embedded Computing

Embedded computing is a critical component of modern digital systems, enabling specialized computation within dedicated devices that operate with minimal human intervention. Unlike general-purpose computing, where devices are designed for a wide range of tasks, embedded systems are tailored for specific applications, often performing real-time functions with high efficiency, reliability, and low power consumption. Embedded computing powers an array of

industrial, automotive, healthcare, and consumer electronics applications, making it one of the most pervasive and impactful areas of modern technology. Embedded systems are typically characterized by:

- **Specialization:** Unlike general-purpose computers, embedded systems are designed for specific applications, optimizing their performance, energy efficiency, and reliability for that function.
- **Real-time operation:** Many embedded systems require deterministic responses, ensuring computations are performed within a strict time frame (*e.g.*, automotive control systems, industrial automation).
- **Integration with hardware:** Embedded computers are tightly coupled with the physical hardware they control, often including sensors, actuators, and communication modules.
- **Resource constraints:** Embedded systems must operate within limited processing power, memory, and energy budgets, especially in battery-operated or remote deployments.
- **Minimal user interaction:** Most embedded systems function autonomously or with minimal human intervention (*e.g.*, pacemakers, traffic control systems).

Unlike mobile computing, which focuses on portability and wireless connectivity, embedded computing emphasizes efficiency, reliability, and real-time processing. These features make embedded systems indispensable in mission-critical applications, where failure or delays could result in catastrophic consequences.

2.2.3.1 Examples

Embedded computing is integral to industrial automation, where programmable logic controllers (PLCs) and SCADA systems manage manufacturing processes, power plants, and industrial machinery with real-time precision. The rise of Industrial IoT (IIoT) sensors has further optimized factory operations by enabling predictive maintenance and energy-efficient automation, reducing downtime and operational costs. In the automotive sector, embedded systems power electronic control units (ECUs) for engine management, braking, and emissions control, while advanced driver assistance systems (ADAS) handle collision detection, lane assistance, and adaptive cruise control. Modern vehicles also integrate in-vehicle infotainment (IVI) systems, providing navigation, multimedia, and voice-controlled interfaces. With the push toward autonomous driving, embedded processors now process real-time sensor fusion from LiDAR, radar, and cameras to enhance safety and automation.

Embedded computing also drives smart home technology and consumer electronics, powering smart thermostats, security systems, and wearable devices. These systems use real-time processing to optimize energy usage, enhance security, and monitor health metrics. Wearables like smartwatches and fitness trackers continuously track heart rate, sleep patterns, and fitness levels, transmitting real-time data with minimal power consumption. In healthcare, embedded systems enable life-critical applications, including pacemakers, medical imaging systems, and portable diagnostic devices. These systems ensure real-time patient monitoring and AI-assisted diagnostics, improving the accuracy and accessibility of medical care. As embedded computing continues to advance, it will play a pivotal role in enabling autonomous industrial systems, smart infrastructure, and next-generation medical technology.

2.3 Edge Intelligence

The integration of deep learning with edge computing is essential for the development of smart applications that offer advanced features such as personalization, context awareness, automation, and natural interactions [40]. These applications rely heavily on DL models to analyze and adapt to user behavior, environmental conditions, and task-specific requirements in real time. By leveraging edge computing, the computational power required to execute DL models is brought closer to the user, enabling low-latency processing and responsive functionality. This is particularly important for applications like autonomous vehicles, wearable health monitors, and smart home devices, where timely and adaptive decision-making is crucial. Without the close proximity of computation to the data source, the user experience could suffer from delays, interruptions, and inefficiencies that undermine the potential of intelligent applications.

Another critical reason for merging artificial intelligence and edge computing lies in the relationship between DL and data. To train high-performing models, DL requires vast amounts of data, which is increasingly generated at the edge through IoT devices, smartphones, and other connected systems. Traditional cloud-based training faces several drawbacks, including high bandwidth costs, increased latency in data transmission, and potential privacy risks associated with transferring sensitive information to centralized servers. Furthermore, the growing volume of data generated at the edge makes cloud-only approaches less scalable. By processing and partially training models at the edge, these challenges can be mitigated. Local data handling reduces dependency on cloud infrastructure, enhances privacy, and allows for more efficient use of resources, paving the way for smarter, more secure, and adaptive AI-driven applications.

Overall, the convergence of AI and EC is driving a new era of technological innovation, enabling DL to realize its full potential across a broad spectrum of applications and use cases. This integration has given rise to a specialized domain known as "edge intelligence." Edge intelligence [41], [42] refers to the seamless fusion of AI and EC, where intelligent algorithms and models are deployed directly on edge devices or near data sources, bypassing the reliance on centralized cloud systems. By combining the computational power of EC with the advanced analytical and decision-making capabilities of AI, edge intelligence facilitates the creation of systems that are faster, more adaptive, and highly efficient. This approach goes beyond a mere technological advancement—it's a paradigm shift that redefines how intelligent systems operate in decentralized environments. It extends the boundaries of AI by enabling models and algorithms to function seamlessly on edge devices such as smartphones, IoT sensors, drones, wearables, and embedded systems.

2.3.1 Defining Characteristics

Edge intelligence applications differ fundamentally from traditional computing paradigms due to their ability to process data locally, respond in real time, and continuously adapt to dynamic environments. Unlike conventional applications that rely on centralized cloud processing, edge intelligence applications leverage decentralized AI models to provide low-latency, context-aware, and autonomous decision-making capabilities at the edge. The defining characteristics of these applications stem from their ability to personalize user experiences, optimize efficiency, predict outcomes, and interact naturally with their environments.

Personalization. Edge intelligence applications offer a high degree of personalization, by tailoring responses to user behavior, preferences, and real-time environmental conditions. Traditional applications often use generalized models that apply the same logic to all users, requiring constant cloud connectivity to retrieve personalized data. In contrast, on-device AI

models enable edge applications to adapt dynamically to individual users while maintaining privacy. For example, smartphones and wearable devices use edge intelligence to personalize voice assistants, fitness tracking recommendations, and smart home automation based on user habits. Similarly, edge-based recommendation systems in retail environments can offer real-time, location-aware suggestions without requiring constant server queries.

Context Awareness. Context awareness is another key differentiator, allowing applications to analyze surrounding conditions and modify behavior accordingly. Edge-powered smart city infrastructure can dynamically adjust traffic signals based on congestion patterns, while autonomous vehicles rely on sensor-driven edge AI to make split-second driving decisions. By understanding spatial, temporal, and user-specific contexts, edge intelligence applications can enhance efficiency and provide a more seamless user experience.

Predictive Capabilities and Proactive Decision-Making. Traditional applications react to user inputs and commands, but edge intelligence applications proactively anticipate needs and predict future events using real-time analytics and DL models. This predictive capability is particularly useful in industrial automation, healthcare, and logistics, where preventing failures or delays can be mission-critical. One of the most impactful use cases of predictive AI at the edge is predictive maintenance. In industrial settings, edge intelligence continuously monitors machinery using embedded sensors to detect anomalies and forecast potential failures. By identifying irregular patterns in temperature, vibration, or energy consumption, edge-based predictive maintenance systems can schedule proactive repairs, reduce downtime, and prevent catastrophic failures. Similarly, in healthcare, wearable medical devices use edge intelligence to monitor patient vitals in real time, detecting early warning signs of health conditions such as heart irregularities or oxygen level fluctuations. Instead of waiting for symptoms to worsen before seeking medical attention, AI-driven predictive diagnostics can alert both users and healthcare providers to take preventive action.

Automation and Decision Autonomy. A key advantage of edge intelligence applications is their ability to operate autonomously, without relying on external input or cloud connectivity. This level of automation is essential in environments where low-latency decision-making is required or where connectivity is unreliable. For example, autonomous drones and robotic systems rely on edge AI to navigate, recognize objects, and avoid obstacles in real time. If a drone had to constantly communicate with a cloud server before making a decision, latency and network disruptions could compromise safety and efficiency. Instead, on-device inference allows for immediate, independent decision-making, ensuring seamless operation even in remote locations. In smart homes, edge intelligence automates energy management systems, security monitoring, and appliance control, reducing the need for user intervention. For instance, smart thermostats adjust temperature settings based on occupancy and weather predictions, optimizing energy usage without requiring manual adjustments.

Natural Interaction and Human-Centric AI. Traditional applications often rely on predefined, rule-based interactions, requiring explicit user commands or structured inputs. In contrast, edge intelligence applications enable more natural, human-like interactions by incorporating speech recognition, gesture detection, and real-time contextual understanding. Voice assistants such as Google Assistant, Siri, and Alexa have evolved from simple command-based systems into adaptive, context-aware conversational agents. With on-device AI processing, these assistants can recognize and respond to voice commands locally, reducing response latency and enhancing privacy. Similarly, gesture-controlled interfaces in AR/VR

environments use edge intelligence to enable intuitive, real-time interactions without requiring cloud processing. Beyond voice and gesture recognition, emotion detection and sentiment analysis at the edge allow AI systems to respond empathetically to users, making interactions more personalized and emotionally aware. For example, AI-driven in-car assistants can monitor a driver's facial expressions and tone of voice to detect signs of fatigue or stress, adjusting music, lighting, or suggesting breaks accordingly.

Continuous Learning and Improvement. Traditional AI applications are typically trained in the cloud and deployed as static models, requiring periodic updates to improve performance. In contrast, edge intelligence applications can adapt dynamically through on-device learning and federated learning techniques. Federated learning allows edge devices to collaboratively improve AI models without transmitting sensitive data to the cloud. Instead of sending raw user data to a central server, devices train AI models locally and share only encrypted model updates, preserving privacy while continuously improving model accuracy. For instance, predictive text and autocorrect algorithms on smartphones utilize federated learning to refine suggestions based on individual typing patterns without exposing private conversations. Similarly, edge AI-powered surveillance systems can learn from local environmental conditions, improving anomaly detection without relying on large-scale cloud datasets. This ability to continuously learn and improve on the edge ensures that AI remains adaptable, efficient, and privacy-preserving, even in highly dynamic environments.

Sustainability, Efficiency, and Optimization. Edge intelligence applications are designed to optimize computing resources, minimize energy consumption, and enhance sustainability, making them ideal for green computing initiatives. Unlike traditional cloud-based applications that require constant data transmission and large-scale server farms, edge intelligence processes data locally, reducing network traffic, bandwidth usage, and overall energy demand. In smart grid management, edge intelligence is used to balance power distribution, optimize renewable energy usage, and prevent electrical outages by processing real-time data from IoT sensors deployed in power networks. Similarly, in agriculture, AI-powered edge devices monitor soil conditions, weather patterns, and crop health, allowing farmers to reduce water usage, optimize fertilizer application, and minimize waste. Beyond environmental impact, edge intelligence enhances computational efficiency by ensuring that only the most critical tasks are processed on-device, whereas non-urgent computations can be offloaded to edge servers or the cloud as needed. This hybrid approach improves responsiveness and energy efficiency, making edge intelligence an ideal solution for scalable, resource-aware AI deployment.

2.3.2 Types of Applications

Edge intelligence is broadly applied in two primary computing environments: mobile and IoT. While mobile edge intelligence focuses on high-performance AI inference on smartphones and tablets, IoT-based edge intelligence is designed for low-power, real-time analytics on embedded microcontrollers and sensors. These distinct categories define the landscape of edge intelligence applications and influence the hardware architectures, software optimizations, and deployment strategies required for efficient execution.

2.3.2.1 Mobile Edge Intelligence

Mobile devices such as smartphones, tablets, and wearables serve as powerful platforms for edge intelligence due to their high processing power, GPU acceleration, and robust connectivity. These devices can execute complex DL models for real-time AI applications, including:

- **Computer vision:** AI-powered image and video analysis, enabling applications such as face recognition, AR, and automated content enhancement.
- **Speech and natural language processing:** On-device execution of speech recognition, language translation, and voice assistants for privacy-preserving and low-latency interactions.
- **Health monitoring and personalization:** Wearable devices use edge intelligence for real-time health tracking, detecting heart irregularities, sleep patterns, and physical activity insights without requiring cloud connectivity.
- **Autonomous mobile applications:** AI-powered gesture recognition, predictive text input, and intelligent camera optimizations leverage DL for an enhanced user experience.

Given the computational power available in modern mobile devices, edge intelligence in this domain benefits from dedicated AI accelerators integrated within mobile System-on-Chips, such as Apple's Neural Engine, Qualcomm's Hexagon DSP, and Google's Edge TPU. These specialized components significantly boost AI performance while maintaining energy efficiency.

2.3.2.2 IoT Edge Intelligence

Unlike mobile devices, IoT-based edge intelligence operates in environments with severe power, memory, and connectivity constraints, making energy-efficient AI execution a primary concern. IoT edge intelligence is commonly deployed in:

- **Industrial automation:** AI-powered predictive maintenance, fault detection, and real-time quality control enhance efficiency in manufacturing, energy, and logistics.
- **Smart cities and infrastructure:** Edge-enabled traffic monitoring, air quality sensors, and energy management systems leverage AI for autonomous decision-making.
- **Healthcare and remote monitoring:** Wearable and implantable IoT devices analyze patient vitals in real time, enabling continuous health tracking and emergency alerts.
- **Security and surveillance:** AI-enhanced intrusion detection systems, anomaly detection in network traffic, and facial authentication reduce reliance on cloud-based analytics.

IoT-based edge intelligence typically relies on low-power microcontrollers and specialized AI chips designed for resource-efficient model execution. Unlike mobile SoCs, which offer higher computational throughput, IoT edge hardware is optimized for low-energy, event-driven AI tasks running on devices powered by batteries or energy-harvesting systems.

2.3.2 Device Hardware

The hardware landscape for edge intelligence is bifurcated between high-performance mobile System-on-Chips (SoCs) and ultra-low-power microcontrollers. Each category presents distinct trade-offs in computational power, energy efficiency, and AI acceleration capabilities.

2.3.2.1 Mobile SoCs for Edge AI

Modern mobile SoCs are highly integrated semiconductor solutions that consolidate multiple processing units, memory controllers, connectivity modules, and specialized accelerators into a single compact chip. This high level of integration allows mobile devices to efficiently perform a wide range of computational tasks, including AI-driven inference at the edge. Unlike traditional computing architectures, where separate components handle different functions,

mobile SoCs are designed for power efficiency, real-time responsiveness, and multi-functional capabilities within the constraints of mobile and embedded devices.

A typical mobile SoC consists of several key components, each optimized for specific workloads and tasks:

- **Processing units:** The core computational components include the Central Processing Unit (CPU) for general-purpose tasks, the Graphics Processing Unit (GPU) for parallelized computations, the Digital Signal Processor (DSP) for efficient signal processing, and the Neural Processing Unit (NPU) for dedicated DL acceleration.
- **Memory subsystem:** Includes LPDDR RAM controllers, cache hierarchies, and high-bandwidth memory interfaces, ensuring efficient data access and processing speeds.
- **Connectivity modules:** Integrated support for Wi-Fi, Bluetooth, 5G modems, and GNSS (Global Navigation Satellite System) to enable seamless wireless communication and real-time data transfer.
- **Power management and thermal control:** SoCs incorporate dynamic frequency and voltage scaling (DFVS), power gating, and advanced cooling mechanisms to balance performance and energy efficiency.
- **Security engines:** Hardware-based encryption, secure boot, and AI-accelerated authentication mechanisms protect sensitive data and prevent unauthorized access.

Processing units are fundamental to edge AI execution, each designed and optimized for specific tasks while remaining adaptable for AI model processing depending on workload requirements, power constraints, and optimization strategies. The CPU serves as the primary processor for general-purpose computing, whereas specialized units—the GPU, DSP, and NPU, collectively known as accelerators—enhance efficiency by boosting processing speed and reducing energy consumption during DNN inference. These accelerators can either offload intensive computations from the CPU or collaborate with it to distribute workloads, improving overall performance. As DNNs become increasingly central to mobile applications, the role of AI accelerators will continue to grow, making them indispensable for next-generation mobile AI and ensuring efficient on-device execution.

Central Processing Unit. The CPU is the most flexible and general-purpose processing unit in a mobile SoC. It is responsible for executing a wide range of tasks, including operating system functions, application logic, and lightweight AI inference workloads. CPUs in mobile devices are typically designed using ARM architectures, such as:

- **ARM Cortex-A series:** High-performance cores for demanding computations.
- **ARM Cortex-M series:** Low-power cores optimized for embedded and IoT applications.
- **ARM big.LITTLE architecture:** Combines high-performance and energy-efficient cores to balance speed and power consumption dynamically.

For AI inference, the CPU may not be the most efficient option. However, it is still used for (a) preprocessing tasks (*e.g.*, image normalization, feature extraction), (b) small-scale ML models that do not justify the use of dedicated accelerators, and (c) control logic and task orchestration in heterogeneous AI pipelines. While CPUs provide low-latency execution for simple AI tasks, they struggle with large-scale DL models, leading to the adoption of more specialized processors.

Graphics Processing Unit. Originally designed for rendering graphics, the GPU has emerged as a powerful engine for parallel computing, making it well-suited for DL inference. Unlike CPUs, which handle computations sequentially, GPUs execute thousands of parallel threads simultaneously, accelerating tasks such as matrix multiplications and convolutions—the core operations in neural networks. Mobile SoCs integrate GPUs optimized for low-power AI inference, such as:

- **ARM Mali GPUs:** Common in Android-based smartphones and tablets, offering hardware acceleration for AI workloads.
- **Qualcomm Adreno GPUs:** Found in Snapdragon SoCs, optimized for real-time graphics and AI acceleration.
- **Apple GPU:** Integrated into Apple's A-series and M-series chips, designed for high-performance AI applications.

Despite their computational power, GPUs consume more resources than other AI accelerators. Therefore, while they are useful for general AI workloads, they are not always the best choice for ultra-low-power edge applications.

Digital Signal Processor. The DSP is a specialized processor designed for high-speed mathematical operations, making it an excellent choice for AI applications involving real-time signal processing. DSPs are particularly useful in audio, speech recognition, and sensor-based AI tasks, where low-power, efficient computation is required. Mobile SoCs integrate dedicated DSPs such as:

- **Qualcomm Hexagon DSP:** Optimized for AI-powered voice recognition, sensor fusion, and computational photography.
- **Apple's A-series DSP:** Used for real-time audio enhancements and speech processing.
- **ARM Ethos-U DSP:** A low-power AI accelerator designed for embedded edge applications.

DSPs are particularly effective in (a) speech and NLP, with running AI models for voice assistants, real-time translation, and wake-word detection, (b) sensor fusion, by combining data from multiple sensors (*e.g.*, accelerometers, gyroscopes, microphones) to enable context-aware AI applications, and (c) energy-efficient AI inference, requiring minimal power consumption, making them ideal for always-on AI features. While DSPs offer high efficiency for specific AI workloads, they are not as versatile as GPUs for DL tasks. They are best suited for fixed-function AI tasks with predictable workloads.

Neural Processing Unit. The NPU is a dedicated AI accelerator designed specifically for neural network inference. NPUs are optimized for executing tensor-based computations with high efficiency and minimal energy consumption, making them the preferred choice for on-device AI processing in modern mobile SoCs. Key NPUs in mobile SoCs include:

- **Apple Neural Engine:** Optimized for on-device DL inference in iPhones and iPads.
- **Huawei Ascend NPU:** Found in Kirin SoCs, designed for real-time AI acceleration.
- **Google Edge TPU:** Specializes in low-power AI inference, particularly for TensorFlow Lite models.
- **MediaTek APU (AI Processing Unit):** Integrated into Dimensity SoCs, enhancing camera AI and speech recognition.

NPUs offer significant advantages, including (a) high efficiency for AI workloads, as they are optimized for low-power, high-throughput AI inference, (b) accelerated neural network execution, designed for various DL architectures, and (c) optimized power consumption, enabling always-on AI features without significantly impacting battery life. Since NPUs are purpose-built for AI, they outperform CPUs, GPUs, and DSPs for most DL inference tasks. As edge AI continues to evolve, NPUs are expected to play an increasingly dominant role in on-device intelligence.

2.3.2.2 Microcontrollers and Specialized IoT AI Hardware

IoT edge intelligence relies on a mix of microcontrollers, AI-enabled edge processors, and low-power accelerators, which are designed to handle lightweight AI inference tasks with minimal energy consumption. These devices power applications in industrial automation, smart cities, healthcare monitoring, and autonomous IoT networks, where continuous operation, ultra-low latency, and on-device intelligence are crucial.

Since IoT devices often function without direct human intervention and are deployed in remote or embedded environments, their computational models differ significantly from those of mobile or cloud-based AI systems. The focus is on executing efficient AI models with minimal hardware overhead, balancing processing capability and power consumption while maintaining real-time responsiveness. IoT edge computing devices are designed with specific constraints that differentiate them from mobile SoCs:

- **Ultra-low power consumption:** Power efficiency is a key requirement for IoT-based edge intelligence, as many devices run on batteries or energy-harvesting methods like solar or kinetic energy. Unlike mobile SoCs, which benefit from rechargeable batteries, IoT devices must maximize energy efficiency for prolonged operation. To achieve this, they use duty cycling, where processors enter low-power sleep modes when idle, and DVFS, which adjusts power consumption based on real-time workload demands.
- **Lightweight AI execution:** Due to limited computational resources, IoT devices cannot support large-scale DL models and instead rely on TinyML techniques to perform inference on low-power microcontrollers. Optimizations such as quantization, pruning, and knowledge distillation reduce model size and computational demands while preserving accuracy. Additionally, hardware-aware optimizations improve efficiency by tailoring model architectures to specialized edge AI processors.
- **Embedded real-time processing**: Many IoT devices operate in mission-critical environments where real-time decision-making is essential. Smart surveillance cameras, industrial sensors, and health monitoring devices must analyze data instantly without cloud dependency to ensure privacy, safety, and rapid response times. To support this, IoT AI hardware integrates real-time operating systems for precise task execution and low-latency AI pipelines for immediate data processing. Sensor fusion further enhances decision-making by combining inputs from multiple sensors, such as accelerometers and gyroscopes, for more context-aware predictions.

Platforms. Several specialized processors and microcontrollers have been developed to support AI inference at the edge, balancing energy efficiency, processing power, and deployment cost. Among the most widely used architectures is the ARM Cortex-M series, which powers a range of low-power AI applications, from wearables to industrial sensors. The Cortex-M4 and Cortex-M7 models feature DSP extensions that enhance AI workloads, whereas the Cortex-M55 integrates an Ethos-U NPU for more efficient DL inference. These architectures allow IoT devices to execute lightweight models while maintaining low energy consumption.

For more demanding edge applications, AI accelerators such as the NVIDIA Jetson Nano provide a compact yet powerful platform for real-time AI computing. Designed for robotics, vision processing, and industrial automation, the Jetson Nano integrates GPU acceleration, enabling it to handle complex DL tasks such as object recognition and autonomous navigation. Unlike traditional microcontrollers, which are constrained by limited processing power, the Jetson Nano allows AI-driven robotics and edge-based computer vision applications to run efficiently without cloud dependency.

Another prominent example is the Google Coral Edge TPU, a low-power AI accelerator optimized for TensorFlow Lite models. Specifically designed for edge inference, the Coral Edge TPU delivers ultra-low latency execution for AI tasks such as facial recognition, security surveillance, and industrial quality control. Its energy-efficient design makes it suitable for deployment in smart cameras, edge gateways, and home automation systems, where real-time AI processing is essential.

Lastly, for cost-effective IoT applications, the ESP32 microcontroller has gained popularity due to its AI acceleration capabilities and integrated wireless connectivity. The ESP32 is widely used in smart home devices and automation systems, supporting applications such as voice recognition, gesture control, and real-time environmental monitoring. By combining edge AI capabilities with built-in Wi-Fi and Bluetooth connectivity, the ESP32 enables intelligent IoT solutions that operate autonomously while maintaining seamless communication with other networked devices.

2.4 On-Device Inference

Many organizations and publications define edge intelligence as the paradigm of executing AI algorithms locally on end devices using data generated directly on those devices. However, due to various challenges—most notably resource constraints—a device may not always be capable of executing a DNN locally. Consequently, edge intelligence should be redefined as a paradigm that fully leverages the available data and computational resources across the hierarchy of end devices, edge nodes, and cloud data centers to optimize the overall performance of DNN execution [43].

It is important to note that "executing a DNN" can refer to either training or inference. Training and inference are fundamentally different processes in the lifecycle of a DNN, each with distinct resource requirements and computational complexities.

- **Training** is significantly more resource-intensive than inference due to the nature of its operations and objectives. During training, the DNN learns patterns and relationships in the data by iteratively adjusting its parameters using optimization algorithms like stochastic gradient descent. This process involves forward propagation, where the input data pass through the network, and backward propagation, where gradients are computed and propagated back to update the parameters. These operations require extensive matrix multiplications, gradient calculations, and weight updates, all of which demand high computational power, memory bandwidth, and energy. Additionally, training typically involves large datasets and requires multiple epochs, further amplifying the computational burden.
- **Inference**, by contrast, focuses on using a pre-trained model to make predictions or classifications on new input data. It involves only forward propagation through the network, which is computationally less demanding than the iterative processes of training. Inference is often latency-sensitive, particularly for real-time applications, but its resource requirements are generally more manageable.

This distinction is crucial in the context of edge intelligence, as training is rarely performed on edge devices due to their limited computational resources. Instead, training is typically conducted on powerful cloud data centers, whereas inference is executed locally on edge devices to leverage proximity to data and achieve low-latency performance. By balancing these processes across the edge-cloud hierarchy, edge intelligence can optimize the deployment and execution of DNNs while addressing the constraints of resource-limited environments.

This dissertation focuses on on-device, or local, inference—the execution of pre-trained DL models directly on edge devices such as smartphones, IoT devices, or embedded systems. This approach enables real-time decision-making and processing without relying on continuous cloud connectivity. By performing inference locally, edge devices can deliver intelligent functionalities, such as object detection, voice recognition, and predictive maintenance, while addressing key challenges like latency, privacy, and bandwidth limitations.

2.4.1 Benefits

The ability to run AI models directly on edge resources provides several advantages over traditional cloud-based inference, particularly in terms of latency, privacy, energy efficiency, and continuous operation.

Low Latency and Real-Time Decision-Making. One of the most significant advantages of on-device inference is the ability to process data with ultra-low latency. Since all computations occur locally, there is no need to transmit data to a remote cloud server, eliminating network delays. This is particularly important for applications that require real-time decision-making. For instance, in self-driving cars, object detection and path planning must be executed within milliseconds to ensure safe navigation, making cloud-based inference impractical due to latency constraints. Similarly, in AR applications, interactive elements must be rendered instantaneously to provide a seamless user experience.

Enhanced Privacy and Data Security. By keeping all computations local, on-device inference significantly reduces the risk of data breaches and unauthorized access. Unlike cloud-based AI, where sensitive user information is transmitted and stored on external servers, on-device processing ensures that personal data remain confined to the device itself. This is crucial in applications such as facial recognition, fingerprint authentication, and health tracking, where maintaining user privacy is a top priority. Industries such as healthcare and finance benefit from this approach, as regulations often require data protection and compliance with stringent security protocols.

Energy Efficiency and Reduced Bandwidth Usage. On-device inference also plays a vital role in optimizing energy consumption, particularly in battery-powered devices such as smartphones, wearables, and IoT sensors. Cloud-based inference often involves frequent communication with remote servers, which not only increases network bandwidth usage but also drains battery life due to continuous data transmission. By running compact AI models locally, devices can reduce power consumption, extending their operational lifespan. This is particularly beneficial for IoT applications deployed in remote environments, where energy efficiency is essential for long-term autonomous operation.

Offline Functionality and Reliability. Another major advantage of on-device inference is its ability to function without internet connectivity. Many AI-powered services depend on cloud access to process requests, making them unreliable in areas with poor network coverage or intermittent connectivity. On-device AI allows applications to run continuously, even in offline environments, ensuring robustness and reliability. This is critical in industrial automation,

disaster response, and mission-critical healthcare systems, where AI must function autonomously without depending on external infrastructure.

Scalability and Cost Reduction. From a deployment perspective, on-device inference reduces the need for centralized cloud computing resources, lowering operational costs. Cloud-based inference can become computationally expensive as the number of connected devices grows, requiring large-scale infrastructure to handle the increasing workload. By distributing inference across edge devices, organizations can reduce dependency on expensive cloud servers, making AI more scalable for large-scale deployments such as smart cities, enterprise AI, and industrial IoT applications.

2.4.2 Challenges

While on-device inference offers numerous advantages, it also presents a range of challenges that must be addressed to ensure efficient, high-performance AI execution on resource-constrained devices. These challenges extend beyond the limitations of resource constraints and include factors such as model complexity, scalability, and environmental adaptability [44]. Addressing these challenges is vital to unlocking the full potential of on-device inference, enabling its seamless integration into a wide array of applications and environments.

Limited Computational Power and Memory Constraints. One of the fundamental challenges is the limited computational power and memory constraints of edge devices. Unlike cloud servers, which leverage high-performance GPUs and TPUs, mobile and embedded systems must execute AI models on low-power processors. Running large-scale DL models, such as large language models (LLMs), on these resource-constrained devices is particularly difficult due to high memory requirements, computational intensity, and energy consumption. Many modern DL models involve billions or even trillions of parameters, requiring considerable storage and RAM, which are often unavailable on mobile and IoT devices.

Hardware Heterogeneity and Cross-Platform Optimization. Beyond software and hardware constraints, cross-platform optimization and hardware heterogeneity introduce additional barriers to widespread on-device AI deployment [45], [46], [47]. Unlike cloud-based inference, where AI models run on standardized infrastructure, on-device inference must be optimized for a wide range of mobile SoCs, embedded processors, and AI accelerators. Each device has different processing architectures, memory hierarchies, and power constraints, requiring customized implementations to achieve optimal performance. AI models that perform well on one hardware platform may not generalize effectively across other devices, necessitating extensive hardware-aware tuning.

Energy Consumption and Dynamic Resource Availability. Another major challenge is energy consumption, particularly for battery-powered devices. AI inference is computationally intensive, and continuously running DL models can rapidly deplete battery life. Unlike cloud-based systems that have access to high-efficiency cooling and power management, mobile and IoT devices must operate within strict power budgets to avoid excessive heat generation and battery drain. This challenge is compounded by the dynamic environment of on-device inference, where resource availability fluctuates due to factors such as processor workload, temperature conditions, and battery level. For example, when a device's processor is under heavy load or operates in high-temperature conditions, thermal throttling may reduce processing speed [48], significantly affecting inference performance. Similarly, power-saving mechanisms may lower CPU and GPU clock speeds when battery levels are low, further constraining available computational power.

Diversity of Deep Neural Networks and Rapid Evolution. Beyond hardware constraints, the diversity of DNNs poses a significant challenge. The rapid advancement of AI research has led to a growing variety of DL models, each with different architectural designs, computational demands, and optimization trade-offs. Some models are designed for high accuracy, requiring substantial computational resources, making them unsuitable for resource-limited edge devices. Others prioritize speed and efficiency but may sacrifice accuracy or generalization capabilities. Additionally, newer, more complex AI models are continuously being developed, each introducing new challenges in resource utilization, hardware compatibility, and real-time performance. Supporting this evolving landscape requires continuous innovation in both hardware accelerators and software optimization techniques to ensure that on-device inference remains feasible for future AI models.

Conflicting Service-Level Objectives. Adding to the complexity, on-device inference must meet strict and diverse application service-level objectives (SLOs), which further complicates optimization. Different AI applications impose unique performance requirements, such as high accuracy, low latency, minimal memory usage, or energy efficiency. These requirements often conflict with each other, making it difficult to develop a single AI model that satisfies all constraints simultaneously. For instance, autonomous vehicles and medical diagnostics demand high-precision models, but these models tend to be computationally intensive and may not meet real-time processing requirements. Conversely, applications such as voice assistants and AR prioritize low-latency responses, which may necessitate smaller, more efficient models that compromise accuracy. Balancing these competing objectives while maintaining model reliability on resource-constrained devices requires sophisticated optimization techniques, where trade-offs between accuracy, speed, and memory footprint must be carefully managed.

Multi-DNN Execution and Resource Allocation. Another significant challenge is the requirement for some applications to execute multiple DNNs simultaneously, often referred to as multi-DNN configurations [49]. Many modern applications involve concurrent AI tasks, each requiring a distinct DL model [50], [51]. For example, a smart camera system may simultaneously run models for object detection, facial recognition, and activity recognition, each requiring real-time execution. Managing multiple DNNs on a single device introduces various issues, including increased computational load, higher memory demands, and potential conflicts in resource allocation. To address these challenges, advanced scheduling algorithms and resource management techniques are required to optimize the execution of multiple DNNs in parallel. Ensuring that each model receives adequate processing time while maintaining responsiveness is a critical challenge in real-time AI applications.

Continuous Learning and Model Updating. Finally, the need for continuous learning and model updates adds another layer of complexity to on-device inference. Unlike cloud-based AI, where models can be updated centrally, on-device AI requires local mechanisms for adapting to new data and improving performance over time. Federated learning has emerged as a promising approach, allowing devices to collaboratively train AI models without transmitting raw user data to the cloud, preserving privacy while enabling incremental learning. However, deploying federated learning at scale presents new challenges, such as ensuring secure aggregation of model updates, managing device synchronization, and optimizing local training for low-power hardware. Addressing these issues is essential for enabling adaptive, self-improving AI models that can operate efficiently in edge environments.

2.5 Efficient Deep Learning

Efficient deep learning (EDL) is a crucial enabler of on-device inference, laying the foundation for deploying sophisticated AI systems in resource-constrained environments. EDL encompasses a wide range of methods, techniques, and optimizations aimed at facilitating or accelerating the training and inference of DNNs [52]. These methods aim to minimize the computational, memory, and energy demands of DNNs while preserving key performance metrics such as accuracy and robustness.

As edge intelligence continues to grow in importance, EDL will remain a driving force behind its evolution. By enabling high-performance DL in constrained environments, EDL not only powers the next generation of edge intelligence but also democratizes access to AI technologies, bringing smart, personalized, and context-aware solutions to the forefront of modern computing. This synergy between EDL and edge intelligence is driving innovation across industries, from healthcare and manufacturing to smart cities and autonomous vehicles, making AI an integral part of everyday life.

2.5.1 Metrics of Interest

The challenge of efficient DNN inference is inherently complex and multifaceted, as it involves balancing multiple critical metrics alongside accuracy during the development and execution of the model. While accuracy remains a primary goal, ensuring that a model performs well on its designated task, other equally important metrics must also be carefully considered, particularly in resource-constrained environments like edge devices:

- **Latency**, which measures the time taken for the model to process a single input and generate an output, is typically expressed in milliseconds (ms) or seconds. It is a crucial factor, especially for real-time applications such as autonomous systems, augmented reality, and interactive voice assistants. High latency can degrade user experience or even compromise system functionality, making low-latency inference a fundamental requirement for many edge applications.
- **Computational complexity** is closely tied to latency and refers to the number of mathematical operations required to execute a DNN, commonly measured in floating-point operations (FLOPs). Models with high computational complexity require more processing power and time, which can limit their deployability on resource-limited devices. Reducing computational complexity without sacrificing accuracy is a major focus in the development of efficient DNNs.
- **Throughput**, or the number of inferences a model can perform per unit of time, is particularly relevant in scenarios where multiple inputs must be processed simultaneously or in quick succession, such as in video analytics or data streams. It is typically expressed in inferences per second (IPS) or samples per second (SPS). High throughput ensures that systems can handle large workloads efficiently, making it a crucial metric for scalability.
- **Model size** pertains to the storage space required to save the model parameters and is usually measured in megabytes (MB) or gigabytes (GB). Edge devices often have limited storage capacity, so reducing the size of the model without sacrificing accuracy is critical for enabling deployment on such devices.
- **Memory footprint** represents the amount of RAM required to load and execute a model, including both its parameters and intermediate computations. It is measured in megabytes (MB) or gigabytes (GB). Many edge devices operate with strict memory

constraints, and ensuring that the model can fit within the available memory is essential to prevent bottlenecks or execution failures. Optimizing the memory usage of both the model's weights and intermediate activations is a core aspect of efficient DNN design.

- **Energy consumption**, a critical metric for battery-powered devices like smartphones, drones, and IoT sensors, is often expressed in joules (J) per inference or watts (W) for continuous operation. Energy-efficient inference not only extends battery life but also reduces operational costs and environmental impact. Achieving energy-efficient inference involves innovations in algorithm design and hardware utilization.
- **Accelerator compatibility**, or accelerator friendliness, is less directly quantifiable but is typically evaluated based on the model's ability to leverage hardware accelerators such as GPUs, TPUs, or specialized AI chips effectively. Metrics such as hardware utilization rate (%) or the number of operations that can be parallelized on an accelerator help assess compatibility. Models designed with hardware accelerators in mind can achieve significant performance gains in terms of latency, throughput, and energy efficiency. This involves tailoring model architectures to align with the computational patterns favored by these devices, such as leveraging parallelism and avoiding operations that are inefficient on the target hardware.

The interplay among these metrics often involves trade-offs. For example, reducing model size might impact accuracy, or designing a model to fully utilize specialized hardware may lead to lower latency but could increase energy consumption if the hardware is not optimized for power efficiency. Balancing these competing objectives requires a holistic, multidisciplinary approach that considers the specific requirements of the application and the constraints of the target device. Efficient DNN inference, therefore, is not solely about creating high-performing models but also about ensuring their deployability and scalability across diverse environments and devices.

2.5.2 Compression Methods

Among all the challenges associated with on-device inference, the constraint of limited computational resources, memory, and energy efficiency has been the most extensively studied. The disparity between the computational demands of DL models and the hardware capabilities of mobile and embedded devices has driven significant research into model compression and optimization techniques. Since DNNs often contain millions, billions, or even trillions of parameters, their execution on resource-constrained devices requires specialized methods to reduce model size, decrease inference latency, and minimize power consumption while maintaining acceptable levels of accuracy.

To address these limitations, various compression techniques have been developed, aiming to make DL models more efficient without compromising their predictive performance. These techniques include quantization, pruning, knowledge distillation, and neural architecture search (NAS), each targeting a different aspect of model efficiency—from reducing numerical precision to identifying the most critical network components and restructuring the model for optimal execution on edge hardware. The widespread adoption of compression techniques has been instrumental in enabling the deployment of DL on mobile devices, IoT sensors, and embedded systems, bridging the gap between state-of-the-art AI models and the realities of constrained edge computing environments. The following section provides a detailed description of quantization, as it is the primary method employed throughout this dissertation.

2.5.2.1 Quantization

Quantization [53], [54] is one of the most widely used techniques for compressing DNNs. It involves representing the weights, activations, and/or mathematical operations of a neural network using lower-bit precision compared to the standard 32-bit floating-point format commonly used in training. The primary observation behind quantization is that DNNs, especially modern architectures, tend to have a large number of parameters, many of which contribute marginally to the overall performance. This redundancy makes it possible to reduce the numerical precision of weights and activations without significantly impacting the network's predictive capabilities. The patterns extracted by neural networks are typically resilient to small numerical errors introduced by lower-precision representations.

Different quantization approaches offer varying levels of benefits. Basic quantization schemes, where only the weights are quantized, primarily reduce the model's size. This reduction leads to decreased storage requirements and lower memory consumption, making the model more lightweight and efficient for deployment. More advanced quantization techniques go beyond the weights by also quantizing the activations and performing computations at reduced precision. These methods can significantly enhance the execution speed of DNNs. Despite its advantages, quantization often results in a trade-off between efficiency and accuracy. While the reduction in precision can lead to a decrease in model accuracy, in most cases, this accuracy loss is negligible compared to the substantial gains in computational efficiency and resource optimization. As a result, quantization has become a critical tool for enabling the deployment of DL models in environments where computational power and memory are limited, paving the way for efficient on-device AI applications.

Table 2.1 provides an overview of four widely used quantization schemes: half-precision floating-point (FP16), 8-bit dynamic range (DR8), 8-bit fixed-point with floating-point fallback (FX8), and full 8-bit fixed-point (FFX8). These schemes are compared against the original 32-bit floating-point (FP32) representation, highlighting differences in model size reduction and the numerical data types used for inputs, outputs, weights, and activations. The data type assigned to the weights directly determines the storage requirements of the model, which accounts for the observed compression ratios.

Table 2.1 Quantization Schemes

Scheme	Compression	Inputs & Outputs	Weights	Activations
FP32	-	fp32/int32/int64	fp32	fp32
FP16	2×	fp32/int32/int64	fp16	fp16(/fp32 fallback)
DR8	4×	fp32/int32/int64	int8	fp32/int8
FX8	4×	fp32/int32/int64	int8	int8(/fp32 fallback)
FFX8	4×	int8/int32	int8	int8

The operational principles of these quantization schemes are explained in detail below, providing insight into how each method processes model components and leverages hardware capabilities to achieve efficiency:

- **FP16**: This scheme uses 16-bit floating-point computations by default, but it includes a fallback mechanism to 32-bit floating-point (fp32) arithmetic when the hardware lacks native support for 16-bit operations. In such cases, weights are dequantized to fp32 before the first inference, and activations are also stored in fp32 format. FP16 is widely supported by mobile GPUs, which makes it a suitable choice for scenarios

where reduced precision can be leveraged without significantly compromising accuracy or performance.

- **DR8**: In this scheme, weights are quantized to 8 bits, whereas activations remain in fp32. However, some activations may undergo dynamic quantization during inference, using quantized kernels for faster execution. DR8 leverages fixed-point arithmetic, when possible, which can reduce computation times compared to floating-point arithmetic. The performance gains depend on the characteristics of the model and the hardware's ability to optimize for mixed-precision operations.
- **FX8**: Similar to FP16, FX8 uses integer kernels as the default execution mode, where weights and activations are quantized to 8 bits. When integer operations are not supported on the hardware, the scheme allows for a fallback to 32-bit floating-point operators. This hybrid approach provides flexibility for deployment across devices with varying levels of integer computation support, making it suitable for platforms with limited fixed-point optimization.
- **FFX8**: This scheme enforces full integer quantization for all model components, including weights, activations, operations, inputs, and outputs. FFX8 relies exclusively on integer arithmetic, ensuring compatibility with integer-only devices such as microcontrollers, DSPs, and NPUs. This makes FFX8 ideal for resource-constrained hardware platforms, where efficient execution and minimal memory usage are critical.

It is evident that each quantization scheme is designed with particular hardware capabilities and performance trade-offs in mind. As such, selecting the appropriate quantization scheme requires a thorough alignment with the characteristics and constraints of the target hardware platform. This includes factors such as the supported data formats, the type and efficiency of arithmetic units and instructions, memory access patterns, and available bandwidth. Additionally, the specific requirements of the application must be taken into account, such as the desired trade-off between computational efficiency, model size, and accuracy. These considerations ensure that the chosen quantization scheme optimizes performance while meeting the operational needs of both the hardware and the application. Therefore, the decision is not solely based on reducing model size or speeding up inference, but also on ensuring that the chosen scheme is compatible with the hardware's architecture and meets the functional requirements of the deployment environment.

3 Optimized On-Device Inference for Mobile Devices

In Section 2.4, the importance of on-device inference and its associated challenges were highlighted, emphasizing that addressing these challenges is crucial to unlocking the full potential of on-device deep learning. Successfully overcoming these obstacles will enable the seamless integration of advanced AI capabilities into our interconnected devices, creating a transformative impact across a range of applications. These challenges are expected to intensify as emerging DL architectures, such as Transformers, become the dominant models for various tasks. While current efforts offer targeted solutions to individual challenges, there remains a critical need for a unified framework capable of addressing these challenges collectively.

In this chapter, `CARIn` [11] is introduced, a novel framework for optimizing the deployment of DL applications on mobile devices, designed to meet the demands of modern and evolving workloads. Building upon the initial work, `OODIn` [55], which introduced a highly parameterized software architecture for optimizing single-DNN applications (focusing primarily on image classification), its capabilities have been significantly extended. By leveraging `OODIn`'s modular architecture, which facilitates efficient modification of model and system parameters, two groundbreaking components are proposed to address the complexities of model multi-tenancy and runtime adaptability:

- **First, an expressive multi-objective optimization (MOO) framework** is developed, enabling the precise modeling of performance requirements and constraints for both single- and multi-DNN workloads. This framework captures the diverse needs of DL applications, considering metrics such as latency, energy consumption, and memory usage.
- **Second, a runtime-aware sorting and search (RASS) algorithm** is introduced—an innovative MOO solver designed for rapid, low-overhead adaptation to dynamic resource conditions. Unlike traditional optimizers that produce a static execution plan for a specific device [56], [57], RASS generates a portfolio of configurations, accommodating variations in resource availability without the need for continuous problem re-solving. This approach ensures sustained high performance while minimizing runtime disruptions.

Furthermore, the scope of `CARIn` is expanded by targeting a broader set of tasks and model architectures, including CNNs and Transformers. The comprehensive evaluation spans a diverse set of realistic scenarios, characterized by varying performance demands, demonstrating `CARIn`'s adaptability and effectiveness in real-world conditions. This chapter lays the groundwork for a unified system capable of addressing the multifaceted challenges of on-device inference while advancing the state of efficient deep learning.

3.1 Related Work

EDL solutions for on-device inference on mobile devices are heavily dependent on the specific challenges (see Subsection 2.4.2) that need to be addressed in each case. Each challenge, whether related to resource constraints, device heterogeneity, dynamic environments, or the diversity of DNN models, requires tailored approaches that optimize performance in a way that

aligns with the unique limitations and requirements of the device and application. These solutions can broadly be categorized into model-level optimizations, system-level optimizations, and joint model-system optimizations, each addressing different aspects of these challenges [58].

Model-level optimizations involve modifying or redesigning the neural network to enhance its efficiency. These methods directly alter the architecture, layers or learned parameters of the model, enabling it to adapt to resource-constrained environments. Typically device-agnostic, model-level optimizations focus on making the model intrinsically efficient, independent of the specific characteristics of the underlying hardware. System-level optimizations, on the other hand, are focused on improving the interaction between the model and the hardware on which it operates. These optimizations leverage hardware accelerators, such as GPUs or NPUs, and exploit compiler and runtime framework enhancements that tailor the model's execution to the specific characteristics of the target device. Joint model-system optimizations go a step further by integrating both model and system design to achieve better efficiency and performance than either could provide independently. These optimizations recognize the interdependence between the model and the system, allowing for co-adaptation.

3.1.1 Limited Resources

Edge devices, including smartphones, IoT sensors, and embedded systems, typically function within stringent resource constraints, such as limited computational power, memory, storage, and energy. These constraints pose considerable challenges for deploying DNNs in such environments. Various solutions have been proposed in the literature to address this challenge, with the most commonly employed approaches focusing on model-level optimizations.

3.1.1.1 Model Compression Techniques

One of the most common strategies to address resource constraints is to reduce the size and complexity of DNNs without significantly sacrificing accuracy. Techniques include:

- **Quantization**: As discussed in Subsection 2.5.2.1, reducing the precision of model parameters and operations (*e.g.*, from 32-bit floating-point numbers to 8-bit integers) can effectively reduce both memory usage and computational requirements.
- **Pruning [59]**: Removing redundant or less important parameters and connections in the network, often resulting in sparse models. Magnitude-based pruning and structured pruning are widely used approaches.
- **Knowledge distillation [60]**: Transferring knowledge from a large, complex model (teacher) to a smaller, simpler model (student), which is then deployed on resource-constrained devices.

3.1.1.2 Efficient Neural Network Architectures

Research has led to the development of lightweight and efficient neural network architectures specifically designed for the edge. These models are tailored to balance accuracy and computational efficiency:

- **MobileNets [61], [62]**: Employ depthwise separable convolutions to reduce the number of operations and parameters.
- **EfficientNets [26], [63]**: Uses a compound scaling method to optimize the trade-off between model depth, width, and resolution.

- **Transformers for edge devices**: Emerging approaches such as TinyBERT [64] and DistilBERT [65] reduce the number of layers or hidden units in Transformer models to fit edge constraints.

3.1.2 Device Heterogeneity

The majority of endeavors aimed at addressing device heterogeneity predominantly concentrate on the model level, *i.e.*, by identifying the most fitting DL architecture tailored to a specific hardware platform. Among the prominent model-level methodologies, neural architecture search (NAS) and model scaling have a central role:

- **Hardware-aware NAS**: HW-NAS approaches seek to optimize DNN architectures both for high predictive accuracy and for efficient execution on a target deployment platform. Its most prominent premise is the inclusion of (a) hardware constraints, and (b) latency, energy and other system metrics, as objectives during the search process [66], [67], [68], [69]. HW-NAS usually involves performance prediction in order to guide the search algorithm. Nonetheless, estimating precise latency, memory or energy figures can be challenging, and the method's effectiveness heavily relies on the accuracy of these estimates. Such approaches can also be computationally intensive due to the need to train and evaluate a large number of candidate architectures.
- **Supernet-based NAS**: Also known as one-shot NAS, it is an approach that leverages a supernet along with weight sharing to facilitate efficient architecture search [70], [71], [72], [73]. A supernet is a network containing all possible architectural choices of a given search space and it enables the exploration of diverse neural architectures while significantly reducing computational overhead. While this approach reduces the training-time computational requirements, it may not be as effective at tailoring architectures to specific hardware constraints and weight sharing may restrict fine-grained control over architectural decisions.
- **Model scaling**: It involves adjusting parameters such as the depth, width and input size of a DNN to strike a balance between accuracy and efficiency [63], [74], [75]. This technique is often applied along with NAS or knowledge distillation methods to also accommodate resource constraints. However, model scaling might not fully exploit the unique hardware characteristics of specific devices, potentially leading to sub-optimal performance.

3.1.3 Dynamic Environment

The dynamic nature of edge devices necessitates the capability for runtime adaptation, a challenge that has been explored through research at both the model and system levels. At the model level, the focus is primarily on developing techniques that enable the dynamic adjustment of a model's architecture in response to variations in resource availability. These adaptive models are designed to modify their architecture and parameters in real time during inference, allowing them to effectively adapt to the changing constraints of the computational environment. Notable examples of such models include adaptive supernets [76], [77], adaptive model scaling [78], [79], multi-branch networks [80], early-exit models [67], [81], and various other innovative approaches. However, designing adaptive mechanisms that function seamlessly across a wide array of devices presents substantial technical challenges. Additionally, the adaptability of these networks may introduce computational overhead, which could potentially affect the performance of real-time applications. At the system level, complementary methods such as dynamic compression [82], adaptive model selection [83], and

efficient scheduling across available hardware resources [84] are employed to further optimize performance and resource utilization.

3.1.4 DNN Diversity

The challenge posed by the diversity of available models emphasizes the necessity for robust model selection systems capable of dynamically identifying the most suitable model for a given scenario. These systems must consider the unique resource constraints of edge devices, as well as the specific performance goals of the application. Such systems should be able to evaluate the computational complexity, memory usage, and energy consumption of models, selecting the one best aligned with real-time conditions. Furthermore, dynamic model selection should be adaptable to factors such as workload fluctuations and environmental changes, enabling execution strategies that align with the device's current capabilities.

3.1.5 DNN Innovations

As new, more sophisticated models are introduced, traditional hardware solutions often find it challenging to keep up with the increasing computational and resource demands. This gap between hardware capabilities and model complexity underscores the need for innovative hardware and software solutions that can support the evolving landscape of AI:

- **Adaptive hardware solutions**: Field Programmable Gate Arrays (FPGAs) have emerged as a critical enabler in the search for future neural processing unit (NPU) designs. Due to their reconfigurability, FPGAs allow for the exploration and evaluation of a wide array of potential hardware designs tailored to specific DNNs. By simulating these designs on FPGA platforms, it becomes possible to assess crucial performance metrics such as processing speed, power consumption, and hardware area. The goal of this process is to identify the Pareto-optimal accelerator design, which strikes a balance across these factors, for a range of representative DNN models. Once the highest-performing design is identified, it can be translated into an Application-Specific Integrated Circuit (ASIC) for mass production. This ASIC can then be integrated into future consumer devices as a specialized NPU, optimizing the execution of DNNs on edge devices.
- **Dynamic software and system-level optimizations**: Alongside hardware innovations, dynamic software solutions are indispensable in addressing the challenge posed by the continuous advancement of DNNs. These solutions enable the adaptation of models to ever-changing conditions by providing software frameworks capable of optimizing model execution based on real-time resource availability. Advanced system-level optimizations, such as compiler-based transformations, dynamic model selection, and real-time model scaling, play a critical role in ensuring that DNN models are efficiently deployed on edge devices.

3.1.6 Diverse Application SLOs

Most prior research focused on achieving SLOs has primarily been centered on system-level developments for edge servers [85], [86], [87], [88], where multiple DNN models can be executed in parallel. A relatively small number of studies, however, have examined the specific challenges of on-device execution, particularly in the context of managing multiple inference requests across heterogeneous processors [89], [90]. These works often focus on optimizing one or a few performance metrics but overlook the broader spectrum of SLOs that are critical for real-world mobile applications.

3.1.7 Multi-DNN Inference

To enable efficient multi-DNN inference, researchers have investigated solutions across both model and system levels [49]. From the model perspective, the execution of multiple DNNs often aligns with the principles of multi-task learning [91], [92], a technique where a single model is trained to perform multiple related tasks concurrently. By utilizing a multi-task model for inference, the need for running multiple independent models simultaneously can be mitigated, offering potential benefits in terms of resource utilization and efficiency. This approach simplifies the execution process and can reduce the overhead associated with maintaining separate models for each task.

At the system level, most research efforts focus on optimizing the use of heterogeneous processors available on edge devices, seeking to identify the most effective mapping strategy for distributing inference tasks. These strategies typically involve partitioning the model at the layer level [50], [93], where each segment of the model is executed on the most suitable hardware unit, or implementing task-level priorities [94] to ensure that critical tasks are given higher precedence in terms of computational resources. This allows for more efficient resource management, ensuring that all available hardware is utilized optimally to meet the performance objectives.

In addition to these approaches, there is a growing body of research dedicated to the development of multi-tenant inference systems [95], [96]. While much of the existing work has focused on server-based configurations [97], [98], where multiple inference tasks are handled simultaneously by shared resources, there is an increasing need to adapt these techniques for on-device applications [49]. On-device multi-tenant systems face additional challenges, such as real-time constraints, limited resources, and the need for efficient orchestration of multiple inference requests across diverse hardware components. Consequently, on-device multi-DNN solutions require novel techniques for managing the interactions between models and optimizing hardware utilization while maintaining application-specific performance goals.

3.2 System Overview

`CARIn` addresses the main challenges of on-device DL inference in two ways. First, a novel approach to modeling DL applications is introduced, utilizing a multi-objective optimization (MOO) framework to encapsulate their characteristics. Given the rising number and diversity of DL applications, `CARIn` is able to analytically represent their various performance requirements and constraints, with the required expressivity to support both single- and multi-DNN scenarios. Second, in order to enable runtime adaptation, RASS is introduced—a runtime-aware MOO solver that allows for low-overhead and effective dynamic adjustment of the execution. The key principle behind RASS's design is to explicitly consider during the MOO solution stage that adaptation may subsequently be required at deployment time. As such, RASS operates in two steps: (a) it generates a set of alternative execution configurations with diverse trade-offs prior to deployment, and (b) configures the inference engine with a policy of switching among them.

Towards alleviating the impact of device heterogeneity and resource fluctuation, `CARIn` operates exclusively at the system level, bypassing the need to produce an optimal model for each target device. Model-level solutions typically include the design, exploration, training, and adaptation of a DNN's architecture to specific target devices and resource availability changes. These procedures can be cumbersome, time-consuming, and lead to complex pipelines. Instead, the proposed framework employs a repository of pre-trained models with varying architectures

and complexities. The singular requisite action in relation to the models entails the application of post-training quantization.

The design of the outlined framework was driven by the fact that satisfying SLOs depends not only on the target model but also on the specific target device, especially the processor in use. Consequently, CARIn's primary objective is to determine, at any given time, the most suitable model-processor pair (or pairs) for a specified device. Here, the term "processor" extends beyond its general definition to encompass specific configurations, such as the number of threads utilized on a CPU or the precision settings employed on a GPU. Internally, the MOO framework expresses this as a DL-based, device-centric problem, to effectively capture both the application's SLOs and the unique characteristics of the target device. Given the device-specific nature of the MOO formulation, a distinct optimization problem is formed for each given device, effectively circumventing the challenge posed by device heterogeneity. Additionally, in order to facilitate real-time adaptation, CARIn leverages the device's intra-level heterogeneity, specifically the array of available processors, as well as the range of solutions offered by the RASS solver, which allow the adoption of a swift and efficient switching mechanism between execution plans.

3.2.1 Workflow

Figure 3.1 depicts CARIn's operational flow, which is divided into the sequential offline and online phases. The offline component is responsible for constructing and resolving the device-specific MOO problem. Then, at runtime, the online component's Runtime Manager (RM) constantly monitors the application's dynamic behavior, ensuring real-time adaptation to emergent changes. Algorithm 3.1 presents a comprehensive top-level overview of the proposed framework, delineating its primary components and illustrating their main operations. These operations will be thoroughly elucidated in Section 3.3.

The framework's input parameters are: (a) the designated DL task (or tasks) associated with the application, (b) the target device's characteristics, and (c) the stipulated service-level objectives, whereas the outputs consist of: (a) the set of solutions (designs $\mathcal{D}$), and (b) the switching policy SP.

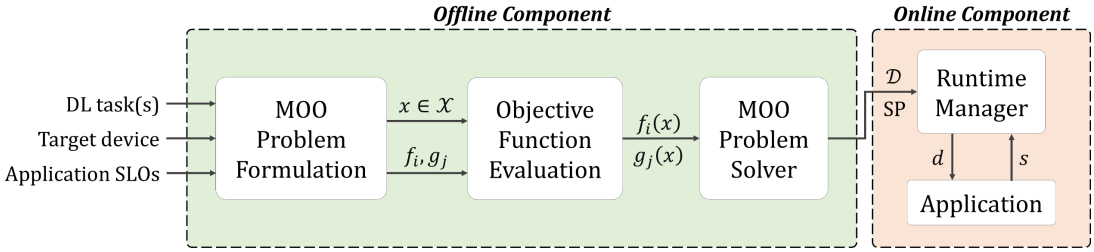

Figure 3.1 High-level workflow of CARIn

The specified DL task(s) dictate the set of models to be considered during the optimization process. A model in CARIn is represented by the following tuple:

$$m = (arch, params, s_{\text{in}}, task, ds, pr) \tag{3.1}$$

where *arch* is the model's architecture (*i.e.*, layers and connections), *params* are the model's trained parameters, s_{in} is the input size, *task* is the target DL problem, *ds* is the name of the corresponding test dataset, and *pr* is the numerical precision in order to account for quantized models.

Algorithm 3.1 CARIn's End-to-End Operation

Input: DL task(s)
Target device
Application SLOs
Output: Designs, $\mathcal{D}$
Switching Policy, SP

/* MOO Problem Formulation */
$\mathcal{HW} \leftarrow$ ObtainHardwareCharacteristics(Target device)
if *app_type == single-DNN* **then**
  $\mathcal{X} \leftarrow$ ConstructDecisionSpace(DL task, $\mathcal{HW}$)
else
  $\mathcal{X} \leftarrow$ ConstructDecisionSpace(DL tasks, $\mathcal{HW}$)
end
$f_i, g_j \leftarrow$ ExtractFunctions(Application SLOs)
/* Objective Function Evaluation */
$f_i(x), g_j(x) \leftarrow$ EvaluateFunctions($\mathcal{X}, f_i, g_j$)
/* MOO Problem Solver */
$\mathcal{X}' \leftarrow \left\{x \mid g_j(x) \leq 0\right\}$ // apply constraints
$opt(x) \leftarrow$ CalculateOptimality($\mathcal{X}'$)
$\mathcal{X}'_s \leftarrow$ Sort($\mathcal{X}'$, opt)
$\mathcal{D}$, SP $\leftarrow$ Search($\mathcal{X}'_s$)
/* Runtime */
while *true* **do**
  $c \leftarrow$ Analyze(s) // c indicates a change in resource availability
  if c **then**
    $d_{\text{new}} \leftarrow$ RM($\mathcal{D}$, SP, c)
  end
end

The target device defines the hardware resources at the system's disposal, which are represented by the tuple:

$$hw = \left(ce, op(ce)\right) \tag{3.2}$$

where $ce \in \mathcal{CE}$ is the compute engine (*i.e.*, processor) performing the inference computations and $op(ce)$ is a set of options tied to the given processor, *e.g.*, the number of CPU threads or the GPU's numerical precision. The tuple of tunable system parameters can be extended to capture a more detailed space, *e.g.*, by including the DVFS governor selection which determines the dynamic voltage and frequency scaling policy of the device.

An individual model m running under the selected system parameters hw represents a single execution configuration:

$$e = \langle m, hw \rangle \in \mathcal{E} \tag{3.3}$$

During the MOO Problem Formulation stage, CARIn considers every generated space of execution configurations, $\mathcal{E}_i$, in order to form the problem's decision space, $\mathcal{X}$, depending on whether the application requires single- or multi-DNN execution. At the same time, the application's SLOs delineate the MOO problem's objective functions and constraints, denoted as f_i and g_j respectively. Once the problem is formulated for the target device, the Objective

Function Evaluation stage evaluates each function for every $x \in \mathcal{X}$. Following this, CARIn's MOO Problem Solver is poised to solve the MOO problem. The functions CalculateOptimality, Sort, and Search shown in Algorithm 3.1, which constitute the three stages of the solver, are discussed in detail in Subsection 3.3.3.

In order for CARIn to accommodate runtime adaptation, it is important to establish a robust system for perpetually monitoring the dynamic aspects of the executing application and the state of the device itself. This ongoing vigilance enables timely recognition of abrupt alterations in operational conditions, thereby facilitating immediate corrective measures. This subsystem is called the Runtime Manager (RM). The output of CARIn's solving algorithm consists of a set $\mathcal{D}$ of highest-performing solutions, called designs, which are passed to RM along with the appropriate switching policy (SP). Leveraging a collection of periodically captured statistics s from the Application's runtime, the RM module has the ability to discern dynamic changes in resource allocation (c in Algorithm 3.1) and rapidly switch to an alternative design d_{new} to effectively and robustly meet the application-level SLOs.

3.3 Multi-Objective Optimization Framework

Multi-objective optimization (MOO) constitutes a mathematical and computational approach employed to find the best solutions or trade-offs in scenarios that involve multiple interrelated and, at times, antagonistic objectives [99], [100]. The appropriateness of a MOO framework for the problem at hand is underscored by (a) the inherent nature of DL application SLOs, which typically comprise objectives that exhibit conflicts, and (b) the inherent attribute of MOO to yield a solution space rich in diversity, which, in turn, can enable dynamic adaptation.

3.3.1 MOO Problem Formulation

For CARIn's DL-based MOO formulation, the following mathematical description is adopted:

$$\begin{aligned} &\text{min/max} \quad f_i(x), && 1 \le i \le N \\ &\text{subject to} \quad g_j(x) = g_j\left(h_j(x)\right) \le 0, && 1 \le j \le P \end{aligned} \tag{3.4}$$

where x denotes the decision variable, N is the number of objective functions, f_i is the i-th objective function, P is the number of inequality constraints, and g_j is the j-th inequality constraint, which is always a composite function of a given inner function h_j. Note that when there is only a single objective function ($N = 1$), then the problem is reduced to single-objective optimization (SOO). The problem's objective functions and constraints are extracted from the application's SLOs, which can be split into two categories:

- **Broad SLOs**: Such objectives define the problem's objective functions and come in the form of $\langle \text{min/max}, p \rangle$, where p is a DL-related performance metric. For instance, $\langle \text{max}, mIoU \rangle$ means that the mean Intersection-over-Union (mIoU) accuracy metric should be maximized for an image segmentation task. For CARIn, this objective translates to the maximization of the objective function $f(x) = A(x) = mIoU(x)$.
- **Narrow SLOs**: These objectives define the problem's constraints and come in the form of $\langle \text{min/max/avg/std/}n^{\text{th}}, p, v \rangle$, which means that the minimum, maximum, average, standard deviation or n^{th} percentile value of p is bounded by a target value v. For instance, $\langle \text{avg}, L, 15 \rangle$ means that the average latency needs to be less than 15 ms, which translates to the constraint $g(x) \le 0$, where $g(x) = g(h(x)) = g(L(x)) = \bar{L}(x) - 15$.

Given that both types of objectives concern the same set of performance metrics, it follows that both the objective and inner functions, f_i and h_j, share a common function space, denoted by $\mathcal{F}$, which encompasses the entirety of available functions associated with various DL performance metrics. For this reason, in cases where the application defines constraints without explicitly specifying objective functions, `CARIn` can duly regard all specified inner functions h_j as objective functions as well.

3.3.1.1 Single-DNN Setting

When there is only one DL task to optimize, the decision variable x is a single execution configuration e, as defined in Equation (3.3). Therefore, the execution configuration space $\mathcal{E}$ effectively transforms into the decision space $\mathcal{X}$:

$$x_{\text{single}} = e = \langle m, hw \rangle \in \mathcal{X}_{\text{single}} = \mathcal{E} \tag{3.5}$$

For the objective functions, `CARIn` leverages the following performance metrics tailored specifically to DNNs:

- **Size (*S*)**: Size is conventionally represented by either the total count of parameters within the neural network or the physical file size of the model stored in memory.
- **Workload (*W*)**: This metric is typically measured in terms of numerical operations, such as floating-point operations (FLOPs) or multiply-accumulate operations (MACs).
- **Accuracy (*A*)**: Accuracy is contingent upon the specific DL task in question, *e.g.*, top-1 accuracy for classification tasks or exact match for question answering tasks.
- **Latency (*L*)**: Latency delineates the temporal lag between the transmission of input data to the DNN model and the reception of the corresponding output. It is quantified in units of milliseconds or seconds.
- **Throughput (*TP*)**: Throughput provides an indication of the model's real-time processing capabilities and is computed as the total number of input samples (batch size), divided by the total inference latency. This metric is denominated in samples per second (*e.g.*, images per second when images constitute the inputs).
- **Energy Consumption (*E*)**: This metric is of paramount importance for the evaluation of the energy efficiency of DNN applications in resource-constrained environments and is measured in energy units, such as watt-hours or joules.
- **Memory Footprint (*MF*)**: Memory footprint encapsulates the extent of random-access memory (RAM) required for the loading and execution of a DNN. It is traditionally assessed in terms of memory size units, such as megabytes (MB) or gigabytes (GB).

Overall, the set of potential objective functions in single-DNN cases is denoted as:

$$\mathcal{F}_{\text{single}} = \{S, W, A, L, TP, E, MF\} \tag{3.6}$$

collectively empowering a multifaceted assessment of DNN models and providing a holistic understanding of their performance across diverse dimensions.

It is important to recognize that the latency and energy consumption metrics are subject to inherent fluctuations when executing DNNs on mobile devices. These fluctuations can arise due to various factors, including device load, temperature, input values, and other environmental variables (see Subsection 3.3.3.2). As a result, relying on a single, instantaneous value may not provide a robust and representative assessment of system performance. To account for these fluctuations, `CARIn` considers statistical measures, such as the average or maximum energy consumption or the variance of the latency, as objective functions.

3.3.1.2 Multi-DNN Setting

When there are M independent DNNs to optimize jointly, the decision variable x comprises of M distinct execution configurations e_i, $1 \leq i \leq M$. Hence, the decision space $\mathcal{X}$ is an M-dimensional space, where each component of the decision variable can separately take values in the corresponding execution configuration space $\mathcal{E}_i$:

$$x_{\text{multi}} = \{e_1,\ldots,e_M\} = \{\langle m, hw\rangle_1, \ldots, \langle m, hw\rangle_M\} \in \mathcal{X}_{\text{multi}} = \mathcal{E}_1 \times \ldots \times \mathcal{E}_M \tag{3.7}$$

The array of potential objective functions is expansively broadened to encompass an additional triad of performance metrics pertaining to parallel execution [101], [49]:

- **Normalized Turnaround Time (*NTT*)**: *NTT* serves as a quantifier of the perceived execution slowdown during multi-DNN execution. The *NTT* for the i-th DNN is computed as:

$$NTT_i = \frac{L_i^{\text{M}}}{L_i^{\text{S}}} \tag{3.8}$$

  where L_i^{S} and L_i^{M} are the average latencies of the i-th DNN under the single- and multi-DNN modes. NTT_i is a value greater than or equal to 1, with lower values indicating superior performance. For the sake of standardization across models, it is common practice to calculate the average or maximum *NTT*.
- **System Throughput (*STP*)**: *STP* quantifies the accumulated single-DNN progress under multi-DNN execution and is computed as:

$$STP = \sum_{i=1}^{M} NP_i = \sum_{i=1}^{M} \frac{1}{NTT_i} = \sum_{i=1}^{M} \frac{L_i^{\text{S}}}{L_i^{\text{M}}} \tag{3.9}$$

  where NP_i is the normalized progress of the i-th DNN. Its maximum magnitude is M, with higher values signifying enhanced performance.
- **Fairness (*F*)**: The concept of fairness in a multi-DNN execution environment is contingent upon the equitable relative progress experienced by co-executing DNNs, in comparison to their single-DNN execution counterparts. Fairness, as denoted herein, is quantified as the minimum ratio of relative normalized progress rates observed among any two DNNs concurrently operating within the system:

$$F = \min_{i,j} \frac{NP_i}{NP_j} \tag{3.10}$$

  This metric adheres to a higher-is-better paradigm with values within the range [0, 1], where 0 signifies an absence of fairness and 1 perfect fairness.

As a consequence, `CARIn`'s objective function set is augmented to encompass both single-DNN metrics, which pertain to individual tasks or DNNs, and multi-DNN metrics, which characterize the collective performance of the entire system during concurrent execution:

$$\mathcal{F}_{\text{multi}} = \{S_i, W_i, A_i, L_i, TP_i, E_i, MF_i\} \cup \{STP, NTT, F\}, 1 \leq i \leq M \tag{3.11}$$

3.3.2 Objective Function Evaluation

Upon the formulation of the device-specific MOO problem, it becomes necessary to assess each objective function across the entire set of decision variables $x \in \mathcal{X}$. Assessing these functions

is straightforward for certain objectives; however, it presents challenges for device-dependent functions like *E* and *MF*, and those relying on latency, including *L*, *TP*, *STP*, *NTT*, and *F*. The approach adopted by `CARIn` for this evaluation involves the profiling of functions on individual target devices. In practical terms, this entails the deployment of all candidate models on each target device, followed by the measurement of each device-reliant objective function for all feasible model-processor combinations. While comprehensive, this procedure is inherently time-consuming and, in many instances, such as in multi-DNN cases, infeasible for seamless integration into real-world scenarios and practical applications. However, the optimization of the evaluation process itself does not constitute a primary objective of this work. Potential enhancements of this aspect are extensively discussed in Section 3.7.

3.3.3 MOO Problem Solver

Following the formulation of the problem and the evaluation of the associated functions, the conclusive stage involves resolving the optimization problem. The initial step is to apply the problem's constraints, thereby restricting the decision variables to the constrained decision space $\mathcal{X}'$, defined as:

$$\mathcal{X}' = \left\{ x \mid g_j(x) \leq 0, \forall j \right\} \tag{3.12}$$

MOO problems are frequently addressed using evolutionary algorithms, such as NSGA-II, SMS-EMOA, and MOEAD, or swarm-based algorithms, such as Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) [102]. These algorithms systematically explore the decision variable space to discover the Pareto frontier, which represents the optimal trade-offs among conflicting objectives. While these algorithms excel in identifying the optimal execution configuration, potential runtime issues may either alter the solution space, consequently affecting the Pareto frontier of a MOO problem, or introduce new constraints that were not considered during the problem's formulation. Consequently, to address the potential decline in performance, it becomes imperative to rerun these algorithms whenever a runtime issue arises. However, such repetitive executions are impractical for real-life applications and systems.

To address this challenge, a runtime-aware sorting and search algorithm is introduced, denoted as RASS, whose primary goal is to solve a device-specific MOO problem once, while concurrently addressing potential future runtime challenges. To achieve this, RASS considers both non-dominated and dominated solutions in a predictive manner, estimating the impact of possible runtime issues. In addition to providing the initial solution, d_0, RASS also yields a set of supplementary runtime designs, d_i, which serve as a proactive measure for runtime adaptation, *i.e.*, in instances where the currently employed design encounters performance issues. This approach alleviates the need for repetitive executions of optimization algorithms.

The operation of RASS involves a sorting stage followed by a search stage. To accommodate both non-dominated and dominated solutions, the solving algorithm initially sorts candidate solutions according to their optimality (Subsection 3.3.3.1), a metric quantifying the distance from the problem's utopia point. Subsequently, based on this sorting, RASS identifies a set of solutions (Subsection 3.3.3.4) representing the various execution plans of the application which correspond to possible runtime issues (Subsection 3.3.3.2), along with a switching policy facilitating prompt transitioning between them (Subsection 3.3.3.3) for the RM module.

3.3.3.1 Optimality

To quantify optimality for a given candidate solution $x \in \mathcal{X}'$, the weighted Mahalanobis distance between the solution's objective vector, defined as $f(x) = [f_1(x), f_2(x), \ldots, f_N(x)]$, and the utopia point, represented as $up = [up_1, up_2, ..., up_N]$, is first calculated:

$$d(x) = \sqrt{\sum_{i=1}^{N} w_i^2 \frac{[f_i(x) - up_i]^2}{s_i^2}} \tag{3.13}$$

where w_i is the user-supplied weight for the i-th objective, s_i^2 is the calculated variance of the i-th objective, and each component of the utopia point depends on the corresponding objective function:

$$up_i = \begin{cases} \max f_i, & \text{if } f_i \in \{A, TP, STP, F\} \\ \min f_i, & \text{if } f_i \in \{S, W, L, E, MF, NTT\} \end{cases} \tag{3.14}$$

By utilizing the Mahalanobis distance, the disparate scales of the diverse objectives are effectively accommodated. Consequently, optimality could also be regarded as a metric of fairness for the problem's objective functions. However, it is important to recognize that these functions may hold varying significance for the problem. To address this, users are provided with the ability to define weights, introducing a formal mechanism for enabling tailored optimization strategies. Notably, the calculated distances range within the interval $[0, d_{\max}]$, where the maximum distance is:

$$d_{\max} = \sqrt{\sum_{i=1}^{N} w_i^2 \frac{(\max f_i - \min f_i)^2}{s_i^2}} \tag{3.15}$$

This factor necessitates the use of normalization, which results in the distance being confined to the [0, 1] range:

$$d_s(x) = \frac{d(x)}{d_{\max}} \tag{3.16}$$

The optimality metric for each $x \in \mathcal{X}'$ can then formally be defined as the reciprocal of the scaled weighted Mahalanobis distance, thus, its range extends from $[1, +\infty)$:

$$opt(x) = \frac{1}{d_s(x)} \tag{3.17}$$

Utilizing these values, the candidate solutions are sorted in descending order, resulting in the creation of the sorted decision space $\mathcal{X}_s$.

3.3.3.2 Runtime Challenges

During the runtime of the application, a multitude of complex and often unpredictable dynamic alterations in the device's resource availability may occur. These fluctuations can significantly impact the problem formulation in different ways, thus necessitating carefully designed and targeted approaches for their effective management. `CARIn` focuses on addressing two main challenges, regarding the processors and memory of the target device:

- **Processor overload or overheating**: Processor-related concerns manifest when the processor at use is continuously subjected to sustained processing demands exceeding its peak processing capacity, primarily due to resource-intensive computational tasks. The protracted imposition of such an overload condition may subsequently lead to overheating, which means the escalation of the SoC's temperature to a critical and potentially harmful level. Overheating may also result from insufficient cooling mechanisms or other impediments hindering the effective dissipation of heat by the SoC. As a protective measure against potential harm, mobile SoCs are equipped with thermal throttling capabilities, which are activated when temperatures exceed predefined thresholds. Thermal throttling encompasses the deliberate reduction of the processor's clock speed and performance to mitigate heat generation and maintain a safe temperature range. The intricate interplay between processor overload and overheating significantly impacts performance and power consumption, underscoring the significance of diligent management and effective mitigation strategies.
- **Variability in RAM utilization**: Owing to the multifaceted nature of mobile devices, the utilization of random-access memory is also characterized by dynamic fluctuations. Within the execution scope of an application, numerous ancillary applications, processes, or services continually initiate and terminate in the background, potentially culminating in an unforeseen saturation of RAM capacity. Consequently, this phenomenon may precipitate performance-related challenges, encompassing lag, application crashes, and an overall deceleration of device functionality. Furthermore, the perpetual management of excessive RAM consumption may also entail elevated power consumption, thereby engendering consequential ramifications.

3.3.3.3 Model/Processor Switching

In response to runtime fluctuations, `CARIn`'s RM adopts a strategic approach that involves alterations to either the model (change model, CM), processor (CP), or both (CB) within the current execution plan. These three fundamental adjustments serve as effective measures for mitigating the challenges encountered during runtime. To this end, a prioritization scheme is introduced. In the case of processor-related phenomena, `CARIn` prioritizes transferring DNN execution from the currently used processors to inactive ones (CP or CB). This transition allows the overloaded or overheated processor to dissipate excess heat and gradually restore its performance. In cases where migration is not a viable option, such as in devices limited solely to CPU usage or multi-DNN scenarios where all processors are occupied, `CARIn` employs an alternative approach which involves replacing the current model with one of reduced computational workload (CM). Conversely, addressing the memory-related issue involves transitioning to a more compact model either on the same (CM) or a different processor (CB).

3.3.3.4 Design Selection & Switching Policy

A primary principle guiding the design of RASS is to ensure low complexity in order to facilitate rapid switching. This objective manifests in the generation of a relatively small number of designs, which offers two additional key benefits:

1. **It minimizes the storage requirements** for the models.
2. **It maintains a concise switching policy**, comprising only a limited number of transition rules.

For RM to determine the appropriate timing to transition to a new execution plan, several system parameters, related to the workload distribution across processors and the aggregate

memory utilization, need to be continuously monitored. These parameters are represented by the boolean variables c_{ce} and c_m, indicating the presence of issues pertaining to a processor *ce* and the memory, respectively.

The first step in identifying the solutions to the problem involves identifying the sets of different model-to-processor mappings viable for processor switching, *i.e.*, for reallocating DL execution to idle processors. Symbolizing the number of these sets as T, in consideration of RASS's need for simplicity, if $T > 3$, only the top three sets are retained, corresponding to the highest attained optimality scores. Next, the sorted decision space $\mathcal{X}_s$ is partitioned into T distinct subspaces $\mathcal{X}_i$, each corresponding to specific model-to-processor mappings and arranged in descending order of observed optimality. Regarding processor-related phenomena, designs associated with the highest optimality score within each set are selected:

$$d_i = \mathcal{X}_i[0], i = 0, \dots, T-1 \tag{3.18}$$

To address the memory-related challenge, the solution that exhibits the smallest possible memory footprint is extracted:

$$d_m = \underset{x}{\operatorname{argmin}}\, MF(x), x \in \mathcal{X}_i, i = 0, \dots, T-1 \tag{3.19}$$

Lastly, complementary designs are obtained for two highly improbable scenarios. The first one arises when all related processors face an overload issue, while the memory does not, prompting the extraction of the solution with the lightest workload:

$$d_w = \underset{x}{\operatorname{argmin}}\, W(x), x \in \mathcal{X}_i, i = 0, \dots, T-1 \tag{3.20}$$

and the second scenario surfaces when both the processors and memory encounter issues simultaneously, necessitating the identification of the solution that strikes the optimal balance between memory usage and workload among d_m and d_w:

$$d_{wm} = \begin{cases} d_w, & \text{if } C[MF(d_w), W(d_w)] < C[MF(d_m), W(d_m)] \\ d_m, & \text{else} \end{cases} \tag{3.21}$$

where the normalized sum is used to compute the cost function C. Collectively, the set of designs is denoted as:

$$\mathcal{D} = \{d_i, d_m, d_w\}, i = 0, \dots, T-1 \tag{3.22}$$

therefore, RASS can generate a maximum of five designs for a MOO problem, since $T \leq 3$.

After establishing the set of designs, RASS's final step involves crafting the rule-based switching policy, which serves as a reference for the RM module, guiding its decision-making process each time the boolean variables c_{ce} or c_m change. With the aim of ensuring simplicity and conciseness in the rule set, the selection of a new design is contingent solely upon the state of the environmental variables and independent of the presently employed design. The rationale behind the construction of the rules is deliberately straightforward, as demonstrated in Subsection 3.6.2, where two representative use cases are presented and analyzed.

3.4 Implementation

`CARIn` is implemented in Java for the Android operating system. Its primary integration leverages the TensorFlow Lite (TFLite) package in its nightly build to facilitate on-device DNN execution, as well as its delegates to access mobile accelerators. The concurrent execution of multiple DNNs is achieved through the utilization of the `java.util.concurrent` package.

In order to create the model suite used for the framework's evaluation, *i.e.*, for model retrieval, training, and preparation, TensorFlow (v2.12.0) was employed in Python. The proposed framework seamlessly interfaces with the TensorFlow Hub and Hugging Face model repositories, which allows researchers to easily access and experiment with a wide range of pre-trained models on publicly available datasets for their specific use cases. Additionally, the TFLite Converter's optimization module was utilized to apply post-training quantization to the models in order to enhance inference speed, efficiency and accelerator compatibility.

Regarding the objective function evaluation process, a diverse set of tools and libraries is employed. For accuracy assessments, custom evaluation scripts were used, as well as TFLite's image classification evaluation tool for the ImageNet ILSVRC 2012 task. Furthermore, to comprehensively capture the model's computational complexity and resource requirements, the `tflite` Python package was used to count model parameters and FLOPs. Lastly, to assess the on-device performance of the models, the C++-based TFLite benchmark tool was employed. This tool offers a comprehensive suite of measurements, encompassing execution time, memory utilization, and other pertinent metrics, thereby providing a robust evaluation of the models' real-world performance characteristics.

3.5 Experimental Methodology

This section outlines the experimental methodology that forms the foundation of this research, providing insight into the comprehensive approach taken to investigate the study's core objectives. The methodology is structured into several key subsections, each addressing a critical aspect of the experimental design. Figure 3.2 illustrates the toolflow used to conduct the experiments.

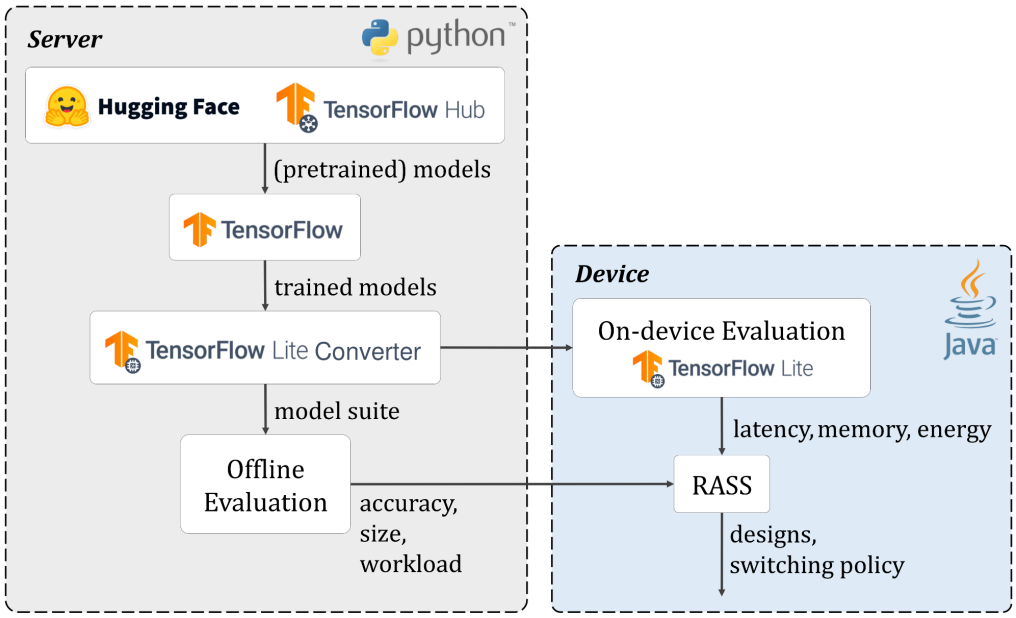

Figure 3.2 Toolflow for the evaluation of CARIn

3.5.1 Quantization

`CARIn` embraces post-training quantization as one of the most simple and mobile-friendly compression methods presently available, with benefits not only in model size, but also in latency and memory requirements. Apart from this, quantization becomes indispensable for the execution of DNNs within DSPs or NPUs designed to primarily support integer models, thus unlocking complete compatibility with mobile accelerators. Details on quantization techniques

are presented in Subsection 2.5.2.1. Notably, additional compression methods which also introduce trade-offs between accuracy and complexity, such as weight pruning or clustering, are orthogonal to the proposed framework and amenable to integration. The potential synergy resulting from the combined application of various compression techniques merits further investigation.

3.5.2 Application Scenarios, Models and Tasks

The following section outlines four distinct application use cases that serve as the foundation for the experimental evaluation. These scenarios represent diverse real-world settings in which the research findings are tested and validated, offering valuable insights into the effectiveness of the proposed formulation and methodology.

Notably, the first two scenarios pertain to the execution of a single DNN, whereas the latter two involve the parallel execution of multiple DNNs, enabling the assessment of performance outcomes in cases where dependencies among multiple DNNs are present. This dichotomy is instrumental in affording a comprehensive evaluation of the methodology's versatility, applicability, and scalability. Specific SLOs are defined for each use case, accompanied by a list of pre-trained models considered during evaluation, along with their device-independent assessment, which includes: (a) the relevant accuracy metric for both the original and quantized variants, (b) the computational workload in FLOPs, and (c) the model size expressed in the number of parameters.

3.5.2.1 Use Case #1 (UC1)

The first single-DNN scenario examines the practical application of real-time image classification. In this setting, the camera of a mobile device continuously captures frames that require prompt and accurate recognition. The term "real-time" is qualified by a temporal restriction mandating that the maximum permissible latency is 41.67 ms, underscoring the necessity to uphold a recognition rate of no less than 24 frames per second (FPS). The principal objectives of this use case encompass the joint maximization of accuracy and throughput. Mathematically, this MOO problem comprises two objective functions and one single constraint:

$$\begin{aligned} \max \quad & A(x), TP(x) \\ \text{subject to} \quad & \max L(x) \leq 41.67 \text{ ms} \end{aligned} \tag{3.23}$$

For UC1 the ImageNet-1k dataset [103] was used. Table 3.1 lists the eight models under consideration, which are drawn from four distinct families: MobileNets [62], EfficientNets [26], RegNets [104], and MobileViTs [105]. The rationale behind this extensive model selection is to ensure a well-rounded exploration of compact and mobile-friendly architectures that span a broad spectrum, encompassing both conventional CNNs and emerging Transformer-based models. Each of these architectural paradigms exhibits unique characteristics and design principles. It is important to note that the inclusion of higher-accuracy models, such as NASNet [106] and ConvNeXt [107], was also considered. However, the assessment revealed that these models did not meet the stipulated latency constraint. As a result, they were excluded from the study to ensure adherence to the predefined performance criteria.

Table 3.1 UC1 Models

DL Task	Architecture	Image Size	FLOPs	#Params	Top-1 Accuracy (%)				
					FP32	FP16	DR8	FX8	FFX8
Image Classification on ImageNet-1k	MobileNet V2 1.0	224×224	0.60 G	3.49 M	71.92	71.96	71.65	71.28	71.26
	RegNetY 008	224×224	1.60 G	6.25 M	74.28	74.28	74.18	74.45	74.47
	MobileViT XS	256×256	2.10 G	2.31 M	74.61	74.61	-	-	-
	EfficientNet Lite0	224×224	0.77 G	4.63 M	75.19	75.23	75.14	75.09	75.11
	MobileNet V2 1.4	224×224	1.16 G	6.09 M	75.66	75.68	75.47	75.41	75.45
	RegNetY 016	224×224	3.23 G	11.18 M	76.76	76.76	76.62	76.92	76.84
	MobileViT S	256×256	4.06 G	5.57 M	78.31	78.30	-	-	-
	EfficientNet Lite4	300×300	5.11 G	12.95 M	80.81	80.80	80.78	80.69	80.71

3.5.2.2 Use Case #2 (UC2)

In the second single-DNN scenario, the task of text classification is studied, with a particular emphasis on the memory requirements of the models. To this end, a memory constraint is imposed, stipulating that the executing DNN's maximum memory footprint must not exceed 90 MB. The objectives of this use case revolve around three critical factors: minimizing the average latency, reducing the model size, and maximizing accuracy. Mathematically, this MOO problem encompasses three objective functions and a singular constraint:

$$\begin{aligned} \min \quad & \bar{L}(x), S(x) \\ \max \quad & A(x) \\ \text{subject to} \quad & MF(x) \leq 90 \text{ MB} \end{aligned} \tag{3.24}$$

For UC2, three pre-trained Transformer models were obtained, trained on various large datasets, including Reddit comments and 2ORC citation pairs, and subsequently fine-tuned on Emotions [108], a dataset comprising of English Twitter messages which is employed for the task of classifying input sequences into six distinct emotions. The dataset's split configuration was adopted, which allocated 16000 samples for training, 2000 for validation, and 2000 for testing. The reported Top-1 Accuracy corresponds to the dataset's test set. The selected models, detailed in Table 3.2, encompass the traditional BERT architecture in a lightweight version, alongside two mobile-grade models: XtremeDistil [109] and MobileBERT [110]. The letter "L" in each model's name stands for the number of Transformer layers and "H" stands for the hidden dimension.

Table 3.2 UC2 Models

DL Task	Architecture	Sequence Length	FLOPs	#Params	Top-1 Accuracy (%)				
					FP32	FP16	DR8	FX8	FFX8
Text Classification on Emotions	BERT-L2-H128	64	0.05 G	4.31 M	92.10	92.10	91.90	91.75	91.75
	XtremeDistil-L6-H256	64	0.63 G	12.57 M	93.30	93.30	93.20	93.15	93.20
	MobileBERT-L24-H512	64	2.66 G	24.33 M	93.80	93.80	93.80	93.65	94.10

To further enhance their suitability for mobile deployment, BERT and XtremeDistil were optimized by replacing the computationally expensive GELU activation function with the more efficient ReLU and substituting Layer Normalization with Batch Normalization. These modifications, guided by the experiments on Transformer model optimizations discussed in Subsection 4.4.2, significantly improve the models' compatibility with resource-constrained devices without compromising their performance. Such targeted adjustments demonstrate the

critical role of algorithmic refinements in bridging the gap between cutting-edge DL architectures and practical on-device applications.

3.5.2.3 Use Case #3 (UC3)

The first multi-DNN scenario employs two DNNs for the purpose of scene recognition. One DNN is dedicated to processing and classifying images, whereas the other can process audio data in order to identify sounds from the device's surroundings. These models operate concurrently, running in parallel, and their outputs are collectively utilized to determine the specific scene within which the mobile device is situated.

In this scenario, the objective is to minimize both the average latency and its standard deviation, while simultaneously maximizing the attained accuracy. Two latency constraints are imposed for both tasks, mandating that (a) the average latency remains consistently below 100 ms to ensure near-real-time responsiveness, and (b) the standard deviation of latency stays below 10 ms for minimal fluctuations. The inclusion of the latency's standard deviation aims to minimize performance variability, which still constitutes a withstanding challenge for on-device inference. Mathematically, this MOO problem is formulated as:

$$\begin{aligned} \min \quad & \bar{L}_i(x), \sigma_{L_i}(x) \\ \max \quad & A_i(x) \qquad\qquad \text{for } i = 1, 2 \\ \text{subject to} \quad & \bar{L}_i(x) \leq 100 \text{ ms}, \sigma_{L_i}(x) \leq 10 \text{ ms} \end{aligned} \tag{3.25}$$

Table 3.3 presents the models for each task. For the vision task, three EfficientNet Lite models were fine-tuned on the MIT Indoor Scenes dataset [111], which includes 67 classes and 100 images per class (80 for training and 20 for testing). The Top-1 Accuracy on the test set is reported.

Table 3.3 UC3 Models

DL Task	Architecture	Input Size	FLOPs	#Params	Accuracy				
					FP32	FP16	DR8	FX8	FFX8
Scene Classification on MIT Indoor Scenes	EffNet Lite0	224×224	0.59 G	3.44 M	69.78	69.70	68.96	69.18	69.18
	EffNet Lite2	260×260	1.51 G	4.87 M	76.72	76.72	77.16	77.69	77.54
	EffNet Lite4	300×300	4.57 G	11.76 M	79.33	79.33	79.18	79.78	79.48
Audio Classification on AudioSet	YAMNet	15600	0.14 G	3.75 M	0.3756	0.3757	0.3620	-	-

For the audio task, the YAMNet model was considered, which is trained on the AudioSet dataset [112] for multi-label classification. The dataset consists of 521 sound events (classes) and 18k samples. The mean Average Precision (mAP) on the validation set is reported. YAMNet's input waveform can vary in length. In the experiments, the model's minimum possible length of 975 ms is used, which corresponds to 15600 input samples and a total workload of 0.14 GFLOPs.

3.5.2.4 Use Case #4 (UC4)

In the second multi-DNN scenario, three distinct models designed for facial attribute prediction tasks, namely gender, age and ethnicity estimation, are employed. These models are conceptualized as the second stage of a face detection and attribute prediction pipeline, wherein they operate concurrently on the same set of input images. As such, it is imperative for these models to adhere to stringent latency constraints to ensure minimal impact on the overall pipeline. UC4's objectives revolve around the collective optimization of five key metrics for

each model, specifically average latency, standard deviation of latency, size, memory footprint and accuracy, all while adhering to a maximum latency threshold of 10 ms. Formally:

$$\begin{aligned} \min \quad & \bar{L}_i(x), \sigma_{L_i}(x), S_i(x), MF_i(x) \\ \max \quad & A_i(x) \qquad\qquad \text{for } i = 1, 2, 3 \\ \text{subject to} \quad & \max L_i(x) \leq 10 \text{ ms} \end{aligned} \tag{3.26}$$

In UC4, the training data are sourced from the UTKFace dataset [113]. To ensure relevance to real-time applications, the dataset is filtered to retain samples corresponding to individuals within the age range of 18 to 75 years. This preprocessing step refines the dataset to a total of 18.6k facial images, which are subsequently partitioned into training, validation, and testing sets with a ratio of 72/8/20%, respectively. The employed models leverage MobileNetV2 as the backbone architecture, extracting 576 features of size 4×4, which are used for predicting the outcomes across the three distinct facial attribute prediction tasks. A key distinction of UC4 within this study is its incorporation of batching during inference, setting it apart from other tasks. Specifically, the models are configured with a batch size of 4, a choice driven by the common scenario in which a preceding face detection component identifies multiple faces within a single image, necessitating simultaneous processing. Table 3.4 details the attained accuracy metrics for each task on the filtered dataset's test set: Binary Accuracy for gender recognition, mean Absolute Error for age recognition, and Top-1 Accuracy for ethnicity recognition across 5 output classes.

Table 3.4 UC4 Models

DL Task	Architecture	Image Size	FLOPs	#Params	Accuracy				
					FP32	FP16	DR8	FX8	FFX8
Facial Attribute Prediction on UTKFace	GenderNet-MNV2	62×62	0.04 G	0.66 M	95.12	94.95	94.90	94.79	94.90
	AgeNet-MNV2	62×62	0.04 G	0.66 M	5.976	5.974	5.964	5.947	5.923
	EthniNet-MNV2	62×62	0.04 G	0.66 M	78.17	78.04	78.55	79.30	79.14

3.5.3 Mobile Devices

In this study, the following three smartphones were selected for evaluation: Google Pixel 7, Samsung Galaxy S20 FE, and Samsung Galaxy A71. These devices have been deliberately chosen to represent distinct categories within the modern mobile phone landscape. A71 serves as an archetype of a mid-tier device, whereas S20 and P7 exemplify the high-end category, showcasing state-of-the-art features and cutting-edge technology. A detailed overview of the specifications and processing capabilities of these smartphones is shown in Table 3.5.

Each of the three devices is equipped with its own NPU. Concretely, P7 incorporates a custom mobile-oriented Tensor Processing Unit (TPU), S20 features the EDEN API, which grants access to the Exynos NPU for fixed-point models and specialized GPU kernels for floating-point models, and lastly, A71 hosts the Hexagon Tensor Accelerator (HTA), a dedicated compute engine for fixed-point CNNs. Additionally, it is noteworthy that among these three devices, only A71 offers access to the device's DSP for DNN inference. As a result, the compute engine sets for each device are defined as follows:

$$\begin{aligned} \mathcal{CE}_{\text{P7}} = \mathcal{CE}_{\text{S20}} &= \{\text{CPU, GPU, NPU}\} \\ \mathcal{CE}_{\text{A71}} &= \{\text{CPU, GPU, NPU, DSP}\} \end{aligned} \tag{3.27}$$

Table 3.5 Target Devices

	Samsung A71	**Samsung S20 FE**	**Google Pixel 7**
Year	2020, January	2020, October	2022, October
SoC	Snapdragon 730	Exynos 990	Tensor G2
CPU	2×2.20 GHz Kryo 470 Gold 6×1.80 GHz Kryo 470 Silver	2×2.73 GHz Exynos M5 2×2.50 GHz Cortex-A76 4×2.00 GHz Cortex-A55	2×2.85 GHz Cortex-X1 2×2.35 GHz Cortex-A76 4×1.80 GHz Cortex-A55
GPU	Adreno 618 @700 MHz	Mali-G77 MP11 @800 MHz	Mali-G710 MP7 @850 MHz
NPU	Qualcomm Hexagon 688	✓	Tensor Processing Unit
RAM	6 GB @1866 MHz	6 GB @2750 MHz	8 GB @3200 MHz
TDP	5 W	9 W	7 W

3.5.4 Profiling Details

This section outlines the available configuration options for each compute engine, *op*(*ce*), within the context of an execution plan's tunable hardware parameters, which are employed by `CARIn` during the profiling phase of the device-specific objective functions. For CPUs, the configuration includes the ability to adjust the number of threads used for multithreading and the option to leverage the XNNPACK delegate, which functions as a back-end for CPU execution, leveraging the XNNPACK library to provide highly optimized implementations for 32- and 16-bit floating-point computations, as well as symmetrically quantized DNN operations. Since all the devices under consideration are equipped with 8 CPU cores, the set of tunable options can be defined as follows:

$$op(\text{CPU}) = \{N_{\text{threads}}, \text{XNNPACK}\} \tag{3.28}$$

where $N_{\text{threads}} \in \{1, 2, 4, 8\}$ and XNNPACK $\in$ {TRUE, FALSE}, resulting in 8 distinct CPU execution combinations. On the other hand, for GPUs and NPUs, `CARIn` exclusively employs fp16 arithmetic when feasible, as it offers reduced latency without compromising accuracy:

$$op(\text{GPU}) = op(\text{NPU}) = \{\text{precision} = \text{fp16}\} \tag{3.29}$$

Lastly, it should be noted that the DSP does not expose any configurable parameters, and thus its set of options can be defined as an empty set:

$$op(\text{DSP}) = \{\,\} \tag{3.30}$$

Regarding the profiling process, each execution configuration begins with five warm-up runs to stabilize the target processor's performance and minimize variability. Following this, 100 executions are performed to obtain statistically significant measurements of latency and energy consumption. To ensure consistent device temperatures and mitigate the risk of overheating, a 2-minute idle period is introduced before initiating the next set of runs.

3.6 Results

This section presents the outcomes of `CARIn`'s comprehensive evaluation. The findings offer valuable insights into the framework's effectiveness in addressing challenges related to device heterogeneity and runtime fluctuations across both single- and multi-DNN scenarios, while simultaneously ensuring compliance with predetermined SLOs.

3.6.1 Designs

The initial assessment evaluates the performance of CARIn's designs including single-processor execution and processor combinations for single- and multi-DNN applications, respectively.

3.6.1.1 Comparison Methods

To comprehensively assess CARIn's performance in comparison to existing methodologies, three simple and experience-based baselines are utilized, along with a comparison to OODIn. The baseline methods are formulated based on empirical observations to establish a fundamental performance level, providing a reference point for setting minimum performance expectations in real-world applications.

- **Single-architecture baseline**: The effectiveness of CARIn is contrasted with the traditional approach of considering a single model architecture, even if it is also accompanied by its quantized versions. This paradigm typically revolves around the selection of the model with the highest accuracy, optimal memory efficiency, most compact size, and other relevant criteria.
- **Transferred baseline**: To assess the extent to which CARIn addresses device heterogeneity, the transferred baseline is utilized, where the MOO problem is solved on a specific device, and the resultant designs are then applied to different devices. This baseline, being device-agnostic, overlooks the inherent characteristics and limitations of individual devices.
- **Multi-DNN-unaware baseline**: The third baseline assesses the efficacy of the proposed framework in handling concurrent model executions, particularly its capability to generate optimal model-to-processor mappings for multi-DNN workloads. The multi-DNN-unaware baseline dissects a multi-DNN MOO problem into M single-DNN uncorrelated problems, solves each one independently and then combines the solutions.
- **OODIn**: In OODIn, the weighted sum method was employed as a technique for addressing MOO problems. More precisely, OODIn aims to maximize the weighted sum obtained from the normalized objective functions. This approach fails to account for the inherent scale discrepancies among the diverse objective functions, particularly evident in DL metrics. While the utilization of assigned weights may potentially mitigate this limitation, it necessitates prior knowledge of the statistical characteristics of the functions involved. When dealing with multi-DNN configurations, OODIn would operate as the multi-DNN-unaware baseline presented above, differing only in its utilization of the weighted sum method instead of optimalities.

3.6.1.2 Single-DNN Execution

Figures 3.3 and 3.4 delineate the benefits of CARIn in relation to the optimality metric for the two single-DNN use cases. The evaluation includes a comparison against two single-architecture baselines, specifically using the model with the highest accuracy (best accuracy, B-A) and the model with the smallest size (best size, B-S), the transferred baselines from the other two devices, collectively designated as T_{A71}, T_{S20} and T_{P7}, and OODIn. The initial designs d_0 for each device are prominently indicated, affirming the presence of device heterogeneity. Patterned bars in the figures highlight instances where certain baselines fail to yield a solution due to non-compliance with the problem's constraints (denoted by !) or inapplicability to different devices (denoted by N/A).

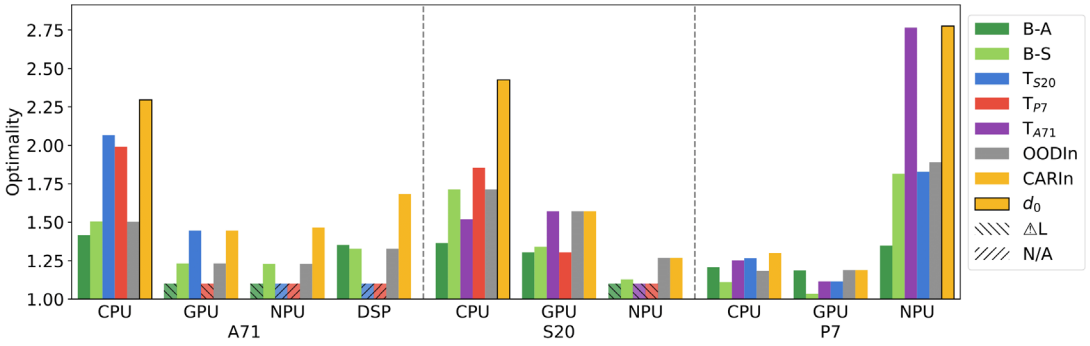

Figure 3.3 UC1 evaluation

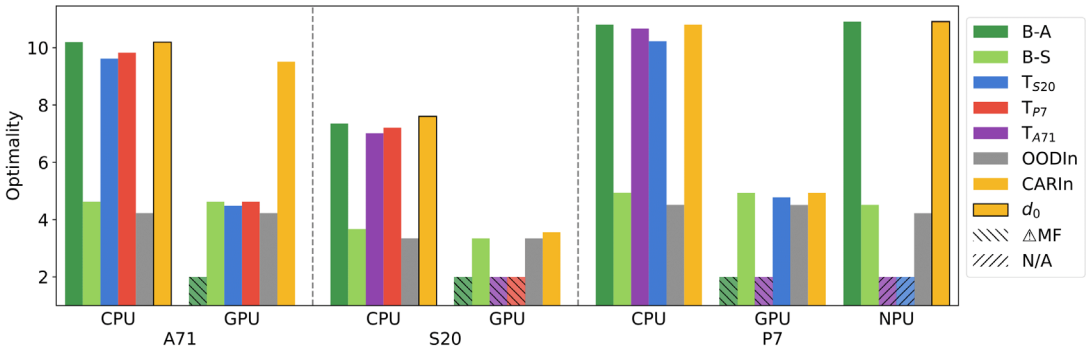

Figure 3.4 UC2 evaluation

Takeaways. *The proposed framework achieves a substantial improvement, with an average gain of 1.19× and 1.57× (up to 1.46× and 1.92×) over the B-A and B-S baselines, respectively. It is noteworthy that these baselines, primarily designed for SOO problems, prove inadequate in capturing the multi-objective nature inherent in DL applications. Regarding the transferred baselines,* ***CARIn*** *achieves an average improvement of 1.17× in optimality (up to 1.84×). Importantly, it not only enhances overall optimality, but also exhibits improvements across all considered objective functions. Specifically, for UC1, there is an observed average increase of 0.156 units in accuracy along with a 32.7% improvement in throughput. Similarly, for UC2, the results indicate an average reduction of 2.8 MB in model size and a 19.9% reduction in latency, all while maintaining the same accuracy level. Compared to* ***OODIn****, an optimality increase of 1.5× is achieved in average (up to 1.99×).*

3.6.1.3 Multi-DNN Execution

Figures 3.5 and 3.6 show the benefits of `CARIn` concerning the optimality metric in the context of the two multi-DNN use cases. In these scenarios, the evaluation includes comparisons against the multi-DNN-unaware baseline, the transferred baselines from other devices, and `OODIn`. The horizontal axis illustrates combinations of processors for each device. In the case of UC3, all possible combinations are presented, whereas for UC4, given the substantial number of possible combinations, the results are organized and presented based on optimality, highlighting the top five configurations for each device.

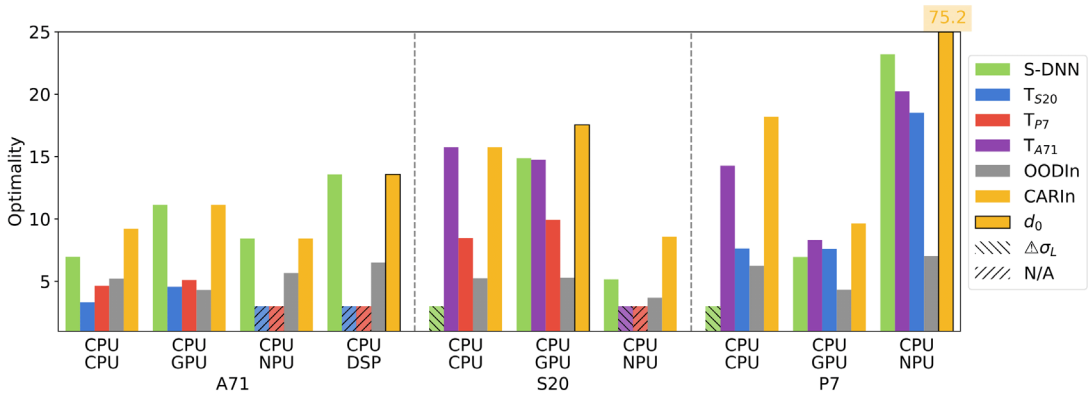

Figure 3.5 UC3 evaluation

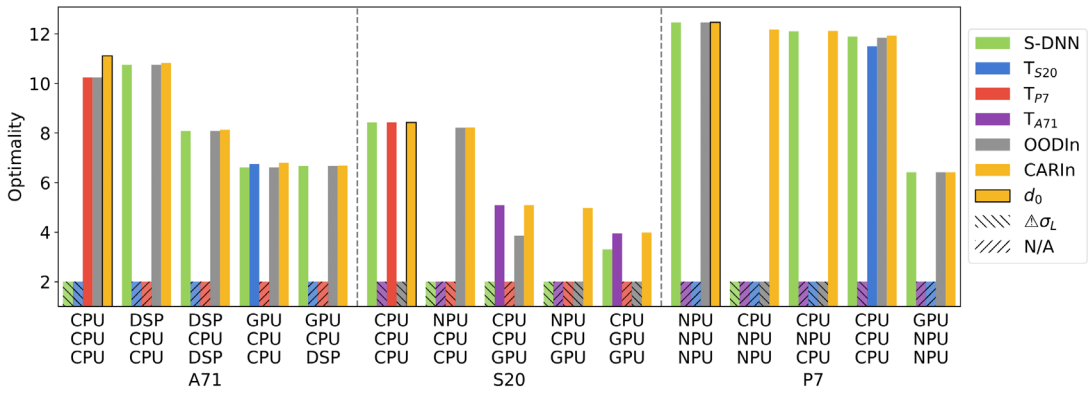

Figure 3.6 UC4 evaluation

Takeaways. *In the context of UC3,* `CARIn` *delivers a significant average optimality improvement of 1.47× across devices (up to 3.24×) over the multi-DNN-unaware baseline and an even more substantial gain of 1.87× (up to 4.06×) over the transferred baselines. Notably, these enhancements extend across all specified objectives. Compared to* `OODIn`*, there is a 2.83× improvement in optimality (up to 10.69×). Meanwhile, UC4 poses a distinctive challenge, where the majority of baselines struggle to produce a viable solution, primarily due to their inability to satisfy the stringent latency constraints inherent in this use case, underscoring the intricacy of UC4. It is noteworthy that, given the utilization of a singular model per task, instances where baselines do not fail result in performance parity with* `CARIn`*, emphasizing the importance of accommodating a diverse array of models for each task.*

3.6.2 Runtime Adaptation

This section evaluates the responsiveness of the Runtime Manager (RM) and its effective utilization of designs generated by RASS to dynamically adapt to runtime fluctuations. The assessment focuses on the UC1 single-DNN scenario on the S20 and the UC3 multi-DNN scenario on the A71.

3.6.2.1 Single-DNN Execution

Table 3.6 presents the selected designs and switching policy, while Figure 3.7 depicts the behavior of RM in the single-DNN scenario. The initial design for UC1 on S20, d_0, involves the utilization of EfficientNet Lite0 FFX8 on the CPU with 4 threads and the enabled XNNPACK library, resulting in 75.11% accuracy and a 16 MB memory footprint. As the CPU gradually becomes overloaded, the throughput experiences a decline until RM identifies an alternative design as the current highest-performing solution. The new configuration, d_1, entails

the use of EfficientNet Lite0 FP16 on the GPU. Following further inferences, RM triggers another switch due to an impending memory issue. In this instance, RASS has identified the memory-efficient design, d_m, to involve the device's CPU.

Table 3.6 Selected Designs and SP for the Single-DNN UC1 Scenario on S20

c_{CPU}	c_{GPU}	c_{NPU}	c_m	d_{new}
F	-	-	F	$d_0 = \langle$EfficientNet Lite0 FFX8, $CPU_{4,T}\rangle$
T	F	-	F	$d_1 = \langle$EfficientNet Lite0 FP16, GPU$\rangle$
T	T	F	F	$d_2 = \langle$MobileNet V2 1.4 FP16, NPU$\rangle$
T	T	T	F	$d_w = \langle$MobileNet V2 1.0 FX8, $CPU_{4,T}\rangle$
T	T	T	T	$d_{wm} \equiv d_w$
-	-	-	T	$d_m = \langle$EfficientNet Lite0 FX8, $CPU_{8,F}\rangle$

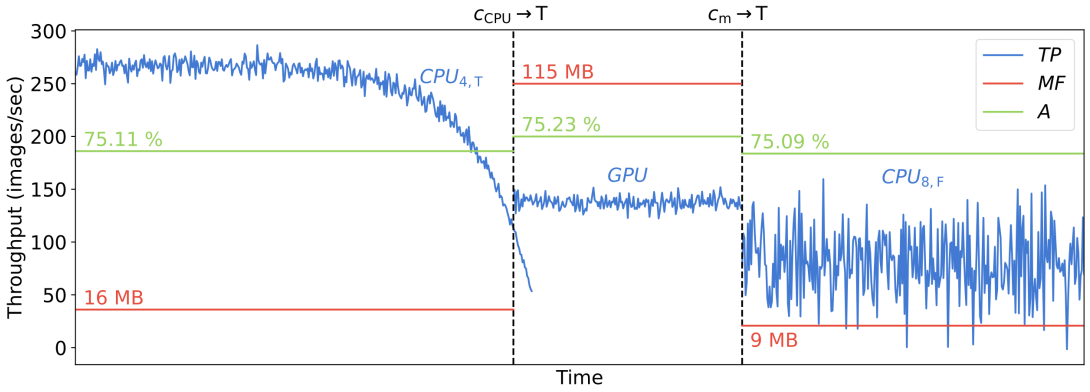

Figure 3.7 CARIn's runtime behavior targeting the single-DNN UC1 scenario on S20

Takeaways. *It is worth highlighting that despite modifications in the execution plan, CARIn consistently upholds accuracy levels, even when employing the memory-efficient design. This steadfast commitment to preserving user Quality of Experience (QoE) underscores CARIn's resilience in the face of dynamic alterations.*

3.6.2.2 Multi-DNN Execution

Table 3.7 and Figure 3.8 correspond to the multi-DNN scenario. In the context of UC3, where two models with distinct workloads are employed, CARIn recognizes the heavier workload associated with the second task and acknowledges that this specific task is primarily responsible for triggering the switching mechanisms. The figure illustrates the average latency, standard deviation of latency, and accuracy for the second task, as well as the combined memory footprint of both models. UC3 involves the processing of audio data, introducing the potential use of the device's DSP for data capture and processing. Given the likelihood of DSP overload during DNN inference, suppose that the highest-performing GPU-based design, d_1, is currently employed with EfficientNet Lite2 FX8. However, due to the impending threat of a memory issue arising from this design's memory footprint, RM opts to switch to the memory-efficient design, d_m, resulting in a saving of 92 MB of RAM. Subsequently, as RM observes a reduction in DSP overload, it triggers a switch to the highest-performing design, d_0, characterized by lower latency and reduced memory requirements. In the event of a potential DSP overload resurgence, RM strategically avoids reverting to the GPU-based design to mitigate previous concerns of excessive memory usage. Instead, it selects the next design in line, transferring the second model to the CPU while maintaining accuracy levels.

Table 3.7 Selected Designs and SP for the Multi-DNN UC3 Scenario on A71

c_{DSP}	c_{GPU}	c_{CPU}	c_m	d_{new}
F	-	-	F	$d_0 = \{\langle$YAMNet FP16, $CPU_{2,F}\rangle, \langle$EfficientNet Lite2 FFX8, DSP$\rangle\}$
T	F	-	F	$d_1 = \{\langle$YAMNet FP16, $CPU_{2,F}\rangle, \langle$EfficientNet Lite2 FX8, GPU$\rangle\}$
T	T	F	F	$d_2 = \{\langle$YAMNet FP16, $CPU_{4,F}\rangle, \langle$EfficientNet Lite2 FFX8, $CPU_{1,T}\rangle\}$
T	T	T	F	$d_w = \{\langle$YAMNet DR8, $CPU_{2,F}\rangle, \langle$EfficientNet Lite0 FFX8, $CPU_{4,F}\rangle\}$
T	T	T	T	$d_{wm} \equiv d_w$
-	-	-	T	$d_m = d_w$

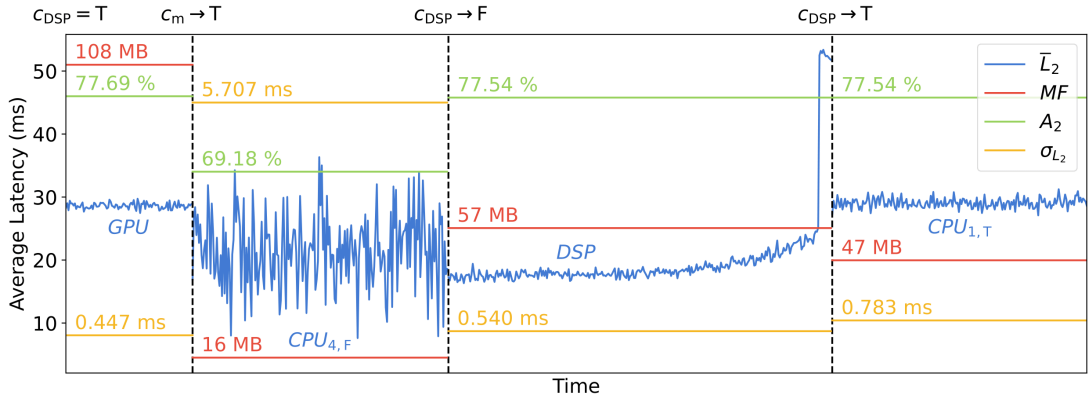

Figure 3.8 CARIn's runtime behavior targeting the multi-DNN UC3 scenario on A71

Takeaways. *It is important to acknowledge that CARIn may not always maintain predefined metric levels. As demonstrated in this instance, transitioning to the memory-efficient design resulted in an 8.5% decrease in accuracy and an increase in jitter. However, such occurrences are considered temporary states of urgency, with a firm expectation that they will be swiftly rectified, thereby minimizing impact on user QoE. Notably, the rise in average latency or the standard deviation of latency does not significantly affect user QoE, as these metrics already meet the specified latency constraints, which precede the optimization of the objectives.*

3.6.2.3 Comparison with OODIn

In OODIn, the model/processor switching technique was introduced to mitigate runtime fluctuations. However, OODIn lacks the ability to predict forthcoming changes in resource availability, so upon detecting such events, the MOO problem necessitates readjustment to the new conditions and subsequent re-solving to determine the new highest-performing solution. CARIn offers the advantage of solving the specified MOO problem once, prior to application initiation, thus switching to a new execution plan during runtime happens instantaneously and is based on the predetermined designs and switching policy. Table 3.8 presents the average and maximum observed solution times of OODIn across diverse applications and devices. Given that this time is inevitable whenever a runtime issue occurs, it has the potential to become a bottleneck, thus impeding the seamless execution of a DL application. The solution time primarily hinges on the number of objectives and the dimensionality of the decision space $\mathcal{X}$, contingent upon the number of DL tasks, utilized models per task, compression techniques, and adjustable system parameters. Given that the time required for the TFLite interpreter to load a model on the CPU is typically around 3-4 ms, it becomes evident that revisiting the MOO problem can potentially become a bottleneck for the application, impacting the user's QoE.

Table 3.8 OODIn's Solving Time in Milliseconds

Decision Space Dimension	A71		S20		P7	
	Average	Maximum	Average	Maximum	Average	Maximum
500	1.45	2.12	0.55	1.55	3.64	7.99
2000	2.80	5.94	1.70	3.04	4.94	9.38
5000	6.56	10.46	4.98	15.97	7.06	1.09
10000	12.14	16.07	11.09	34.25	10.41	13.38

Aside from the time overhead incurred by repeated problem solving, OODIn also requires constant access to the entire array of considered models, necessitating their storage on the user's device, which can impose limitations on the assortment of models and compression techniques initially considered. CARIn obviates the necessity to store all model variants, requiring only those selected by RASS. Table 3.9 elucidates this contrast in terms of model storage requirements for every examined use case.

Table 3.9 Storage Requirements of CARIn and OODIn in MB

	A71			S20			P7		
	CARIn	OODIn	Reduction	CARIn	OODIn	Reduction	CARIn	OODIn	Reduction
UC1	13.18	276.36	19.98×	34.37	443.10	12.89×	34.19	443.10	12.96×
UC2	48.64	311.45	6.40×	40.98	311.45	7.60×	52.96	311.45	5.88×
UC3	25.74	205.22	7.97×	58.70	205.22	3.50×	52.81	205.22	3.89×
UC4	2.65	6.56	2.48×	3.95	6.56	1.66×	3.95	6.56	1.66×

3.7 Limitations and Future Directions

Despite the challenges addressed by CARIn, certain limitations persist that hinder its performance in practical deployment scenarios. First, as mentioned in Subsection 3.3.2, the computation of device-dependent metrics associated with objective functions or constraints across all candidate solutions is unsuitable for realistic mobile applications due to its substantial time requirements and the necessity of deploying entire models onto target devices, particularly within expansive decision spaces. Within the broader landscape of related studies, numerous works have harnessed performance prediction methodologies to estimate such metrics when executing DNNs on specific hardware platforms, without resorting to direct measurements. These models consider a range of inputs, encompassing (a) architectural characteristics of the DNN model such as network topology, layer configurations, and overall complexity, (b) hardware specifications including compute architecture, memory hierarchy, interconnectivity, and support for parallelism, and (c) environmental parameters like batch size, input data characteristics, runtime conditions, and temperature/power conditions. Such approaches are orthogonal to the proposed framework and can be integrated within CARIn to provide a more expedient alternative to exhaustive profiling. In the future, the exploration of such methods is envisioned to furnish a comprehensive assessment of the framework's performance and suitability for real-world scenarios.

An additional limitation arises from the selection of models for evaluation. In the contemporary landscape of generative AI, the inclusion of generative models, such as autoregressive language models, becomes paramount. These models, characterized by their ability to generate outputs sequentially based on previously generated tokens, impose heightened demands, particularly within the context of mobile environments. Therefore, it is

imperative to account for such intricacies when assessing the efficacy of AI frameworks intended for deployment in resource-constrained settings.

3.8 Conclusion

This research underscores the paramount significance of optimizing the on-device execution of DNNs to meet the evolving demands of today's AI applications. The presented framework, `CARIn`, aims to spearhead progress in this direction. While the challenges of device heterogeneity, runtime adaptation, and multi-DNN execution persist, `CARIn` provides a novel and comprehensive solution towards alleviating them. The integration of an expressive multi-objective optimization framework and the introduction of RASS as a runtime-aware MOO solver manage to enable efficient adaptation to dynamic conditions while adhering to user-specified SLOs. RASS stands out for its ability to foresee upcoming runtime issues and generate a set of configurations which enable rapid, low-overhead adjustments in response to environmental fluctuations.

4 Deploying Transformer-Based Models on Mobile Devices

Over the past several years, convolutional neural networks have been the cornerstone of computer vision, consistently delivering state-of-the-art accuracy across a wide range of tasks. Their dominance in the field is attributed to their ability to effectively capture spatial hierarchies in visual data, making them exceptionally well-suited for applications such as image classification, object detection, and segmentation. Researchers have extensively studied and refined CNN architectures to improve their performance and efficiency, particularly for resource-constrained environments like mobile and embedded systems. These advancements have paved the way for efficient inference of computer vision applications directly on edge devices such as smartphones, drones, and IoT cameras.

The introduction of BERT (Bidirectional Encoder Representations from Transformers) in 2018 by researchers at Google [29] marked a significant turning point in the field of NLP. Built on the original Transformer architecture [28], BERT revolutionized NLP by leveraging bidirectional context, meaning it could consider both the left and right context of a word in a sentence during training. This was a major architectural shift compared to earlier approaches like RNNs or LSTM networks, which processed input sequentially and were limited in their ability to model long-range dependencies efficiently. BERT's introduction led to a paradigm shift in NLP tasks [114], [115], such as language modeling, machine translation, and question answering, where it set new benchmarks for accuracy. Beyond NLP, the Transformer architecture soon found applications in other domains. In computer vision, models like Vision Transformers (ViTs) [33] demonstrated that Transformers could outperform traditional CNNs on image recognition tasks. Similarly, Transformers have been adopted in speech recognition [116], where they are used to model audio sequences, and in drug discovery, where they excel at understanding molecular structures and predicting chemical properties. The rise of LLMs like GPT, Claude, Llama, and Gemini, has further cemented the dominance of Transformers. These large models, trained on massive datasets and containing billions—sometimes even trillions—of parameters, have demonstrated unprecedented versatility, enabling applications ranging from conversational AI to code generation and content creation.

However, despite their flexibility and success, Transformers are computationally intensive, requiring significant resources for both training and inference. Training large Transformer models often demands high-performance hardware, such as GPUs or TPUs, and substantial energy consumption. The inference phase, though less demanding than training, still requires considerable computational power to handle real-time processing, especially when deploying models at scale. This resource-intensive nature has so far limited the deployment of large Transformer-based models to centralized cloud environments, where server farms can provide the necessary computational infrastructure [117]. While this approach supports many applications effectively, it creates challenges for scenarios requiring low-latency responses, privacy-preserving computations, or autonomous operations in edge devices.

The deployment of Transformers in resource-constrained environments is still a relatively underexplored area. CNNs have been widely adopted in edge computing due to their successful adaptation to hardware constraints. Through years of development, both model-specific techniques, such as depthwise separable convolutions [61], and model-agnostic techniques like

model pruning, quantization, and knowledge distillation have emerged as key innovations, reducing the computational complexity and memory footprint of CNNs without significantly compromising accuracy. In contrast, Transformers are inherently more challenging to deploy on edge devices. Their significantly larger model sizes, driven by the large number of parameters required for effective performance, make them more demanding in terms of memory and computational power. Additionally, the self-attention mechanism that defines Transformer models, while highly powerful, involves complex matrix operations that are computationally expensive and difficult to optimize for energy efficiency in hardware with limited capabilities.

This contrast raises an important research question: can the well-established practices and findings that have made CNNs viable for on-device inference be applied or adapted to Transformers? While both architectures today serve as foundational building blocks for deep learning, their fundamental differences necessitate a re-examination of traditional optimization strategies. To address this question, it is crucial to first explore the current state of on-device Transformer execution. Understanding the limitations and opportunities in deploying these models on resource-constrained hardware enables the identification of pathways for innovation, bridging the gap between research and practical deployment.

4.1 Related Work

The on-device execution and evaluation of traditional Transformer models for NLP has only recently begun to attract research attention. Existing studies, such as [118] and [119], offer limited insights, as they typically examine only a small subset of Transformer models, neglect on-device accuracy evaluation, and overlook the applicability of compression techniques. Furthermore, these works fail to comprehensively evaluate all possible accelerators and execution configurations available on modern mobile devices, leaving significant gaps in understanding how to optimize these models for on-device deployment.

In contrast, there has been extensive research focused on benchmarking Vision Transformers and comparing their performance with CNNs for vision tasks on mobile devices [120], [121]. These efforts have provided valuable insights into the suitability and efficiency of Vision Transformers in mobile contexts. However, given the existing body of work, Vision Transformers fall beyond the scope of this study. Instead, the focus is exclusively on traditional NLP Transformer models, addressing critical gaps left by previous research and aiming to establish a more comprehensive evaluation framework for their on-device deployment.

Additionally, this study does not include comparisons with RNNs, which were the predominant models for NLP tasks prior to the advent of Transformers. Unlike CNNs, which can be optimized effectively for mobile applications, RNNs are inherently challenging to optimize for on-device execution due to their sequential nature of processing, which limits parallelization. Furthermore, RNNs lack robust support in many modern libraries and frameworks tailored for mobile environments, further diminishing their practicality for on-device inference. For these reasons, this work focuses on evaluating Transformer models, which represent the current state-of-the-art for NLP, while excluding RNNs from direct comparison.

4.2 Experimental Methodology

The evaluation strategy involves constructing a comprehensive benchmark suite of Transformer models, which will then be executed on two distinct Android mobile devices to investigate their performance across varying configurations. This approach enables the assessment of key factors such as latency, accuracy, resource consumption, and the impact of quantization schemes on model efficiency. For the evaluation infrastructure, TensorFlow (v2.11.0) was

selected due to its robust range of quantization techniques and its compatibility with mobile deployment via TensorFlow Lite (TFLite). Specifically, the nightly builds of TFLite were utilized for the deployment process.

The benchmark models were sourced from Hugging Face's Hub through the Transformers library (v4.27.4) as pre-trained versions. Table 4.1 outlines the key architectural parameters of the Transformer model that are most relevant to the focus of this research. In this context, particular emphasis is placed on the parameters that influence the execution speed and resource utilization of the model during the inference stage, as this directly impacts the efficiency and responsiveness of Transformer models when deployed in real-world, resource-constrained environments. While other parameters, such as dropout rate, regularization strength, and weight initialization, are also critical to the model's performance, they do not affect the model's operational efficiency and are therefore outside the scope of this study.

Table 4.1 Transformer Architectural Parameters

Parameter	**Description**
Batch Size (*B*)	Number of input samples to be processed simultaneously
Sequence Length (*S*)	Maximum number of tokens in one input sample
Embedding Size (*E*)	Width of a token's embedding
Hidden Size (*H*)	Dimensionality of the model's internal vector space
Attention Heads (*A*)	Number of parallel heads for the attention mechanism
Intermediate Size (*I*)	Width of the FFN Network
Layers (*L*)	Number of layers in the model

4.2.1 Tasks and Models

Table 4.2 provides a detailed overview of the benchmarked Transformer models, including their key architectural parameters, workload characteristics, model size, and accuracy metrics. The performance of these models is evaluated on two distinct NLP tasks, which were selected to provide a comprehensive assessment of the models' capabilities in handling various types of language-based challenges, thereby gauging their effectiveness in real-world applications.

Table 4.2 Transformer Models

Task	**Embeddings**			**Encoder**						**Overall Model**		**Accuracy (%)**			
	Vocab	***E***	**#Params**	**Name**	***L***	***A***	***H***	***I***	**#Params**	**FLOPs**	**#Params**	**FP32**	**FP16**	**DR8**	**FFX8**
	50265	24	1.21M	BERT Tiny	6	2	24	16	0.02 M	3.53 M	1.23 M	90.05	90.05	90.05	89.85
	30522	128	3.91 M	ELECTRA Tiny	6	2	24	16	0.03 M	4.19 M	3.94 M	90.25	90.30	90.15	90.50
	30522	256	7.81 M	XDistil-L6-H256	6	8	256	1024	4.75 M	0.49 G	12.57 M	93.20	93.25	93.00	30.95
Sequence	30522	384	11.72 M	MiniLM-L3	3	12	384	1536	5.35 M	0.54 G	17.07 M	93.25	93.25	93.00	91.95
Classification	30522	128	3.91 M	MobileBERT	24	4	128	512	20.42 M	2.06 G	24.33 M	93.30	93.30	93.10	92.95
on Emotions	30522	384	11.72 M	XDistil-L6-H384	6	12	384	1536	10.67 M	1.09 G	22.39 M	93.35	93.35	93.10	82.30
Dataset	30522	384	11.72 M	MiniLM-L12	12	12	384	1536	21.32 M	2.18 G	33.04 M	93.45	93.45	92.65	78.40
	28996	512	14.85 M	RoBERTa Tiny	4	8	512	2048	12.64 M	1.28 G	27.49 M	93.50	93.50	93.40	92.30
	30522	128	3.91 M	ELECTRA S	12	4	256	1024	9.55 M	0.98 G	13.46 M	93.55	93.55	93.40	92.25
	30522	768	23.44 M	DistilBERT	6	12	768	3072	43.16 M	4.30 G	66.60 M	94.50	94.50	94.55	77.85
Text	50257	768	38.60 M	DistilGPT2	6	12	768	3072	42.69 M	10.47 G	81.29 M	46.85	-	-	-
Generation	50257	768	38.60 M	GPT2 Small	12	12	768	3072	85.28 M	15.99 G	123.88 M	51.41	51.41	-	-

For the downstream classification task, the Emotions dataset [108] was employed, which comprises a collection of English Twitter messages labeled with six distinct emotion categories. The dataset is split into 16000 samples for training, 2000 for validation, and 2000 for testing. The reported accuracy is based on the top-1 accuracy metric, evaluated on the test set of the dataset. After performing statistical analysis on the sample lengths, the input token sequence

length was standardized to 50 tokens across all models. Regarding training, the majority of the models were initially pre-trained on large datasets, such as Reddit comments and 2ORC citation pairs, and were then fine-tuned on the Emotions dataset. Notable exceptions include DistilBERT [65], for which the fine-tuned version on Emotions was directly obtained from Hugging Face, and BERT Tiny and ELECTRA Tiny, which were trained from scratch. The training and fine-tuning configuration consisted of a batch size of 64, using either the Adam or RMSprop optimizer with weight decay and an initial learning rate of 10^{-4}, which was subject to exponential decay over time. The models selected for this evaluation include optimized architectures such as XtremeDistil [109], MiniLM [122], and MobileBERT [110], as well as lighter versions of the original RoBERTa [115] and ELECTRA [123] models.

In addition to the ten discriminative models, the analysis also includes GPT2 [124], used for text generation, in both its small and distilled versions. Both the input sequence and output prediction lengths are fixed at 64 tokens. To examine the effects of distillation and FP16 quantization, accuracy is reported in the context of the next-token prediction task using the LAMBADA test set, which consists of 5000 passages. Generative models generally require a large number of parameters to produce high-quality outputs, making their integration into mobile devices a challenging task. Although GPT2 Small and DistilGPT2 represent some of the most compact generative models currently available, their performance remains moderate, and their model sizes are still prohibitive for truly mobile-friendly applications.

4.2.2 Mobile Devices

To accommodate a range of device capabilities, two smartphones were selected for evaluation: Samsung Galaxy A71, representing the mid-tier category, and Samsung Galaxy S20 FE, representing the high-end category of modern mobile devices. These two models offer distinct processing capabilities, enabling a comprehensive comparison across devices with varying hardware specifications. A summary of their key characteristics, including information on processing power, memory, and other relevant hardware features, is provided in Table 4.3.

Table 4.3 Target Smartphones

	Samsung A71	**Samsung S20 FE**
Year	2020, January	2020, October
SoC	Snapdragon 730	Exynos 990
CPU	2×2.2 GHz Kryo 470 Gold 6×1.8 GHz Kryo 470 Silver	2×2.73 GHz Exynos M5 2×2.5 GHz Cortex-A76 4×2 GHz Cortex-A55
GPU	Adreno 618 @700 MHz	Mali-G77 MP11 @800 MHz
NPU	Qualcomm Hexagon 688	✓
RAM	6 GB @1866 MHz	6 GB @2750 MHz
TDP	5 W	9 W

These devices were chosen to reflect a broad spectrum of hardware configurations, from the mid-range Snapdragon 730 and its Qualcomm Hexagon NPU in the A71, to the more powerful Exynos 990 SoC with integrated NPU in the S20 FE. This range allows for an in-depth exploration of how Transformer models perform across devices with varying computational power, memory bandwidth, and thermal dissipation. Evaluating both mid-tier and high-end models provides insights into how varying hardware configurations impact the execution of DL models, shedding light on the trade-offs that developers must consider when targeting a wide range of mobile devices.

The Qualcomm Snapdragon 730 SoC in the Samsung Galaxy A71 features the Hexagon Tensor Accelerator (HTA), a dedicated, scalable, and power-efficient hardware accelerator specifically designed for fixed-point deep convolutional neural networks. It is part of the Hexagon DSP and optimized for tasks involving lower precision arithmetic. In contrast, the Exynos 990 SoC in the Samsung Galaxy S20 FE is equipped with a more versatile NPU that supports both fixed-point and floating-point operations. This NPU is accessed via the Exynos Deep Neural Network (EDEN) interface, providing flexibility for executing a broader range of DL models with varying precision requirements. These distinct NPUs are tailored to different types of workloads, impacting their compatibility with the quantization schemes of the models evaluated.

4.2.2.1 Delegates

By default, inference tasks are carried out on the device's CPU. However, to leverage the specialized capabilities of other processing units, computations can be offloaded to one or more accelerators. Many DL libraries and frameworks facilitate this offloading process through the use of delegates. A delegate is a software component that manages the delegation of part or all of the neural network's computation graph to a designated accelerator. This delegation ensures that the workload is executed in the most efficient manner possible, optimizing both performance and energy consumption.

In the context of Android and TFLite, several delegates are available to optimize inference. As of the time of this research, the available delegates include:

- **XNNPACK**: This delegate leverages the XNNPACK library, which provides highly optimized implementations of neural network operators using 32- and 16-bit floating-point arithmetic, as well as symmetrically quantized operators. It serves as a back-end for CPU execution, offering significant performance improvements, with latency speedups of up to 2.3× for floating-point operations and 1.3× for quantized operations. XNNPACK is particularly effective for models that are deployed on CPUs and is a widely used option for improving inference speed.
- **GPU**: The GPU delegate is designed to offload computations to the device's graphics processing unit, which operates using 16-bit or 32-bit floating-point numbers. While the delegate supports quantized models, the computations are performed using floating-point precision, ensuring compatibility with a broader range of models. This delegate is particularly beneficial for models requiring high parallelism and processing power, making it an attractive option for many modern mobile devices with powerful GPUs.
- **NNAPI**: The Neural Networks API is a hardware acceleration API developed by Google to facilitate the use of specialized accelerators available on Android devices. NNAPI dynamically selects the most suitable accelerator for a given model based on factors such as the model's architecture and the types of operations involved. However, the performance of NNAPI can be variable, as it is influenced by factors such as the specific hardware available on the device, the current load on the hardware, and the model's requirements.
- **HEX**: The Hexagon delegate is an experimental option designed to accelerate models that comply with TensorFlow's 8-bit symmetric quantization specification. It targets Android devices equipped with Qualcomm's Hexagon DSP. The HEX delegate enables the use of quantized models to achieve faster and more efficient inference on compatible devices, making it an important tool for models deployed on devices with limited computational resources.

By carefully selecting the appropriate delegate based on the target hardware and model characteristics, developers can significantly improve the performance of DL models on Android devices, optimizing both speed and energy efficiency. Quantization plays a critical role in this process. When a model's quantization scheme is not compatible with the target delegate, or if the delegate does not support all the necessary operations for the model, additional computational overheads may emerge. These include the need for (de-)quantization between layers and the potential fallback to the default CPU implementation. Such overheads can result in suboptimal scheduling, degraded overall performance, and a negative impact on the model's accuracy. Therefore, understanding the circumstances under which these overheads arise is essential for optimizing both performance and accuracy in practical applications.

4.2.2.2 Quantization-Delegate Compatibility

Table 4.4 presents the average percentage of nodes in the evaluated models that can be offloaded to each processor via the available delegates for the two target devices. A value of zero indicates that the model could not be executed due to unsupported operations, whereas *N/S* denotes that execution is not supported by default.

Table 4.4 Quantization-Delegate Compatibility

Variant	Samsung A71						Samsung S20 FE		
	CPU	GPU		DSP		NPU	CPU	GPU	NPU
	XNNPACK	GPU	NNAPI	HEX	NNAPI	NNAPI	XNNPACK	GPU	NNAPI
FP32	74.1	99.8	72.6	*N/S*	*N/S*	*N/S*	74.1	99.8	0
FP16	77.6	81.8	0	*N/S*	*N/S*	*N/S*	77.6	81.8	0
DR8	67.6	99.8	71.3	*N/S*	*N/S*	*N/S*	67.6	99.8	2.6
FFX8	64.0	99.8	0	24.9	0	1.4	64.0	99.8	1.1

Among the available accelerator delegates, the GPU is the only one likely to provide acceleration on both devices, as it supports the vast majority of operations commonly used in Transformer models. In contrast, both the DSP and the NPUs are limited in the number of operations they can handle, which results in a high number of model partitions. This limitation is expected, as these NPUs were specifically optimized during their development for CNNs, RNNs, and simple MLPs, which were the predominant workloads at the time. When comparing to EfficientNet-Lite0, a mobile-friendly image classification CNN, the corresponding percentages of operations supported by the delegates exceed 98% for all the supported combinations listed in Table 4.4.

4.2.3 Benchmarking Details

To benchmark model performance on the target devices, a custom Android application was developed with a lightweight and straightforward user interface designed to simulate real-world usage scenarios. For CPU performance measurements, the evaluation included testing the XNNPACK library or multithreading, by adjusting the number of threads from 1 up to the device's total number of CPU cores. Since the XNNPACK library already utilizes multithreading to enhance execution, using additional multithreading with the XNNPACK delegate is not recommended. For the GPU and NNAPI delegates, 16-bit computation was used wherever applicable.

Before conducting any measurements, each model was executed 1–5 times to allow the processor to warm up and reduce measurement variability. Following this, latency and memory footprint were recorded. To maintain a stable device temperature and prevent overheating, the number of inference runs was adjusted based on model size, with larger models limited to

around 20 runs and smaller models tested up to 100 runs. Additionally, a 2–3-minute idle period was enforced between measurements to allow the device to cool down, minimizing thermal effects on performance.

4.3 Results

In this section, the performance of the benchmark models is evaluated on the target devices by measuring key metrics, which provide a comprehensive overview of each model's efficiency and effectiveness in real-world scenarios.

4.3.1 On-Device Accuracy

The accuracy reported in Table 4.2 was obtained through evaluation using Python's TFLite Interpreter running on a system equipped with an Intel® Xeon® CPU. Subsequently, an investigation was conducted to determine whether similar accuracy outcomes could be achieved when utilizing the available delegates and processors on the target devices. The findings reveal that, with the exception of the GPU delegate, all delegate-processor combinations (as detailed in Table 4.4) yield the accuracy values presented in Table 4.2. Notably, MobileBERT is the only model that maintains its original accuracy when executed on the GPU, whereas the accuracy of the other discriminative models experiences a significant decline, dropping to below 35%. This discrepancy can likely be attributed to the architectural distinctions between MobileBERT and the other models. While the nine models exhibiting poor performance on the GPU follow the standard architecture, incorporating the GELU activation function and Layer Normalization layers, MobileBERT substitutes these components with ReLU activations and element-wise linear transformations, which contribute to its better compatibility with the GPU delegate.

4.3.2 CPU Performance

Figures 4.1 and 4.2 illustrate the impact of the XNNPACK delegate and CPU multithreading on the throughput of all FP32 models across the two target devices. Each model's five bars correspond to XNNPACK, 1, 2, 4, and 8 CPU threads when viewed from top to bottom. The optimal number of threads for each model has been highlighted with bolder shading for clarity. The following observations can be made:

- **Although XNNPACK is enabled by default in the TFLite Interpreter** to accelerate inference, it does not necessarily represent the optimal configuration for the majority of models on both devices.
- **There is no universal optimal configuration across models or devices.** For example, on the Samsung S20 FE, the best configuration for ELECTRA Tiny is achieved by enabling the XNNPACK delegate, whereas for RoBERTa Tiny, setting the CPU to 4 threads results in the best latency. On the Samsung A71, instead of using the XNNPACK delegate, the optimal configuration for ELECTRA Tiny is achieved with 2 threads. These findings are further substantiated by the analysis of CPU configurations for the quantized variants of the models, which often yield different optimal configurations compared to their FP32 counterparts. For instance, the best configuration for the FFX8 models on both devices is obtained by enabling the XNNPACK delegate.

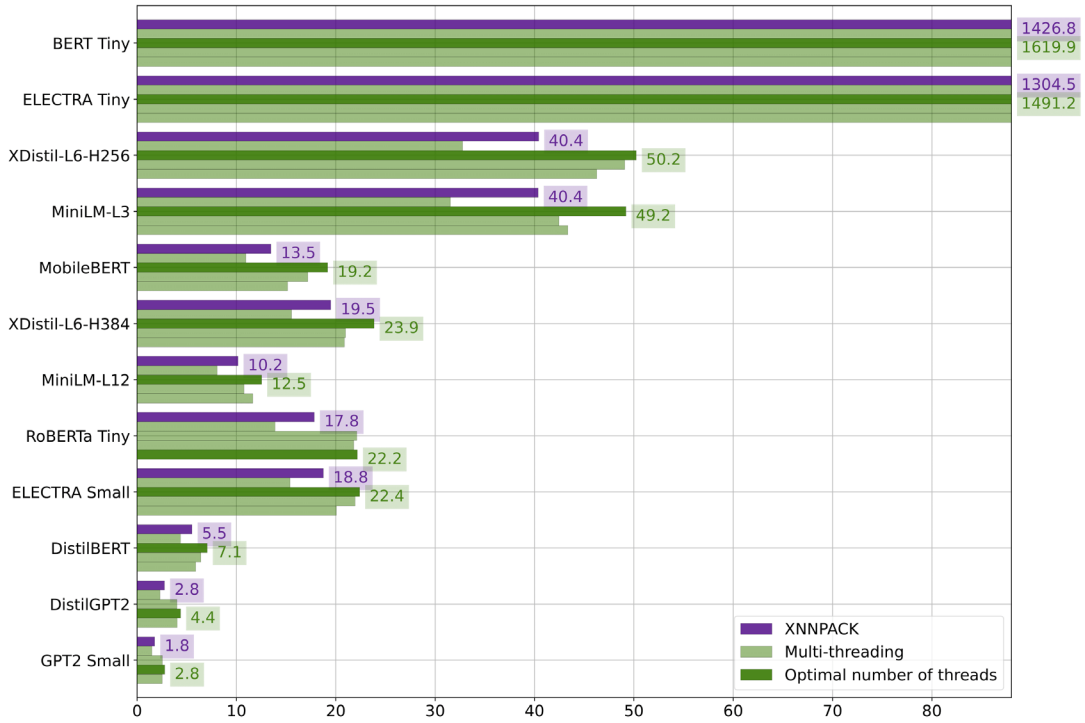

Figure 4.1 CPU throughput for Samsung A71

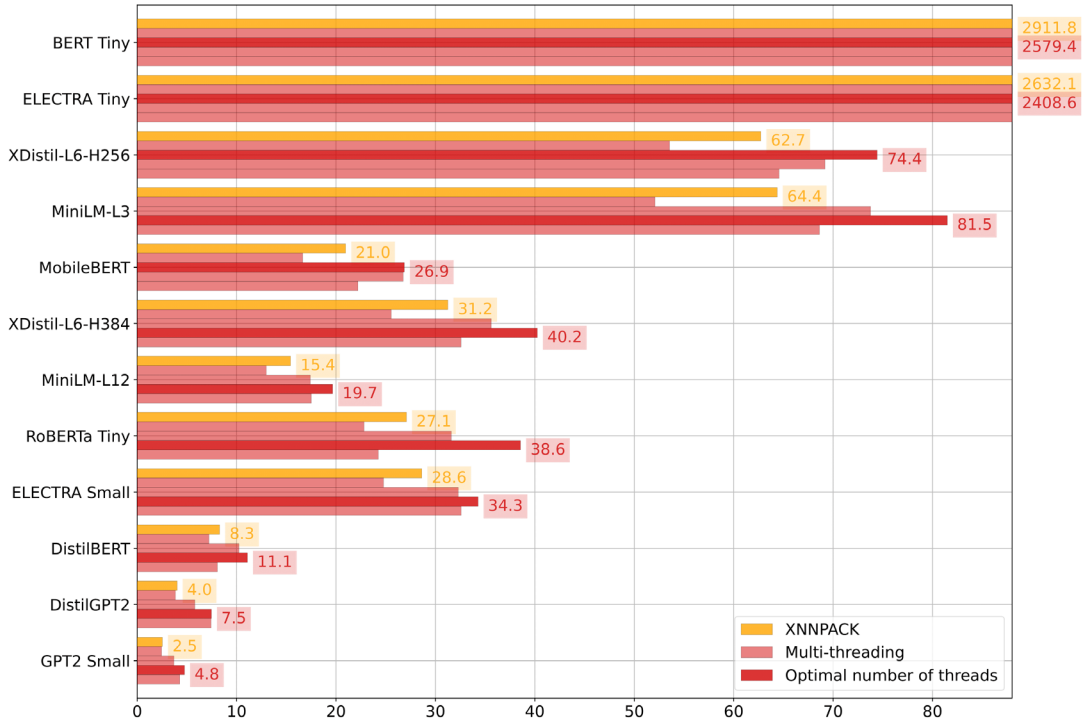

Figure 4.2 CPU throughput for Samsung S20 FE

4.3.2.1 Quantization

Table 4.5 presents the impact of quantization on latency and memory usage, with the memory requirements for each FP32 model provided in MB for comparison. The values are averaged across both devices, as their results were nearly identical. FP16 models exhibit the same latency as FP32 models but require at least 70% more memory. In contrast, integer models accelerate execution and simultaneously reduce memory requirements for the majority of architectures.

Table 4.5 Impact of Quantization on CPU

Model	Latency Speedup			RAM	RAM Reduction		
	FP16	DR8	FFX8	FP32	FP16	DR8	FFX8
BERT Tiny	0.97×	0.97×	0.42×	4.7	0.41×	0.89×	1.28×
ELECTRA Tiny	0.99×	0.99×	0.39×	5.1	0.18×	0.91×	1.13×
XDistil-L6-H256	0.99×	1.62×	1.51×	26.2	0.32×	2.54×	3.03×
MiniLM-L3	1.05×	1.86×	1.64×	30.4	0.28×	2.84×	2.54×
MobileBERT	1.03×	1.72×	1.85×	89.4	0.59×	2.93×	3.16×
XDistil-L6-H384	0.99×	1.82×	1.75×	52.9	0.37×	3.07×	3.27×
MiniLM-L12	1.02×	1.89×	1.77×	95.2	0.47×	3.41×	3.47×
RoBERTa Tiny	0.99×	1.65×	1.81×	65.0	0.37×	3.34×	3.94×
ELECTRA Small	1.01×	1.53×	1.82×	44.7	0.51×	3.01×	3.01×
DistilBERT	1.06×	2.27×	2.32×	191.9	0.46×	3.31×	3.51×
DistilGPT2	-	-	-	585.2	-	-	-
GPT2 Small	0.98×	-	-	673.0	0.59×	-	-

4.3.3 Accelerators

Based on the data presented in Table 4.4, execution on NPUs does not yield significant benefits. Therefore, the analysis focuses on calculating the latency speedup achieved by the GPU and DSP processors for floating-point and fixed-point models, respectively. These speedups are calculated in comparison to the best CPU configuration for each model, based on multithreading and the use of the XNNPACK delegate. Although the GPU delegate generally results in a drop in accuracy for most models, examining GPU acceleration remains valuable due to its potential for performance improvements. Table 4.6 outlines the achieved speedups for each model, with the exception of GPT2, which was found to be incompatible with the available accelerators.

Table 4.6 Accelerator Latency Speedup

Model	Samsung A71		Samsung S20 FE
	GPU	DSP	GPU
BERT Tiny	0.08×	0.44×	0.06×
ELECTRA Tiny	0.09×	0.49×	0.06×
XDistil-L6-H256	0.86×	0.83×	0.90×
MiniLM-L3	0.69×	0.72×	1.06×
MobileBERT	0.97×	0.92×	1.06×
XDistil-L6-H384	1.24×	0.86×	1.20×
MiniLM-L12	1.22×	0.92×	1.29×
RoBERTa Tiny	1.19×	0.87×	1.91×
ELECTRA Small	1.37×	0.85×	1.56×
DistilBERT	1.22×	0.71×	2.77×

As anticipated, the DSP fails to deliver any meaningful acceleration due to its limited compatibility with the models. Consequently, the focus shifts to the GPU for performance evaluation. Note that the use of quantization does not impact the GPU's performance, as all model variants are executed using fp16 arithmetic. This means that the GPU is able to accelerate only floating-point models, whereas integer models benefit more from CPU acceleration, as demonstrated in Table 4.5. Lastly, Table 4.7 provides a summary of the increase in memory

usage for each model when executed on the GPU, highlighting the additional memory requirements incurred by utilizing the GPU for processing.

Table 4.7 Accelerator Memory Increase

Model	Samsung A71 GPU	Samsung S20 FE GPU
BERT Tiny	7.31×	13.41×
ELECTRA Tiny	6.12×	18.16×
XDistil-L6-H256	3.52×	5.65×
MiniLM-L3	3.49×	4.57×
MobileBERT	3.52×	4.46×
XDistil-L6-H384	3.34×	4.09×
MiniLM-L12	3.33×	4.12×
RoBERTa Tiny	2.79×	3.43×
ELECTRA Small	3.63×	4.69×
DistilBERT	2.80×	3.10×

4.4 Discussion and Future Work

In the preceding section, several key observations were made, offering valuable insights into potential optimization strategies for enhancing execution performance. These optimizations can be grouped into two primary categories: system-level optimizations and model-level optimizations. Both levels of optimization are crucial for achieving the best performance outcomes in real-world applications.

4.4.1 System Optimizations

At the system level, improvements are imperative to address the multifaceted challenges outlined in Subsection 2.4.2, which this work confirms are equally relevant for Transformer models. The lack of a universally optimal configuration for Transformers mirrors the broader challenges posed by the heterogeneity of devices, models, and application requirements. This reinforces the understanding that Transformers are subject to the same systemic constraints as other machine and deep learning models, further underscoring the need for targeted optimizations.

Future advancements in this domain should focus on developing robust system-level frameworks capable of dynamically identifying and applying optimal configurations tailored to both model-specific characteristics and application performance objectives [83]. These frameworks should integrate device-specific profiling, real-time resource monitoring, and AI-based decision-making algorithms to intelligently select the most appropriate delegate and configuration. Such an approach would account for the current state of the device, including available resources, competing processes, and application-specific requirements. Given the dynamic and resource-constrained environment of mobile devices, it is essential for these frameworks to support access to a diverse array of processors. This capability would allow for seamless switching between processors in response to significant fluctuations in resource availability, such as changes in battery level, temperature, or processor workload. By dynamically adapting to these conditions, the system could maintain efficient execution without compromising performance or energy efficiency. A unified system designed to address these challenges was proposed in Chapter 3, showcasing a comprehensive solution to improve the

adaptability and efficiency of on-device inference in real-world, resource-constrained environments.

Furthermore, enabling processor co-execution—where computational tasks are intelligently distributed across multiple processors simultaneously—represents a promising avenue for maximizing resource utilization. This strategy could help balance workloads, reduce bottlenecks, and ensure that the full computational potential of the device is leveraged. By combining these innovations, future frameworks could significantly enhance the efficiency, scalability, and adaptability of DL workloads on mobile devices, particularly for computationally intensive models like Transformers.

4.4.2 Model Optimizations

The analysis presented in this chapter revealed that Transformers exhibit limited compatibility with hardware accelerators. While GPUs can offer marginal speedups, they significantly degrade the accuracy of most models. Furthermore, DSPs and NPUs were found to have minimal compatibility with Transformers, rendering them largely ineffective for these architectures. This section investigates the underlying causes of the observed accuracy loss when executing Transformer models on GPUs, as well as the accuracy degradation noted in half of the FFX8 models in Table 4.2. To address these issues, inspiration is drawn from the MobileBERT architecture, which was designed with efficiency and compatibility in mind. Specifically, two model-level architectural substitutions are considered to mitigate these problems:

1. **Replacement of the GELU activation function with ReLU (R1):** The GELU activation function, commonly used in standard Transformer architectures, despite its smooth approximation properties, introduces computational overhead and is less compatible with hardware accelerators due to its reliance on non-linear approximations. The traditional ReLU, in contrast, is simpler, widely tested and supported across processors, and has been shown to maintain competitive accuracy in many neural network architectures.
2. **Replacement of Layer Normalization with Batch Normalization (R2):** Layer Normalization calculates statistics (mean and variance) independently for each individual data sample in a batch, normalizing across the feature dimension. This requires recalculating the statistics for every input during both training and inference and involves operations such as the reciprocal of the square root, which can be computationally intensive and less efficient on hardware accelerators. In contrast, Batch Normalization relies on moving averages of the mean and variance computed during training, allowing these precomputed statistics to be applied directly during inference. As a result, it is more computationally efficient at runtime, avoiding per-sample recalculations and enabling faster execution on many devices.

These substitutions are not arbitrary but are grounded in addressing the specific inefficiencies and incompatibilities identified in the evaluation. By systematically testing these modifications, the goal is to determine whether they provide a more generalizable solution to optimize Transformers for mobile devices, going beyond the scope of MobileBERT and offering broader insights into model optimization in resource-constrained settings. Table 4.8 illustrates the impact of the proposed substitutions on accuracy and latency for two representative Transformer models: XDistil-L6-H256, which is incompatible with integer quantization, and ELECTRA Small. The reported speedup values are averaged across both target devices, as the results exhibited minimal variation between them.

Table 4.8 Model Optimizations

Model		On-Device Accuracy								Average Speedup					
		Original		R1		R2		R1+R2		R1		R2		R1+R2	
		CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
XDistil-L6-H256	FP32	93.20	34.75	93.45	34.75	93.05	93.05	**93.35**	**93.35**	1.47×	1.03×	1.18×	1.07×	**1.49×**	**1.08×**
	FP16	93.25	34.75	93.45	13.75	93.05	93.05	**93.35**	**93.35**	1.52×		1.20×		**1.54×**	
	DR8	92.95	34.75	93.25	13.75	92.90	92.90	**93.15**	**93.15**	1.87×		1.29×		**1.97×**	
	FFX8	30.00	34.75	88.20	34.75	92.95	93.05	**93.20**	**93.30**	1.01×		2.47×		**2.50×**	
ELECTRA S	FP32	93.55	13.75	93.50	13.75	**92.55**	**92.50**	92.15	92.20	1.67×	1.03×	1.20×	1.09×	**1.75×**	**1.12×**
	FP16	93.55	13.75	93.50	13.85	**92.55**	**92.55**	92.15	92.10	1.70×		1.20×		**1.75×**	
	DR8	93.10	13.75	93.50	13.85	**92.15**	**92.20**	92.10	92.05	2.30×		1.26×		**2.44×**	
	FFX8	92.45	3.30	93.40	34.75	**92.55**	**92.35**	91.95	92.05	1.01×		2.58×		**2.64×**	

- **Substitution R1 (GELU ⟶ ReLU):** Improves execution speed for all model variants on the CPU except FFX8. However, R1 does not address the accuracy issues previously observed, particularly on the GPU. This suggests that the computational efficiency gained through R1 does not translate into improvements in numerical stability or compatibility with hardware accelerators.
- **Substitution R2 (Layer Normalization ⟶ Batch Normalization):** Resolves the accuracy issues, while also reducing latency on the CPU. For FFX8 variants, R2 achieves an average speedup of 2.5×, whereas for the remaining variants, it delivers a 1.2× speedup. These results highlight the dual benefits of R2 in enhancing execution efficiency and maintaining model fidelity. This improvement is likely due to the hardware-friendly nature of Batch Normalization, which avoids the computationally intensive operations required by Layer Normalization.
- **Combined application of R1 and R2:** Achieves the best accuracy for XDistil-L6-H256, demonstrating the potential for these substitutions to address model-specific hardware compatibility challenges. However, it slightly reduces the accuracy of ELECTRA Small by 1.4%. These findings indicate that the effectiveness of each optimization is model-dependent and highlights the importance of tailoring substitutions to the specific requirements and constraints of a given model and hardware configuration. This nuanced understanding can guide future efforts to optimize Transformer models for resource-constrained environments.

The optimizations discussed above primarily enable GPU utilization (through R2) while also improving CPU latency. However, they offer no measurable benefits in terms of memory efficiency or performance on other accelerators. Focusing specifically on GPU execution, the observed speedups are minimal, averaging only 1.1×. This limited improvement stems from the inherent characteristics of TFLite Transformer models, where approximately half of the operations are reshape operations. These operations, which adjust the shape or arrangement of a tensor without modifying its underlying data, are computationally lightweight on CPUs but become relatively costly on GPUs. This disparity arises because GPUs are optimized for highly parallelizable mathematical computations, such as matrix multiplications, rather than the less computationally intensive reshape operations. Consequently, the unchanged proportion of reshape operations in the optimized models constrains the achievable performance gains on the GPU. These findings underscore a fundamental limitation in the structure of TFLite Transformer models: their operational mix is not fully aligned with the strengths of GPU architecture. Addressing this imbalance may require further architectural modifications or a more granular approach to delegate selection, for instance by focusing on offloading only the most GPU-efficient operations while keeping reshape operations on the CPU.

Future work could explore several avenues to further improve the performance of Transformer models on mobile devices, building upon the proposed model-level optimizations and enhancing their generalizability. Some potential directions include:

1. **Alternative activation functions and dynamic normalization:** While replacing GELU with ReLU improved execution time, future research could investigate other activation functions (*e.g.*, Swish, Leaky ReLU, or mobile-optimized GELU variants) that balance speed and accuracy. In parallel, dynamic or hybrid normalization techniques, such as Instance Normalization or Group Normalization, could be explored as alternatives to Batch Normalization, potentially offering better adaptation to mobile constraints and diverse input patterns.
2. **Pruning, sparsity, and quantization:** Further investigation into pruning redundant or low-importance parameters could reduce memory usage and speed up inference, especially if combined with adaptive, Transformer-specific pruning strategies. Additionally, advancing quantization techniques—such as hybrid precision approaches where select layers operate at reduced precision—could yield further efficiency gains without significant accuracy loss.
3. **Hardware-specific optimizations:** Optimizing Transformer architectures for specific mobile accelerators (*e.g.*, NPUs, DSPs, or Apple's Neural Engine) could involve redesigning model components or leveraging hardware-aware layer implementations to better align with the computational characteristics of target devices.
4. **Expanded evaluation across model types and tasks:** Future work should extend the evaluation of these optimizations beyond the current focus, incorporating a broader range of Transformer architectures—including autoregressive models—and testing them across diverse datasets and real-world applications to assess generalizability and robustness.

By advancing these areas, future research could lead to significant improvements in the deployment of Transformer models on mobile devices, enhancing both performance and efficiency while maintaining high accuracy.

4.5 Conclusion

This study proves that the well-established practices and insights related to the on-device execution of CNNs, including the application of various quantization schemes, are not directly transferable to Transformer models. To support this assertion, a comprehensive investigation was conducted into the current state of on-device Transformer inference. This investigation encompassed benchmarking a diverse range of Transformer models, assessing their compatibility with various mobile processors, validating their on-device accuracy across different execution configurations, and evaluating the efficacy of multiple quantization techniques. By uncovering the unique challenges associated with deploying Transformers on resource-constrained mobile devices, this work provides actionable insights and outlines potential avenues for optimization, paving the way for improved performance and broader applicability of Transformers in real-world mobile environments. The key contributions of this work are as follows:

1. **A comprehensive benchmark suite and software infrastructure**: A benchmark suite is introduced, encompassing a diverse set of Transformer models for NLP tasks, along with the necessary software infrastructure to facilitate their evaluation. This

infrastructure is designed to enable systematic, reproducible, and regulated testing of on-device inference performance across multiple devices and execution configurations.

2. **A detailed examination of on-device Transformer inference**: Through an extensive evaluation, this work provides the first comprehensive insights into the state of on-device Transformer inference. This includes analyzing performance metrics such as latency, throughput, and memory usage, while also examining the impact of different processors on model accuracy. The analysis reveals critical limitations, such as the poor compatibility of Transformers with certain mobile processors (*e.g.*, DSPs and NPUs) and the lack of robust support for integer quantization in many architectures.
3. **Insights into future optimizations**: Opportunities for improving the hardware compatibility and computational efficiency of Transformer models on mobile devices are identified. For instance, architectural modifications are highlighted, such as replacing computationally intensive components like GELU and Layer Normalization with simpler alternatives that can enhance compatibility. Furthermore, quantization techniques and their trade-offs are explored, demonstrating how to balance reduced computational requirements with maintained accuracy.

In conclusion, the benchmarking results indicate that despite the growing use of dedicated hardware in modern devices, general-purpose hardware such as the CPU continues to be heavily utilized due to its versatility in supporting diverse workloads and adapting to new software infrastructures. The findings further emphasize the necessity of device- and model-specific system-level optimizations, as default configurations (*e.g.*, XNNPACK) are rarely the most efficient. Moreover, the results demonstrate that with minimal model optimizations, only the GPU can be effectively leveraged for on-device execution, whereas other specialized accelerators remain largely underutilized. These insights point to the need for (a) continued research on optimizing Transformer models for mobile environments, and (b) the advancement of mobile accelerators designed specifically to accommodate Transformer architectures.

5 Advancing Early Intrusion Detection for the Internet of Things

The Internet of Things has revolutionized digital ecosystems by enabling interconnected devices to collect, process, and exchange data in real time across domains such as smart homes, healthcare, and industrial automation. At its core, IoT relies on smart objects—embedded systems with sensing, processing, and communication capabilities—that seamlessly integrate the physical and digital worlds [125]. These devices generate diverse types of data, including sensor readings, control signals, video streams, and network traffic logs, often operating autonomously with minimal human intervention.

Embedded systems are the core of IoT devices. Every IoT device is essentially an embedded system with networking capabilities, allowing it to communicate and process data in real time. However, as discussed in Subsections 2.2.3 and 2.3.2.2, these devices typically operate under significant constraints, including limited computational power, memory capacity, energy availability, and storage space. Due to these restricted capabilities, there is a pressing need for highly optimized hardware architectures, lightweight software stacks, and efficient data-processing models. Consequently, developers and researchers focus on creating resource-efficient operating systems, specialized networking protocols, and streamlined AI models specifically tailored to the demands of embedded IoT environments.

The widespread adoption of IoT devices has introduced significant security vulnerabilities, largely due to their limited computational resources, weak authentication mechanisms, and frequent exposure to untrusted networks. These vulnerabilities make IoT ecosystems prime targets for cyberattacks, ranging from malware infections and distributed denial-of-service (DDoS) attacks to more sophisticated man-in-the-middle (MitM) exploits and adversarial manipulation of data streams. Ensuring the security of IoT systems is therefore critical [126], as security breaches can lead to data leaks, service disruptions, and potential physical harm in safety-critical applications. More importantly, the long-term success of the IoT paradigm hinges on user trust; without confidence in the security and reliability of the underlying infrastructure, consumers, industries, and institutions are unlikely to adopt IoT technologies at scale.

To mitigate such security risks, intrusion detection systems (IDS) have been widely employed. IDS solutions monitor network traffic or host activity to detect malicious behavior and potential intrusions. In recent years, ML and, more prominently, DL, have become integral to the development of IDS technologies, driven by their ability to detect complex attack patterns that traditional rule-based systems fail to recognize. Unlike conventional IDS, which rely on manually crafted signatures or predefined heuristics, DL-based IDS can learn from data, enabling the detection of previously unseen threats and zero-day attacks. Particularly in the context of IoT security, where network traffic is highly dynamic and attacks can evolve rapidly, DL models such as CNNs [127], RNNs [128], Transformers [129], and hybrid architectures [130] have been increasingly employed to analyze network flows, uncover anomalies, and improve detection accuracy.

A key challenge in intrusion detection is the need for real-time threat detection to minimize response times and mitigate the impact of attacks. This has led to the emergence of early intrusion detection systems (EIDS) [131], which aim to classify and detect intrusions as early as possible within a network session. Recent DL advancements highlight Transformers'

effectiveness in sequential data processing, particularly in NLP [132]. However, unlike text, network traffic flows are time series, where packet arrival times convey critical contextual information often overlooked by existing IDS. To address this limitation, a Transformer-based approach is proposed, incorporating novel positional encodings that integrate packet timestamps. This enhancement enables the model to capture both sequence structure and temporal dynamics, leading to improved intrusion detection. The key contributions of this work include:

- **The design and implementation of A-THENA**, a Transformer-based early intrusion detection system with novel time-aware positional encoding mechanisms for rapid and lightweight attack detection.
- **The development of a data augmentation pipeline for network traffic** that improves model robustness and generalization.
- **The introduction of a custom training loss function** that enhances earliness and improves performance.
- **An empirical evaluation of the system's real-world applicability**, demonstrating its computational efficiency and suitability for deployment on resource-constrained IoT devices.

5.1 Preliminaries and Related Work

Recent advancements in deep learning have significantly improved the capabilities of IDS, enabling more accurate and adaptive threat detection. However, challenges remain in achieving efficient and early detection, particularly in resource-constrained environments such as IoT networks. This section reviews existing research on DL-based IDS, augmentation techniques for cybersecurity, and Transformer positional encodings, highlighting their strengths, limitations, and the gaps the proposed approach aims to address.

5.1.1 DL-Based Intrusion Detection

AI-enabled IDS have gained significant traction in recent years, with most approaches relying on feature engineering [133], [134], [135], [136], [137]. In these methods, domain-specific features are extracted from raw network traffic and used as model inputs, typically encompassing flow-level statistical metrics such as packet count, byte count, flow duration, and inter-arrival times. While feature engineering has proven effective, the extraction process is often computationally expensive and time-consuming, posing a constraint for real-time detection, particularly in environments with limited computational resources. Furthermore, predefined feature sets may fail to capture complex temporal dependencies and subtle attack signatures, reducing the system's ability to detect novel and evolving threats.

To overcome these challenges, recent studies have investigated DL-based IDS that process raw network traffic directly, eliminating the need for handcrafted features. These approaches leverage architectures such as convolutional neural networks [138], [139], [140], recurrent neural networks [141], Transformers [142], and hybrid models [143], [144], [145], among others. By learning hierarchical and sequential representations from packet streams, these models enhance detection accuracy and improve adaptability to new attack patterns.

Despite recent advancements, achieving early and efficient threat identification in resource-constrained environments remains a considerable challenge. Although some studies have explored training models on variable-length network flows [146], [147], [148], [149], a critical aspect of early threat detection, these approaches often overlook the temporal dynamics of packet flows, which are key to accurately characterizing attacker behavior. To bridge this gap,

the proposed method incorporates packet timestamps into the detection process leveraging novel dynamic temporal positional encodings within Transformer-based architectures.

5.1.2 Augmentation Strategies for Cybersecurity Datasets

The availability of real-world cybersecurity datasets is limited, with most existing datasets containing a restricted number of attack sessions, resulting in an insufficient sample size for effective model training. Furthermore, these datasets often include a diverse range of attack types, whose variability and heterogeneity contribute to substantial class imbalances, as certain attack categories may be significantly underrepresented compared to those that generate higher volumes of traffic. To address these challenges, augmentation techniques are essential for enhancing dataset diversity and mitigating class imbalance. By synthetically increasing the number of samples, data augmentation improves model generalization, enabling IDS to effectively identify both prevalent and rare attack patterns.

Several recent studies have investigated augmentation methods to enhance dataset variability and address class imbalance. However, the proposed models in these studies primarily operate on extracted features rather than raw network traffic, leading to the adoption of augmentation techniques designed for tabular data. These approaches include the Synthetic Minority Oversampling Technique (SMOTE) [150], [151], [152], Conditional Tabular Generative Adversarial Networks (CTGANs) [151], [153], Variational Autoencoders (VAEs) [152], [154], and Transformer-based generative models [155], all of which have demonstrated effectiveness in tabular data augmentation. More recently, diffusion models have been explored for raw network traffic generation [156], [157]; however, their application in attack scenario generation remains unexplored.

5.1.3 Transformer Positional Encodings

Unlike recurrent models, which inherently capture sequential order through their structure, Transformer models rely on a fully parallelized self-attention mechanism, which does not incorporate any intrinsic notion of token order. As a result, Transformers require an explicit method to encode the positions of tokens in a sequence, so that the model can learn meaningful representations that depend on the order of these tokens. To address this limitation, positional encodings are introduced to provide information about the relative or absolute positions of tokens within the sequence. By incorporating positional information, the model can capture sequential dependencies while maintaining the parallelization advantages of the Transformer architecture, unlike recurrent models that process data sequentially.

In this work, two categories of positional encodings are considered: (a) input positional encodings, which are directly added to the input embeddings, and (b) attention positional encodings, which are applied to the query and key matrices before the self-attention computation. Each category serves a distinct purpose in encoding positional information, influencing how the model attends to different parts of the input sequence. Below, three widely used positional encodings are described: sinusoidal and Fourier-based encodings, which belong to the first category, and the rotary positional encoding, which falls into the second.

5.1.3.1 Sinusoidal Positional Encoding

The sinusoidal positional encoding was introduced in the original Transformer architecture [28] as a method to inject position-dependent information into the model without relying on learned embeddings. These encodings are computed using sine and cosine functions of varying frequencies, ensuring that each position has a unique representation while also allowing the model to generalize to unseen sequence lengths. Given a sequence of length n, the sinusoidal

positional encoding assigns each position $p \in \{0, 1, \ldots, n\text{-}1\}$ a vector of dimension d_{m}, where d_{m} denotes the hidden dimension of the Transformer model. The encoding is computed using the following formulas:

$$\begin{aligned} PE(p, 2i) &= \sin\left(\frac{p}{10000^{2i/d_{\mathrm{m}}}}\right) \\ PE(p, 2i+1) &= \cos\left(\frac{p}{10000^{2i/d_{\mathrm{m}}}}\right) \end{aligned} \tag{5.1}$$

where i = 0, 1, ..., d_{m}/2-1 indexes the sine and cosine components within the encoding dimension. The choice of the base 10000 ensures a smooth distribution of frequencies across the encoding space, allowing the model to capture both fine-grained and long-range positional dependencies while maintaining numerical stability during training.

5.1.3.2 Fourier-Based Positional Encoding

Since the introduction of the sinusoidal encoding, several alternative methods have been proposed to enhance the effectiveness of positional information in Transformers. The Fourier-based positional encoding [158] extends the idea of the sinusoidal encoding by leveraging a more general Fourier feature mapping. Instead of using a fixed base, this encoding is derived from a learnable frequency basis that enables richer and more flexible positional representations. This approach has been shown to improve the generalization of positional information in various applications. The Fourier positional encoding at position p is given by:

$$\begin{aligned} PE(p, 2i) &= \sin\left(2\pi f_i\, p\right) \\ PE(p, 2i+1) &= \cos\left(2\pi f_i\, p\right) \end{aligned} \tag{5.2}$$

where f_i is the learnable frequency parameter associated with the i-th sine-cosine pair in the encoding.

5.1.3.3 Rotary Positional Encoding

Another widely adopted method for infusing positional information in Transformer architectures is the rotary positional encoding (RoPE) [159]. By rotating representations within a high-dimensional space, RoPE directly incorporates positional cues into the self-attention mechanism. This rotation-based formulation inherently supports relative position modeling, making it well-suited for tasks requiring long-range dependency capture. In its standard form, RoPE applies a rotation matrix to both the query and key vectors in the self-attention mechanism. Let $\mathbf{x} = (x_0, x_1, \ldots, x_{d_{\mathrm{m}}\text{-}1})$ be a packet embedding. For each position p and index i, the corresponding subvector, consisting of two consecutive embedding elements (x_{2i}, x_{2i+1}), is transformed as follows:

$$\begin{bmatrix} x_{2i}^{\mathrm{rot}} \\ x_{2i+1}^{\mathrm{rot}} \end{bmatrix} = \begin{bmatrix} \cos(p\theta_i) & -\sin(p\theta_i) \\ \sin(p\theta_i) & \cos(p\theta_i) \end{bmatrix} \begin{bmatrix} x_{2i} \\ x_{2i+1} \end{bmatrix} \tag{5.3}$$

where θ_i is the rotational angle defined as:

$$\theta_i = 10000^{-2i/d_{\mathrm{m}}} \tag{5.4}$$

5.1.3.4 Time-Aware Positional Encodings

Some recent studies have explored the utilization of timestamps from sequential data to generate time-aware positional encodings, enhancing the ability of models to capture temporal dependencies. However, many of these approaches introduce considerable computational

complexity [160], [161], [162], making them unsuitable for scenarios requiring lightweight and efficient models. Additionally, existing works primarily focus on extending the sinusoidal positional encoding [163], [164], [165], limiting their applicability to alternative encoding strategies. This study is the first to apply time-aware positional encodings to network traffic data and systematically evaluate their impact across three distinct encoding mechanisms.

5.2 Proposed Approach

Network traffic can be conceptualized as a collection of flows traversing network elements. The term "flow" has multiple definitions within the internet community. According to RFC 7011 [166], a traffic flow is a set of packets or frames passing through an observation point over a specified time interval. For instance, in a host-based IDS, the observation point is the potential victim, such as an IoT device vulnerable to attacks. All packets within a flow share common attributes, with one of the most widely accepted flow definitions being the 5-tuple representation: source and destination IP addresses, source and destination transport layer ports, and the protocol in use.

Early intrusion detection in the context of a flow-level IDS refers to the capability of accurately classifying a network flow as benign or malicious as early as possible, using only a partial sequence of packets within the flow. The goal is to minimize the time and data required for threat identification, enabling rapid response and mitigation before an attack fully unfolds. An effective early detection system balances classification accuracy with earliness, ensuring that malicious activities are identified promptly while minimizing false positives and computational overhead. In order to facilitate early and accurate detection, the proposed approach prioritizes four key objectives:

- **Feature-free learning:** Unlike traditional methods that rely on handcrafted feature engineering, the system processes raw packet data without requiring manual feature extraction, reducing preprocessing overhead.
- **Computational efficiency:** The model is designed to be lightweight, ensuring low latency and minimal memory footprint, making it suitable for deployment on resource-constrained environments such as IoT and edge devices.
- **Rapid threat detection:** By classifying network flows using only a small fraction of their packets, the system enables early response and mitigation, preventing attacks from fully unfolding.
- **Adaptability to variable-length flows:** The detection mechanism is designed to handle flows of varying durations without requiring fixed-length representations, ensuring flexibility in real-world traffic analysis.

Figure 5.1 illustrates the proposed system for early intrusion detection, comprising three main stages:

1. **The *Data Preparation* stage** (Subsection 5.2.1), which is common to both training and inference, involves processing labelled PCAP files or raw network traffic through a dedicated pipeline consisting of the Flow Identification, Packet Filtering, and Packet Preprocessing modules. This stage ensures that data are structured appropriately for subsequent analysis.
2. **In the *Training & Evaluation* stage** (Subsection 5.2.2), the processed data are first split into training and test sets. The training data undergo basic offline augmentation to enhance diversity and boost early detection, followed by on-the-fly augmentation,

which includes a series of techniques targeted to network data, during each training step. The final selected model is then evaluated on the test set.

3. **The *Inference* stage** employs the trained model for real-time monitoring of network traffic. The evaluation results from the training phase inform the performance of the system during deployment.

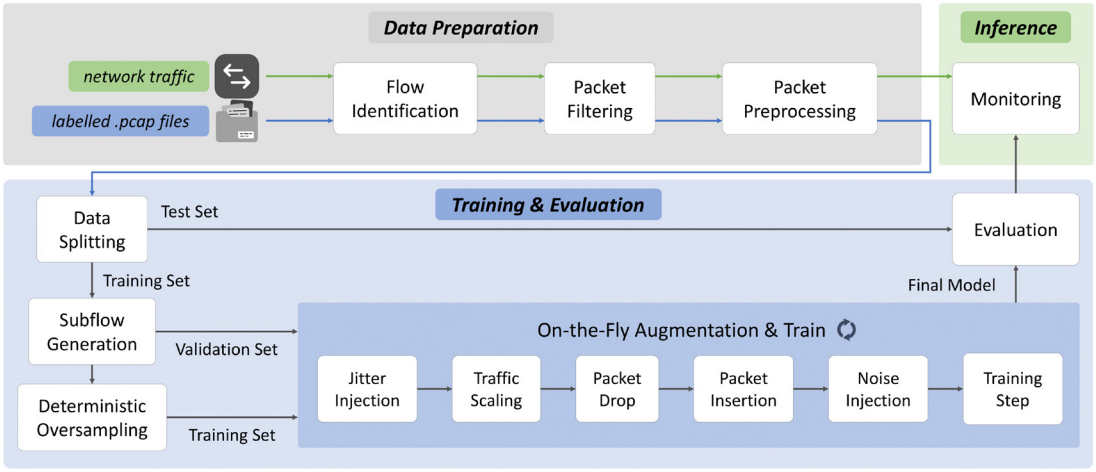

Figure 5.1 The proposed A-THENA system

5.2.1 Data Preparation

The goal in converting raw network traffic into a format suitable for DNN training is to maximize computational efficiency and enable rapid processing. As previously noted, to achieve this, the proposed system departs from traditional methods that rely on extracting predefined features and instead utilizes raw packet bytes as direct input to the model. This approach offers several advantages. First and most importantly, it reduces preprocessing overhead, as it eliminates the computational cost and additional latency associated with feature extraction, making it more efficient, fast and scalable. Moreover, this methodology eliminates the need for manual feature engineering, which is often constrained by domain expertise and may introduce biases or overlook critical patterns. Lastly, this method preserves fine-grained information that might otherwise be lost in the aggregation or summarization process inherent to traditional feature extraction techniques.

5.2.1.1 Flow Identification

The initial step in the proposed system involves maintaining a record of all active flows within the network, regardless of whether they originate from network captures (PCAP files) or real-time network traffic. Upon the arrival or departure of a new packet, the system determines whether the packet corresponds to an existing flow or necessitates the creation of a new flow instance. Formally, a flow F can be represented as an ordered sequence of packets:

$$F = \{P_1, P_2, \ldots, P_n\}, P_i \in \mathbb{R}^d \tag{5.5}$$

where P_i is the i-th packet, n is the length of the flow and d is the length of a packet in number of bytes. Each flow can have a maximum length N, therefore, the number of packets in any given flow satisfies the condition:

$$1 \leq n \leq N \tag{5.6}$$

To construct network flows from raw traffic, a well-defined methodology must be established. Network attacks exhibit diverse characteristics, behaviors, and patterns. For the

conventional 5-tuple definition—which consists of the source IP, destination IP, source port, destination port, and protocol—to be effective in attack detection, these parameters must remain consistent throughout the entire attack session. For instance, brute force attacks (*e.g.*, SSH, FTP, or RDP brute forcing) involve repeated login attempts to a specific service, maintaining a stable 5-tuple across multiple authentication requests. However, many sophisticated cyber threats do not adhere to a consistent 5-tuple structure, rendering detection based solely on this definition inadequate. Examples include distributed attacks, where IP addresses frequently change, and scanning attacks or port-hopping techniques, which result in a high number of ephemeral, often two-packet flows, if classified using the 5-tuple. These limitations prevent an effective representation of the ongoing attack.

To overcome this challenge, the proposed system initially employs the 5-tuple definition, which is appropriate for most single-session attacks, while concurrently monitoring the number of active flows. If this number surpasses a predefined threshold—determined as a multiple of the normal traffic baseline—the system dynamically adjusts its flow aggregation strategy. Specifically, flows are first grouped by ignoring transport layer ports, and if the volume remains excessive, flows are further aggregated by disregarding source IP addresses. Once traffic levels return to normal, the system reverts to standard 5-tuple tracking. This adaptive approach ensures efficient flow management, maintaining lightweight tracking under normal conditions while dynamically expanding flow definitions only when necessary, thereby enhancing attack detection capabilities.

5.2.1.2 Packet Filtering

The second essential step in data preparation involves isolating network packets relevant to the threats targeted by the IDS. By filtering out irrelevant packets before further analysis, the system reduces computational complexity and directs its focus toward potentially malicious traffic. Certain network protocols, such as HTTP, ARP, and ICMP, are commonly exploited for attacks, making their targeted analysis beneficial for intrusion detection. For example, HTTP traffic is frequently associated with web-based threats, including cross-site scripting (XSS), SQL injection, and denial-of-service (DoS) attacks. By selectively filtering HTTP packets, the system can analyze the requests and responses exchanged between clients and web servers, facilitating the detection of suspicious activity.

5.2.1.3 Packet Preprocessing

The final step is essential for converting raw network packet data into a structured format suitable for DL models. Packet preprocessing removes extraneous information and standardizes packets into a fixed-sized array to ensure consistency. This process begins by discarding irrelevant data: the entire Ethernet header is removed, as it contains metadata unrelated to the packet payload, and the IP source and destination addresses are excluded from the IP header to prevent potential model overfitting to specific address patterns. These elements are typically not essential for detecting network intrusions, as the focus is on the payload and the behavior of the protocols in question. Next, each packet is either truncated or padded to a fixed length, denoted as d, ensuring uniform input dimensions for the DL model. To further enhance model training stability, normalization is applied by scaling byte values to the range [0, 1] through division by 255. This step mitigates large value disparities in the input data, improving the model's convergence.

Additionally, packet timestamps are extracted to capture inter-arrival times (IAT), which provide critical temporal context. These timestamps are stored as a separate vector, T, serving as a secondary input for the time-aware positional encodings used by the proposed models. The

timestamp of the first packet in a flow is set to 0, and subsequent timestamps represent the absolute time elapsed since the first packet's arrival. This temporal information is vital for recognizing sequential patterns in network traffic and detecting anomalies based on timing deviations.

5.2.2 Training

The training phase of the proposed system is designed to develop DL models capable of early attack detection using raw network packet data. This phase follows the data preparation stage, utilizing the preprocessed packet representations and corresponding timestamp vectors as input.

5.2.2.1 Data Splitting

First, the dataset is divided into training and test sets. This split ensures that the model is trained on a portion of the data and evaluated on unseen samples, providing a reliable measure of its performance and generalization ability. Since Transformers operate on batches of fixed-length sequences, all network flows must be padded to a consistent length before training. To achieve this, each flow in the training and validation sets is padded with zero packets to match the maximum flow length N. This padding process also generates attention masks, which indicate which parts of the input sequence should be attended to and which should be ignored. Specifically, padded positions are assigned a mask value of 0, ensuring they are excluded from the model's attention mechanism, whereas valid positions receive a mask value of 1.

5.2.2.2 Augmentation

The data augmentation process consists of two stages: offline and online augmentation. In the offline stage, subflows are initially generated to enhance early detection capabilities, followed by the application of deterministic oversampling, if necessary, to increase the dataset size. In the online stage, a set of augmentation techniques is applied to each packet prior to each training step, aiming to enhance data variability and improve model generalization.

Subflow Generation. Subflows are created by retaining only the first k packets of each flow in the training dataset, with k ranging from 1 to the total flow length. This approach trains the model to classify flows based on partial information rather than requiring the entire flow. This is crucial for early detection, where identifying threats from initial packets reduces response time. By learning to detect anomalies with incomplete data, the model better simulates real-world scenarios, enhancing its ability to recognize attacks as early as possible. Following subflow generation, 5% of the training samples are reserved for validation.

Deterministic Oversampling. As discussed in Subsection 5.1.2, real-world large-scale network datasets are often limited. To mitigate this, deterministic oversampling is applied after subflow generation as an additional augmentation step. Each sample is duplicated exactly z times within the training dataset, ensuring a balanced class representation. Unlike random oversampling which selects and repeats samples stochastically, this method guarantees a uniform distribution by maintaining a fixed replication count for each sample.

On-the-fly Augmentation. The fixed duplication strategy is justified by the incorporation of random perturbations at each training epoch, ensuring variability and mitigating the risk of overfitting. Although the base samples are duplicated, each instance undergoes a unique transformation during every epoch. Consequently, the model never encounters an identical sample twice, preventing memorization and reinforcing its ability to generalize effectively by learning robust patterns rather than memorizing specific instances.

As such, on-the-fly or epoch-wise augmentation techniques are applied independently on each training sample, ensuring that the model is exposed to a broad range of slightly modified yet realistic network flows. By leveraging packet timestamps as input, timestamp-based augmentation techniques further enhance the variability of the dataset. Five augmentation techniques are performed in the sequence depicted in Figure 5.1, each targeting specific characteristics of the flow.

1. **Jitter Injection.** The first augmentation technique focuses on the timestamps of the packets within a flow. It introduces small, random variations in packet arrival times to simulate the real-world jitter commonly observed in network communications. For each timestamp in the flow, the minimum temporal distance between the previous and next timestamp, denoted as t_{min}, is first calculated. A random perturbation is then added to the timestamp, drawn from the continuous uniform distribution:

$$\mathcal{U}(-0.7 \cdot t_{\text{min}}, 0.7 \cdot t_{\text{min}}) \tag{5.7}$$

   This perturbation helps simulate network conditions such as congestion or packet delays, making the model more robust to variations in packet timing.
2. **Traffic Scaling:** The second technique applied to the timestamps is traffic scaling. This method simulates different network speeds by randomly choosing a scaling factor from the set $\{0.5, 0.75, 1.0, 1.25, 1.5\}$. This scaling factor is then applied to the inter-packet times, either increasing them to simulate slower networks or reducing them to mimic high-speed links. This variation exposes the model to different network conditions and improves its ability to generalize across a wide range of traffic speeds.
3. **Packet Drop:** This is the first augmentation technique to operate at the packet level. It randomly drops a number of packets from each flow. The maximum number of packets that can be dropped depends on the length of the flow and is calculated as:

$$max\_packets\_to\_drop = \lfloor 0.25n - 0.5 \rfloor \tag{5.8}$$

   where n is the length of the flow. The actual number of packets to drop is drawn from the discrete uniform distribution:

$$\mathcal{U}\{0, max\_packets\_to\_drop\} \tag{5.9}$$

4. **Packet Insertion:** The packet insertion technique is the second packet-level method. It randomly adds a number of zero-byte packets into a flow. The maximum number of zero packets to be inserted is based on the flow length and is calculated as:

$$max\_packets\_to\_insert = \lfloor 0.15n - 0.5 \rfloor \tag{5.10}$$

   The actual number is again drawn from a discrete uniform distribution:

$$\mathcal{U}\{0, max\_packets\_to\_insert\} \tag{5.11}$$

5. **Noise Injection:** The final augmentation technique involves adding noise to the packet bytes. For each flow, at most $\lfloor n/3 \rfloor$ packets are modified, and for each modified packet, at most $\lfloor d/100 \rfloor$ bytes are altered. The values for the byte modification are randomly selected from a discrete uniform distribution. The noise itself is drawn from a continuous normal distribution with zero mean and a standard deviation of 0.1. Since all byte values have already been normalized to the range [0, 1], the added noise is small but effective in simulating random variations or errors in the packet data.

It is important to note that not all augmentation techniques are applied to every sample. For example, if a scaling factor of 1 is selected, no transformation occurs. Likewise, if no packets are chosen for dropping, insertion, or noise injection, the corresponding techniques remain inactive. Additionally, the first two augmentation techniques, which modify timestamps, are specifically designed for models that incorporate timestamp vectors as input. For models that do not utilize temporal information, these techniques are not applicable, further highlighting the advantage of time-aware models in network traffic analysis.

5.2.2.3 Early Detection Loss Function

Aiming to improve early classification performance, Early Detection Loss (EDeL) is introduced as a custom training loss function that increases both the model's accuracy and confidence when processing short flows. This is achieved by applying higher penalties to misclassifications in shorter flows than in longer ones. Such an approach encourages the model to become more accurate even when it has access to fewer packets, which is crucial for timely anomaly detection in real-time network traffic analysis.

During training, for each batch of b samples, the cross-entropy loss is first computed individually for each sample. The overall batch loss is then obtained as a weighted average of these individual losses:

$$L = \sum_{i=0}^{b-1} w_i CE_i \tag{5.12}$$

where w_i represents the weight associated with the i-th sample, and CE_i is the corresponding cross-entropy loss. The weight w_i is defined as:

$$w_i = \mathrm{e}^{-0.1 n_i} \tag{5.13}$$

where n_i denotes the length of the i-th flow in the batch. This weighting mechanism ensures that the model prioritizes minimizing errors in shorter flows, thereby improving its effectiveness in early classification scenarios. By integrating this custom loss function into the training process, the model is encouraged to make more accurate predictions with fewer packets, which is essential for applications requiring quick, real-time decisions.

5.2.3 System

In this work, the Transformer architecture is utilized as the core model, leveraging its self-attention mechanism to capture long-range dependencies in network flows, where each flow consists of a sequence of packets. Unlike traditional models, Transformers can effectively learn packet relationships independent of their positions, making them well-suited for attack detection and traffic classification. Additionally, the temporal aspect of packet arrival times can be crucial for identifying malicious activity, as many attacks exhibit distinct timing patterns. The Transformer architecture offers a significant advantage in this regard, as it can leverage the temporal information through time-aware positional encodings, allowing it to model temporal dependencies within a flow. This capability improves the accuracy and robustness of attack detection by leveraging both sequential and temporal information.

5.2.3.1 Base Model

The architecture of the base model—defined as the core Transformer architecture excluding positional encoding—is built upon the standard Transformer framework introduced in [28], with specific modifications tailored for network flow analysis. As the task focuses on classification, only the encoder component of the Transformer is utilized.

Since the preprocessed raw bytes from each packet directly serve as token embeddings, an explicit input embedding layer is not required. However, to align the input data with the model's feature space, a fully connected layer that maps the input dimension d to the hidden dimension d_{m} is applied. The transformed input is subsequently processed through a sequence of L Transformer encoder blocks, in which the standard GELU activation function is replaced with ReLU to enhance computational efficiency and promote training stability (see Subsection 4.4.2). Following the final Transformer encoder block, global average pooling aggregates the sequence of packet-level embeddings into a single fixed-length vector. This aggregated representation is then passed through a final fully connected layer comprising c output neurons, followed by a softmax activation function to produce class confidence scores.

Table 5.1 summarizes the hyperparameters employed in the proposed system. These values were selected to achieve rapid inference and a lightweight model architecture, essential for efficient operation in practical network environments. Notably, the constraint $d_{\mathrm{h}} = d_{\mathrm{m}}/h$ is not imposed, providing additional flexibility in configuring attention heads. The choice of values for input data representation was guided by empirical analysis and practical considerations:

- **A maximum sequence length of $N = 30$ packets** allows the model to capture sufficient contextual information while maintaining low computational complexity, considering that most malicious activities manifest within shorter packet sequences.
- **A packet feature size of $d = 448$** provides comprehensive representation, capturing essential packet information such as critical headers and payload content, necessary for accurate attack detection.

Table 5.1 System Hyperparameters and their Corresponding Values

Hyperparameter	Symbol	Value
Packet length	d	448
Maximum flow length	N	30
Hidden dimension	d_{m}	8
Number of Transformer blocks	L	1
Number of attention heads	h	4
Attention head dimension	d_{h}	8
FFN intermediate dimension	d_{ff}	16
Dropout rate	p_{drop}	0.1
Number of output classes	c	*dataset-specific*

The total number of trainable parameters in the base model is calculated as:

$$P = d_m[2L(2hd_{\mathrm{h}} + d_{\mathrm{ff}} + 3) + d + c + 1] + L(3hd_{\mathrm{h}} + d_{\mathrm{ff}}) + c \tag{5.14}$$

which indicates that increasing either the model width (d_{m}) or depth (L) significantly increases the model's parameter count, with direct implications for computational overhead and memory usage. With the selected hyperparameters, the base model consists of approximately 5100 total trainable parameters, excluding any positional encoding mechanisms. This lightweight architecture facilitates rapid decision-making while maintaining the necessary representational power to distinguish between normal and attack traffic.

5.2.3.2 Proposed Time-Aware Positional Encodings

Traditional positional encodings assume uniformly spaced sequence positions. However, in network traffic, packet flows exhibit varying inter-arrival times, making this assumption

inaccurate and potentially reducing the effectiveness of standard positional encodings. To address this, the predefined position indices, $\mathcal{P} = \{0, 1, \ldots, n\text{-}1\}$, are replaced with the actual timestamps of the packets in the flow, denoted as $T = \{t_0, t_1, \ldots, t_{n\text{-}1}\}$. This modification can be applied to sinusoidal, Fourier-based, and rotary positional encodings, resulting in three time-aware positional encoding variants specifically designed for sequential network traffic data.

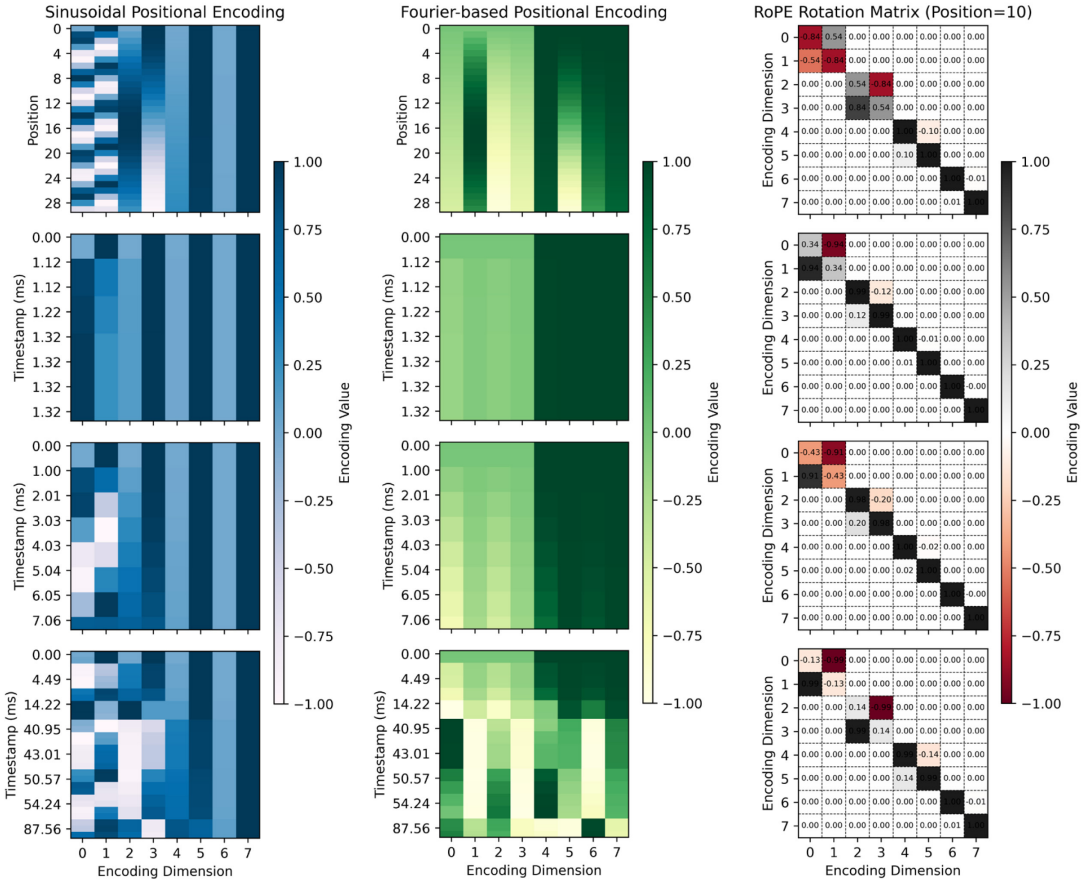

Figure 5.2 Time-aware positional encodings

Figure 5.2 demonstrates the capability of the proposed time-aware positional encodings to effectively represent temporal patterns within network traffic, compared to conventional positional encodings. The first row of the figure depicts traditional positional encodings, which rely exclusively on discrete packet indices and thus omit temporal context. The subsequent rows introduce the proposed time-aware positional encodings, computed using the absolute timestamps of packets within individual network flows. Specifically, the second row corresponds to a representative SSH brute-force attack scenario, characterized by packets transmitted in rapid succession, thus yielding subtle encoding variations. The third row illustrates benign network traffic with moderately spaced packet transmissions, resulting in clearer encoding distinctions. Finally, the fourth row corresponds to a web-based backdoor malware attack scenario with significantly delayed packet transmissions, exhibiting the most pronounced encoding variations. Across all positional encoding methods considered, the proposed time-aware variants distinctly and systematically reflect temporal differences, thereby highlighting their suitability for accurately capturing critical timing information essential for enhanced network flow classification.

Beyond network traffic analysis, the proposed encodings can be extended to encompass other types of time-series data where samples are generated at irregular intervals. Examples of such data include sensor readings recorded only during significant changes, user activity logs

(*e.g.*, clicks, searches, and logins), as well as social media and communication logs, among others. By integrating time-aware encodings, these domains can benefit from more precise temporal representations, improving downstream predictive tasks.

5.2.3.3 A-THENA

The proposed system identifies the optimal time-aware positional encoding by evaluating each encoding's effectiveness with respect to the specific characteristics of the attack types analyzed. The model achieving the lowest loss on the validation set among the proposed time-aware sinusoidal, Fourier-based, and rotary positional encodings is selected, ensuring superior generalization performance on unseen data. This integration of multiple time-aware encoding strategies forms the foundation of the system's **T**ime-Aware **H**ybrid **E**ncoding (**THE**), designed to precisely capture the temporal dynamics of network flows. Concurrently, the **N**etwork-Specific **A**ugmentation (**NA**) techniques, detailed in Subsection 5.2.2.2, enhance data diversity and further support robust generalization. The synergistic combination of time-aware representation learning and augmentation-driven robustness constitutes the core of the proposed early intrusion detection system, A-THENA.

5.3 Implementation

The implementation of A-THENA is structured into three main components: network data processing, model development, and deployment on edge devices. Scapy, a Python library for packet manipulation and network traffic analysis, is employed to extract relevant packet flows and features from raw PCAP files, enabling efficient flow reconstruction and preprocessing. The Transformer-based detection models are developed using TensorFlow, incorporating time-aware positional encodings to capture fine-grained temporal dependencies within network flows. To facilitate real-time inference on resource-constrained IoT devices, the trained models are deployed on the Raspberry Pi Zero 2 W using LiteRT, a lightweight runtime optimized for efficient deep learning execution, formerly known as TFLite. This deployment strategy ensures low-latency processing, high detection accuracy, and minimal resource overhead, making the system well-suited for practical IoT security applications where computational efficiency and rapid response times are critical.

5.4 Experimental Methodology

This section provides a detailed overview of the methodology employed for conducting experiments and systematically evaluating the performance of the proposed system. It details the experimental setup, dataset characteristics, and evaluation framework, ensuring a rigorous analysis of the proposed approach.

5.4.1 Datasets

For the experiments, three publicly available benchmark datasets for intrusion detection in IoT environments were employed: CICIoT2023 [167], MQTT-IoT-IDS2020 [168], and IoTID20 [169]. As outlined in Table 5.2, each dataset encompasses distinct attack categories. In the case of CICIoT2023, the focus was specifically placed on web-based attacks, resulting in a subset of the dataset referred to as CICIoT23-WEB. Consequently, incorporating the benign traffic class, the final classification tasks consist of 6, 5, and 9 classes for CICIoT23-WEB, MQTT-IoT-IDS2020, and IoTID20, respectively.

Table 5.2 Attack Categories in the Selected Datasets

CICIoT23-WEB	MQTT-IoT-IDS2020
SQL Injection Command Injection Backdoor Malware Uploading Attack Cross-Site Scripting (XSS)	Aggressive Scan UDP Scan Sparta SSH Brute-Force MQTT Brute-Force
IoTID20	
DoS SYN Flooding	Mirai ACK Flooding
Mirai Host Brute-Force	Mirai HTTP Flooding
Mirai UDP Flooding	MiTM ARP Spoofing
Host & Port Scan	OS Scan

The selected datasets provide a diverse and representative set of IoT intrusion scenarios, enabling a comprehensive evaluation of the system's generalization capabilities. CICIoT23-WEB focuses on web-based application-layer attacks (*e.g.*, SQL Injection, Command Injection, XSS) that target IoT web services. MQTT-IoT-IDS2020 covers protocol-specific and reconnaissance attacks (*e.g.*, MQTT Brute-Force, SSH Brute-Force, UDP Scans) affecting IoT communication protocols. IoTID20 includes Mirai botnet-driven DDoS attacks (*e.g.*, SYN Flooding, HTTP Flooding, MiTM ARP Spoofing) representing large-scale IoT security threats. By testing across these datasets, A-THENA is validated against application-layer, protocol-specific, and DDoS-based intrusions, demonstrating its robustness and real-world applicability in IoT security.

5.4.2 Training Configuration

For model optimization, the Adam optimizer is employed with a fixed learning rate of 0.0002. Moreover, the Early Detection Loss function described in Subsection 5.2.2.3 is used, along with a batch size of 4 samples. To prevent overfitting, early stopping is applied, which monitors the validation loss and halts training once performance stagnates.

5.4.3 Evaluation Metrics

To assess the effectiveness of A-THENA, an evaluation process was designed to closely replicate real-world deployment conditions. This approach ensures that performance metrics accurately reflect the system's practicality and reliability in real-world applications. In practice, when deployed on a host machine, the system continuously monitors network flows to promptly determine whether an attack is occurring (*Inference* stage in Figure 5.1). Given the critical need for early threat detection in cybersecurity, the evaluation focuses on both the accuracy and responsiveness of the system.

5.4.3.1 Confidence-Based Performance Metrics

To assess classification performance, a set of confidence-based metrics is employed. Given a confidence threshold τ applied to the top-1 softmax score, the evaluation focuses on how quickly the system arrives at a confident decision. The process begins with the first packet of each test flow, gradually adding packets until the model's confidence exceeds τ. If the threshold is not met after processing all N packets, the final classification is based on the full sequence.

A key metric is Earliness, measuring the number of packets needed before the model reaches the confidence threshold and a correct prediction is made. Lower values indicate faster, more efficient classification, which is crucial for real-time intrusion detection. At the threshold point, Top-1 Accuracy, *i.e.*, the percentage of correctly classified flows, False Negative Rate

(FNR), *i.e.*, the proportion of attack flows that are incorrectly classified as benign, and False Alarm Rate (FAR), *i.e.*, the proportion of benign flows mistakenly classified as attacks, are also computed. A high FNR is particularly concerning, as undetected attacks pose a severe security risk. Respectively, minimizing FAR is essential to prevent unnecessary security alerts, which can lead to operational inefficiencies and desensitization to genuine threats.

Lastly, the Early Risk Detection Error (ERDE) [170] is utilized, a metric that evaluates both the correctness of the model's predictions and the delay in reaching a decision. ERDE is a parametric metric; flows requiring more than o packets for accurate classification incur a higher penalty. Given a flow $F \in \mathcal{F}$, it is computed as follows:

$$ERDE_o(F) = \begin{cases} \frac{TP}{|\mathcal{F}|}, & \text{if } F \text{ is a False Positive} \\ 1, & \text{if } F \text{ is a False Negative} \\ 1 - \frac{1}{1 + e^{d-o}}, & \text{if } F \text{ is a True Positive} \\ 0, & \text{if } F \text{ is a True Negative} \end{cases} \tag{5.15}$$

where TP denotes the number of True Positives, and d represents the number of packets required to correctly classify a malicious flow, *i.e.*, the flow's earliness.

5.4.3.2 Resource-Constrained Deployment Evaluation

Beyond classification performance, the feasibility of deploying the proposed system on resource-limited edge devices is also assessed. To simulate real-world IoT applications, the system is deployed on the Raspberry Pi Zero 2 W, a compact, low-power embedded device featuring a 1GHz quad-core 64-bit ARM Cortex A53 processor and 512 MB of RAM, commonly used in IoT environments. This evaluation measures two key factors:

- **Latency:** The time required for the model to process and classify each packet. Low latency is essential for real-time intrusion detection.
- **Memory requirements:** The system's RAM and storage footprint, which determines whether it can be deployed on constrained IoT devices without excessive resource consumption.

The Raspberry Pi Zero 2 W serves as an ideal test platform, as its limited processing power and memory reflect the constraints of real-world IoT deployments. By validating the system's efficiency in such an environment, its suitability for lightweight cybersecurity applications is demonstrated.

5.5 Results

This section provides a detailed analysis of the evaluation results, highlighting the effectiveness of the proposed approach in comparison to existing methods.

5.5.1 Comparison Methods

The proposed system is evaluated against several well-established baselines and relevant prior works. Specifically, the comparison includes various traditional positional encodings, including sinusoidal, Fourier-based, and rotary encodings, along with embedding-based, convolutional, global relative encodings, and a model variant without any positional encoding.

Embedding Layer. The embedding layer provides a straightforward and effective approach for incorporating positional information into Transformer models. It maps integer indices to dense

vectors, which are learned during training to represent the position of tokens—or, in this case, packets—within a sequence. A key advantage of this approach is its flexibility, as the model learns data-driven positional encodings rather than relying on predefined functions. However, embedding-based encodings introduce additional parameters and can lead to overfitting, particularly in small models or when training data are limited.

Convolutional Encoding. The convolutional positional encoding employs a one-dimensional (1D) convolutional layer to dynamically learn positional information from the input sequence. Unlike fixed or explicitly learned positional embeddings, this approach allows the model to extract local positional dependencies directly from the data. The convolutional layer applies d_{m} learnable filters, each of size $K = 3$, across the sequence, effectively capturing short-range positional relationships between elements. By leveraging convolutional operations, this method provides a lightweight and adaptive alternative to traditional positional encodings, making it particularly useful for capturing spatial locality in network traffic sequences.

Global Relative Encoding. The concept of the relative positional encoding was first introduced by Shaw et al. [171] to enhance the self-attention mechanism by incorporating relative rather than absolute positional information. This approach was later refined by Huang et al. [172], improving efficiency and applicability in various sequence modeling tasks. The adopted implementation employs a global relative encoding scheme, where relative position information is incorporated directly into the attention computation. Specifically, the standard attention mechanism is modified by introducing a learnable relative position embedding matrix $\mathbf{E}_{\mathrm{r}} \in \mathbb{R}^{N \times d_{\mathrm{h}}}$, resulting in the following attention score formulation:

$$\mathbf{A} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T + \mathbf{Q}\mathbf{E}_{\mathrm{r}}^T}{\sqrt{d_{\mathrm{h}}}}\right) \tag{5.16}$$

Table 5.3 Summary of Considered Positional Encodings

Encoding	Learnable	Number of Parameters
Time-Aware Sinusoidal	✗	0
Sinusoidal	✗	$d_{\mathrm{m}} \cdot N = 240$
Time-Aware Fourier	✓	$d_{\mathrm{m}}/2 = 4$
Fourier	✓	$d_{\mathrm{m}}/2 = 4$
Time-Aware RoPE	✗	0
RoPE	✗	0
Embedding	✓	$d_{\mathrm{m}} \cdot N = 240$
Convolutional	✓	$d_{\mathrm{m}} \cdot (K \cdot d + 1) = 10760$
Global Relative	✓	$N \cdot d_{\mathrm{h}} = 240$

Table 5.3 presents a detailed summary of the characteristics and parameter counts of the nine considered encoding mechanisms, offering a comprehensive comparison of their impact on model performance. Additionally, based on the related work outlined in Subsection 5.1.3, the system is evaluated against four neural network architectures proposed for EIDS in [146], [147], [148], [149], referred to as eRNN, eTransformer, eAtt, and eGlo. Notably, these models do not utilize packet timestamps when classifying network flows.

5.5.2 Earliness and Accuracy

Figure 5.3 presents the evaluation of the system at a confidence threshold of 95% on the test set, assessing its performance across all metrics for the three datasets. Cases where FNR or FAR values are missing indicate that they remained zero for all evaluated models, signifying perfect detection with no false positives or false negatives. The results reveal distinct dataset-specific trends. Specifically, CICIoT23-WEB shows the highest variability, suggesting that encoding effectiveness depends on the attack types present. In contrast, MQTT-IoT-IDS2020 demonstrates stable performance, indicating reduced sensitivity to encoding variations. Meanwhile, IoTID20 highlights the importance of effectively handling longer network flows, as the complexity of its attack patterns necessitates the use of extended packet sequences for accurate detection.

A-THENA's time-aware hybrid encoding consistently yields the highest accuracy across all evaluated datasets, underscoring its effectiveness in modeling the temporal dynamics of network traffic. In contrast, traditional encodings perform notably worse, with A-THENA achieving average accuracy gains of 18.57, 7.58, and 9.21 percentage points on CICIoT23-WEB, MQTT-IoT-IDS2020, and IoTID20, respectively. This performance gap can be attributed to the inability of conventional positional encodings to account for irregular packet inter-arrival times, which are critical for detecting anomalies in IoT traffic. While some related work models (indicated by striped bars) demonstrate competitive performance, none surpass A-THENA. On average, A-THENA outperforms these existing models by 10.89, 6.26, and 6.69 points on the respective datasets, further validating the advantages of incorporating time-aware positional information.

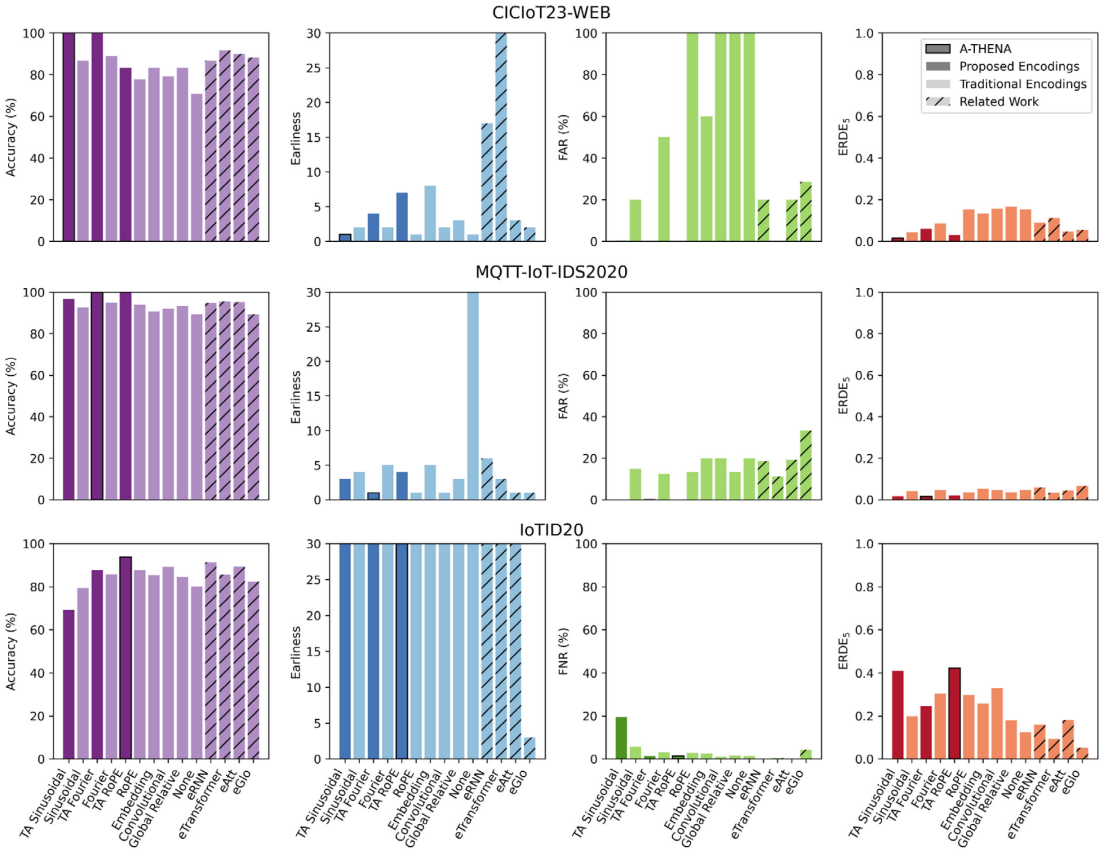

Figure 5.3 A-THENA's confidence-based evaluation

The results further indicate that lower earliness values generally align with higher accuracy, suggesting that earlier detection correlates with stronger classification performance. However, latency measurements on the Raspberry Pi Zero (detailed in the next section) reveal that the length n of an input flow has minimal impact on inference latency, allowing the system to classify longer sequences efficiently. Regarding false detection rates, CICIoT23-WEB and MQTT-IoT-IDS2020 exhibit higher FAR values for traditional encodings, which could lead to unnecessary security responses. IoTID20 achieves consistently low FNR values across all encodings, demonstrating the dataset's robustness in minimizing undetected intrusions. Models from related work generally show higher FAR/FNR values, reinforcing the advantages of the proposed hybrid encoding scheme in ensuring reliable detection. Lastly, while the best-performing time-aware encoding does not always correspond to the lowest $ERDE_5$ score, this discrepancy is mainly due to variations in earliness, which, as previously established, has minimal impact on latency.

The final models selected for each dataset, determined by the lowest validation loss, are as follows: for CICIoT23-WEB, the time-aware sinusoidal encoding proves to be the most effective, achieving 100% accuracy, 1-packet earliness, 0% FAR/FNR, and an $ERDE_5$ score of 0.015 on the test set. For MQTT-IoT-IDS2020, the time-aware Fourier encoding provides the best trade-off, yielding 100% accuracy, 1-packet earliness, 0% FAR/FNR, and an $ERDE_5$ score of 0.014. Lastly, for IoTID20, the time-aware RoPE encoding achieves the highest performance, with 93.83% accuracy, 30-packet earliness, 0% FAR, 1.39% FNR, and an $ERDE_5$ score of 0.422.

5.5.3 Latency and Memory Footprint

Table 5.4 presents the trade-offs between size (number of parameters), latency (ms) and memory footprint (MB) when deploying A-THENA on the Raspberry Pi Zero 2 W for CICIoT23-WEB, with flow lengths set to $n = 30$. The results demonstrate that A-THENA maintains a latency below 1.5 ms and a memory footprint under 4 MB, making it well-suited for deployment on resource-constrained edge devices. Experimental analysis indicates that both sequence length (n) and the choice of positional encoding have a negligible impact on these metrics, introducing only minor variations. Specifically, the system processes a single-packet flow in 0.17 ms, whereas a 30-packet flow requires just 1.42 ms, confirming that latency scales sub-linearly with sequence length. This property enables A-THENA to handle longer sequences efficiently without introducing excessive computational delays. Consequently, latency is primarily dictated by model size, reinforcing the necessity of designing a highly compact Transformer architecture optimized for real-time IoT intrusion detection.

Table 5.4 Model Complexity and Efficiency Comparison

System	Parameters	Latency	Memory Footprint
A-THENA	5086	1.42 ms	3.25 MB
eRNN	47559	20.12 ms	9.13 MB
eTransformer	1224838	35.79 ms	7.96 MB
eAtt	15655	1.39 ms	3.38 MB
eGlo	16934	0.57 ms	3.25 MB

When compared to related work, A-THENA achieves an optimal balance between computational efficiency and resource consumption, significantly outperforming alternative architectures in parameter efficiency. While eAtt and eGlo, the two CNN architectures, achieve comparable latency due to their convolutional structure, they require approximately three times

the number of parameters used by A-THENA, making them less efficient. Furthermore, despite their speed, these CNN models fail to match A-THENA's accuracy (see Figure 5.3), further emphasizing the advantage of time-aware hybrid encodings in intrusion detection. In contrast, eRNN and eTransformer exhibit significantly higher latency (20.12 ms and 35.79 ms, respectively), rendering them impractical for real-time IoT security applications. Additionally, their high memory footprint makes them unsuitable for low-power IoT deployments. These findings highlight that A-THENA effectively balances speed, memory efficiency, and detection performance, making it a highly efficient solution for early intrusion detection in IoT environments.

5.5.4 Evaluating Core Components of A-THENA

This section examines the core design components of A-THENA, including the augmentation pipeline, the proposed Early Detection Loss function, and the impact of quantization. Figure 5.4 presents the evaluation results of A-THENA and its ablated variants across the three considered datasets, highlighting the contributions of each component to overall system performance.

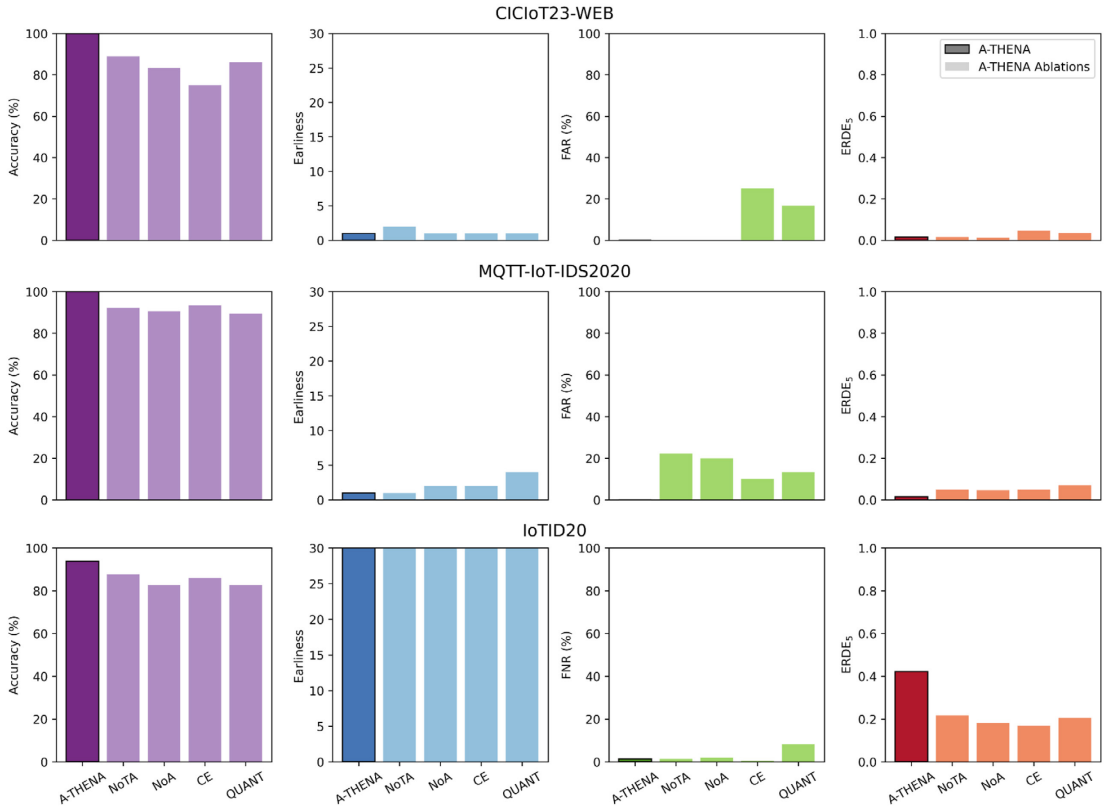

Figure 5.4 Impact of augmentation, EDeL, and quantization on A-THENA's performance

To analyze the role of augmentation, NoTA (No Timestamp Augmentation) applies only packet augmentation, omitting the two time-aware techniques—Jitter Injection and Traffic Scaling. In contrast, NoA (No Augmentation) removes all augmentation methods, providing a baseline for assessing the impact of data augmentation on early intrusion detection. The CE (Cross-Entropy) variant replaces A-THENA's EDeL loss function with a traditional cross-entropy loss, allowing an evaluation of the effectiveness of loss-based optimization for early detection. Lastly, QUANT applies post-training INT8 quantization, measuring the trade-off between model compression and detection performance.

The results strongly validate the design choices behind A-THENA, demonstrating that each component significantly contributes to achieving high-performance IoT intrusion detection across all datasets. Notably, the inclusion of time-aware augmentation techniques leads to superior detection accuracy and robustness, outperforming conventional packet-only augmentation. The findings reinforce the necessity of incorporating timestamp-aware transformations and specialized early detection optimization techniques to enhance real-time intrusion detection for IoT environments.

Table 5.5 presents the effects of post-training INT8 quantization on both latency speedup and memory reduction for A-THENA and related work models. The results show that quantization mainly enhances latency performance for A-THENA, achieving a 1.37× improvement. However, no reduction in memory footprint is observed. This lack of memory reduction is likely due to the inherent memory overhead associated with the LiteRT runtime. Specifically, LiteRT's memory arena, which manages intermediate tensors during model inference, imposes a fixed memory overhead. This overhead can be substantial, sometimes even exceeding the model's own size. Consequently, irrespective of how compact the model is, this baseline memory requirement remains relatively constant, limiting potential memory savings through quantization. Among the evaluated architectures, the eTransformer model experiences the most significant benefits, achieving a latency speedup of 1.52× and a substantial memory reduction of 1.77×. Other architectures demonstrate comparatively marginal improvements in these metrics.

Table 5.5 Quantization Benefits

System	Latency Speedup	Memory Reduction
A-THENA	1.37×	1.00×
eRNN	0.86×	1.04×
eTransformer	1.52×	1.77×
eAtt	1.43×	1.08×
eGlo	1.25×	1.13×

The limited effectiveness of quantization for A-THENA can further be explained by its already compact Transformer architecture, which offers minimal scope for additional compression. Conversely, larger models like eTransformer, containing over one million parameters, substantially benefit from reducing high-precision weights to INT8, resulting in considerable memory and latency improvements. Additionally, the slight performance degradation observed in the eRNN model (0.86× latency speedup) implies that out-of-the-box post-training quantization may introduce inefficiencies in recurrent computations.

5.6 Conclusion

This work presented A-THENA, an innovative early intrusion detection system tailored for IoT networks, addressing critical limitations in existing models through its Transformer-based architecture enriched with Time-Aware Hybrid Encoding (THE). By incorporating packet-level temporal information directly into positional encodings, A-THENA effectively captures intricate timing patterns indicative of malicious behavior, significantly enhancing detection performance. Additionally, the proposed augmentation pipeline for network-specific data, combined with the custom Early Detection Loss function, substantially improves both accuracy and early detection capabilities. Experimental evaluations across diverse IoT-specific datasets underline A-THENA's superior performance, achieving near-perfect accuracy, minimal false

detection rates, and efficient operation on resource-constrained devices. Deployments on edge hardware further affirm its practical applicability, demonstrating sub-millisecond inference latency and a minimal memory footprint. Future work may explore extending the approach to other sequential data domains and further optimizing the system for a broader array of IoT environments.

6 Discussion and Concluding Thoughts

The field of deep learning has seen extraordinary progress, enabling AI systems to achieve human-like capabilities in a range of domains, from natural language understanding to real-time perception and decision-making. However, this progress has largely been fueled by an ever-increasing demand for computational resources, making DL models reliant on powerful hardware and large-scale cloud infrastructures. This trend presents a fundamental challenge: while AI becomes more capable, its accessibility and usability in resource-constrained environments remain limited. The ability to bring intelligence to mobile and embedded devices is not just a technical goal but a necessity for AI to be truly pervasive.

This dissertation has explored ways to narrow the gap between state-of-the-art deep learning research and its practical application in mobile and embedded computing environments. Unlike conventional cloud-based AI, where computational resources are virtually unlimited, edge computing introduces a new paradigm—one that demands efficiency, adaptability, and optimization at every level, from model architecture to execution strategies. The research presented here has approached this challenge from multiple perspectives, contributing new methodologies to enhance DL efficiency while maintaining practical usability.

At its core, this work demonstrates that efficient AI is not just about minimizing resource usage but about rethinking how intelligence is deployed and executed in real-world conditions. The introduction of CARIn showcases how DL inference can be dynamically optimized for heterogeneous mobile hardware, ensuring that multiple models can operate efficiently despite fluctuating constraints. The study of Transformers in mobile environments provides critical insights into the architectural bottlenecks that hinder their deployment, offering optimizations to improve their feasibility for edge computing. Finally, the exploration of time-aware positional encodings for network security applications through A-THENA highlights how efficiency-driven AI solutions can extend beyond mainstream domains, addressing critical challenges in cybersecurity and IoT security.

6.1 Key Findings

The findings of this dissertation reinforce the idea that efficiency is not merely an optimization goal, but a fundamental requirement for the next generation of AI systems. The ability to execute DL models effectively in dynamic, resource-constrained environments is essential for ensuring their applicability beyond high-performance computing centers. The key insights gained from this research include:

- **Holistic optimization is necessary for mobile deep learning:** Isolated optimizations are insufficient for achieving practical efficiency. Instead, system-wide solutions that account for hardware heterogeneity, real-time adaptation, and multi-model execution are required.
- **Transformers must be redesigned for edge deployment:** While Transformer models have redefined AI, their current architectures remain incompatible with mobile constraints. Hardware-aware optimizations and model-specific adaptations are essential for bridging this gap.

- **Efficient AI can enhance real-time security applications**: The application of efficient deep learning techniques to intrusion detection and network security highlights their potential to improve the responsiveness and accuracy of cybersecurity solutions while maintaining computational feasibility.

6.2 Looking into the Future

While this dissertation addresses several pressing challenges in efficient deep learning, the field continues to evolve, presenting new research opportunities. Future work can expand upon these contributions in the following ways.

6.2.1 On-Device Training

This dissertation has primarily focused on systems for on-device inference, as the training phase remains significantly more demanding in terms of resource requirements. As outlined in Section 2.4, training involves repeated updates to model parameters through iterative computation, which imposes high demands on processing power, memory bandwidth, and energy consumption. Conventional training approaches—often referred to as batch training—necessitate access to large-scale datasets, prolonged training durations across multiple epochs, and extensive hyperparameter tuning.

With the emergence of powerful computational infrastructure and the availability of massive datasets, such training procedures have been successfully employed to develop foundation models, which are large, general-purpose models pre-trained on diverse data sources. These models can subsequently be fine-tuned for specialized downstream tasks using relatively smaller datasets, thereby leveraging the broad representational knowledge embedded in the foundation model. Executing full-scale batch training directly on mobile or embedded devices, however, remains impractical due to several key limitations:

- **Limited computational and memory resources**, which constrain the ability to handle large models or datasets.
- **The sequential and often sparse nature of data generation** on these platforms, making it difficult to accumulate sufficiently large and diverse training batches.
- **The low volume of data produced per device**, which reduces the effectiveness of standalone training efforts.
- **Stringent energy constraints**, especially for battery-powered devices, where prolonged or intensive computation can quickly deplete available power.

To overcome these challenges, the paradigm of online training has gained increasing attention. Online training builds upon the foundation of transfer learning, enabling lightweight, incremental updates to a pre-trained model using locally acquired data on the device itself. This approach significantly reduces the volume of data and the computational load required for each training step, making it better suited for constrained environments. A particularly promising research direction lies in the development of on-device online training techniques for personalization, allowing models to adapt to individual users' behaviors, preferences, or local conditions while preserving privacy and minimizing communication overhead.

One illustrative example involves extending a pre-trained classifier—originally trained offline to distinguish among n predefined categories—so that it can incrementally learn to recognize m new classes directly on the device, where m is smaller than n. This scenario reflects realistic usage conditions, as mobile and embedded systems continuously encounter novel data patterns that were not represented during initial training. Incremental, on-device adaptation

enables these models to remain up-to-date and context-aware without the need for full retraining. Beyond improving accuracy and generalization, this approach also enhances personalization, preserves user privacy by retaining data locally, and increases responsiveness by minimizing the need for cloud-based computation. These characteristics are well aligned with the emerging vision of edge intelligence and decentralized learning architectures.

The following sections examine this direction in greater detail, outlining the adopted methodology, the challenges inherent in enabling on-device online training, the experimental framework employed, and a summary of preliminary findings.

6.2.1.1 Server-Side Training

To support the experimental evaluation, the MobileNet V2 architecture, pre-trained on the ImageNet dataset containing 1000 object classes, was adopted. This pre-trained model served as the foundational backbone for the transfer learning approach targeting the CIFAR-10 dataset. To adapt the model to this new classification task, the original classification head was replaced with a new output layer compatible with the ten CIFAR-10 classes. In the context of this study, the model was trained on a subset of the CIFAR-10 dataset comprising $n = 9$ classes, with the final output neuron corresponding to the remaining $m = 1$ class intentionally left untrained for subsequent on-device learning.

Data preprocessing involved upsampling the CIFAR-10 images from their original resolution of 32×32 pixels to 160×160 pixels using bilinear interpolation, thereby aligning more closely with the input resolution expected by MobileNet V2. Following this, pixel intensities were normalized to the range [0, 1] to facilitate numerical stability and accelerate convergence. A suite of basic image data augmentation techniques was employed during training to enhance the model's generalization capacity. These included random horizontal flips, rotations, contrast adjustments, translations, and zoom transformations.

The training process utilized the Adam optimizer with a conservative learning rate of 10^{-6}, paired with the standard cross-entropy loss function. A batch size of 128 images was used. To mitigate overfitting and improve training efficiency, early stopping was implemented based on the validation performance. Initially, the convolutional feature extractor of the pre-trained MobileNet V2 was kept frozen to retain its general-purpose representations. After a few training epochs, it was selectively unfrozen to allow fine-tuning, thereby enabling the model to better adapt to the specific characteristics of the CIFAR-10 dataset.

Upon completion of the server-side training phase, the model achieved a classification accuracy of 95.6% on the held-out test set comprising the $n = 9$ classes. For comparison, a baseline model was also trained using the full set of $n + m = 10$ classes, resulting in a slightly lower accuracy of 95.3%. This slight difference underscores the potential of the proposed hybrid server-device training strategy to maintain performance while deferring parts of the learning process to the device itself.

6.2.1.2 Sample Efficiency Experiments

Training a model to recognize new classes presents multiple challenges, irrespective of whether training occurs on a server or directly on-device. One of the foundational concerns is understanding how the number of available training samples impacts model convergence, overall training duration, and classification performance. To explore this, a series of controlled experiments was conducted to investigate the relationship between the quantity of training data and the model's ability to integrate new knowledge while retaining previously acquired representations. These experiments were executed on a server environment but were carefully designed to simulate the conditions and constraints of on-device training.

A key objective was to determine the point of training stability—that is, the stage at which the model exhibits balanced classification accuracy across all classes, including both the $n = 9$ previously learned categories and the newly introduced $m = 1$ class. Among the various configurations tested, the most effective strategy involved training the model using mini-batches that included a mixture of samples from both the new class and the already-learned classes. This mixed-sample training strategy was found to mitigate the well-documented phenomenon of catastrophic forgetting, where a model's performance on previously learned classes deteriorates as it incorporates new knowledge. The presence of familiar samples in the training batches reinforces the model's prior representations, effectively counterbalancing the influence of the new class and maintaining overall classification integrity.

However, this approach introduces a practical limitation concerning data availability. Specifically, supporting on-device training using samples from the previously learned classes requires transferring a portion of that data to the device—a non-trivial task given the storage and memory constraints of mobile and embedded platforms. For context, each class in the CIFAR-10 dataset contains 5000 training images. Although modest in size by modern ML standards, transferring all 45000 images for the 9 known classes to a device is both inefficient and impractical. This necessitates the development of sample selection strategies that can identify and retain only the most informative examples from each class.

The underlying motivation behind sample selection is to maximize training efficacy while minimizing data volume. In other words, not all samples are considered equally valuable for the learning process, and identifying those with higher utility can significantly improve training efficiency. To this end, two distinct sampling methodologies were evaluated, assuming a fixed budget of $z = 5$ samples per class:

- **Random sampling:** Used as a baseline for comparison, this strategy involves selecting z samples at random from the training data for each class. While computationally simple, this approach does not incorporate any information about sample informativeness or model confidence, thus serving as a lower-bound reference point.
- **Confidence-based sampling:** This method leverages the Best-versus-Second-Best (BvSB) metric, a commonly used indicator of prediction confidence. The BvSB score quantifies the difference between the most likely and the second most likely class predictions. Higher BvSB values signify greater model certainty. For each class, the correctly predicted instances were retained, and the top z samples exhibiting the highest BvSB values were selected. This subset, referred to as the "best" set, is hypothesized to consist of clean, unambiguous examples that reinforce the model's internal representations during incremental learning.

In both cases, the resulting training dataset comprised $5 \times 9 = 45$ samples representing the previously learned classes. A similar issue emerges when selecting samples for evaluation. The CIFAR-10 test set contains 1000 images per class, amounting to a total of 10000 images. Transferring this full set to a user device is similarly infeasible. To address this, the effectiveness of stratified random sampling was examined on the test set, and the results indicated that it yields performance metrics statistically indistinguishable from those obtained using the full test set, thereby validating its suitability as a compact yet representative evaluation strategy.

In the final phase of the experiment—aimed at incorporating the 10^{th} class—the model was trained using a mini-batch size of 5. Each batch included four samples drawn randomly from either the "best" or "random" sets described above, along with one new sample from the target class to be learned. Each new sample was stored and reintroduced periodically after k epochs,

where $5 \leq k \leq 10$, to ensure sufficient exposure and memory consolidation. An optimal learning rate of 5×10^{-6} was identified through empirical tuning. Using this setup, the model was trained on a total of 45 unique samples for the new class across 160 training epochs. The training concluded upon reaching the previously defined accuracy stability point, where the model attained a final accuracy of 91.2% across all ten classes. This result demonstrates the feasibility of extending classification capabilities via selective, resource-aware training on constrained platforms.

6.2.1.3 Planned Extensions

While the results presented in this study are encouraging, the research remains at an early stage, with substantial potential for further exploration and refinement. Several promising directions have been identified for future work:

- **Generalization to additional datasets:** Future experiments should examine the model's ability to generalize by incorporating new classes drawn from other publicly available datasets or even real-world user-captured data. This will help evaluate how variations in data distribution affect model performance under the same learning framework.
- **Generalization across different configuration parameters:** Additional evaluations are needed for varying the number of known and new classes (*i.e.*, different *n*-*m* splits), as well as experimenting with different subset sizes for sample selection. This will allow for broader applicability and a deeper understanding of the trade-offs involved.
- **Hyperparameter tuning:** Systematic optimization of training parameters—such as learning rate and batch size—may lead to further improvements in training efficiency and accuracy, especially under constrained conditions.
- **Use of alternative base models:** While MobileNet V2 was selected for its balance between efficiency and performance, testing other lightweight architectures (*e.g.*, EfficientNet, MobileViT) may yield better adaptation or improved deployment characteristics for specific tasks.
- **Deployment on actual devices:** A critical next step involves moving from server-based simulations to real-world deployment on mobile and embedded hardware. This will allow for empirical validation of computational and memory demands, training time, and energy consumption in actual operating environments, and enable direct comparisons between server-side and on-device training.
- **Integration of few-shot or zero-shot learning techniques:** To further reduce the data requirements for on-device personalization, future work could explore methods that allow the model to generalize from minimal or even no labeled samples, expanding its adaptability to unseen categories.
- **Comparison with related works:** Finally, a thorough benchmarking against existing approaches for on-device learning and personalization will be essential to contextualize the proposed method's performance, efficiency, and practicality within the broader research landscape.

These directions aim to reinforce the feasibility of efficient on-device training while broadening the applicability of the proposed methodology across use cases, model architectures, and hardware platforms.

6.2.2 DNN Performance Prediction in Resource-Constrained Devices

Another promising future research direction arises from a core limitation of the CARIn framework, as discussed in Section 3.7. Specifically, the inability to conduct a fully end-to-end implementation and evaluation is primarily due to the computational and time-related burden associated with evaluating the objective functions of the underlying multi-objective optimization problem. These evaluations often involve deploying and benchmarking DNNs on target hardware platforms, a process that is both time-consuming and resource-intensive.

A potential solution lies in integrating performance prediction methodologies that can estimate key performance indicators of DNN execution on resource-constrained devices without requiring full deployment. Such metrics typically include latency, energy consumption, and memory footprint. Incorporating predictive models would not only enable CARIn to be evaluated more holistically—without exhaustive benchmarking for each candidate solution—but also has broader utility in the domain of DL system design. In particular, predictive performance models are commonly used to support NAS, automate hardware-aware model selection, and guide trade-off decisions between performance and resource efficiency.

This line of work can be framed as a series of regression tasks, where the objective is to train models capable of estimating the performance metrics of a DNN given its architectural description and hardware-specific parameters. According to the literature, such predictors can take various forms, including:

- **Analytical models**, which rely on mathematical expressions derived from low-level hardware and computational principles.
- **Rule-based models**, which use expert-defined heuristics to approximate performance.
- **Learning-based models**, which learn mappings from DNN configurations to performance metrics using empirical data.

This work focuses on learning-based approaches due to their adaptability, scalability, and ability to capture complex relationships that may not be easily modeled analytically. Such models can be particularly effective when trained on carefully constructed datasets containing diverse neural architectures and their measured performance on specific hardware platforms. The goal is to create these datasets and subsequently train compact neural networks—designed for both efficiency and deployment feasibility—that can predict latency and related metrics directly from architectural features.

The remainder of this section outlines the initial experimental methodology and reports early results obtained from a small-scale prototype, laying the groundwork for future large-scale predictive modeling within the CARIn framework and beyond.

6.2.2.1 Dataset Creation and Preprocessing

To enable learning-based latency prediction, a dataset was constructed by generating a large and diverse set of MLP models designed for classification tasks. The architectural space was defined by varying a set of hyperparameters, including the input size, number of hidden layers, number of neurons per layer, output size, activation function, and the inclusion or exclusion of components such as batch normalization and INT8 quantization. This process resulted in a total of 432 distinct MLP architectures.

Given the focus on mobile environments and the prediction of inference latency, each of these models was executed on physical smartphone devices. Latency was measured under different execution configurations, defined by parameters such as batch size, the selected processor (CPU or GPU), and processor-specific settings, including the number of CPU threads

or the precision level used in GPU inference (*e.g.*, FP32/FP16). By combining each model with various execution configurations, a comprehensive set of model-configuration pairs was obtained. In total, this procedure produced 21600 latency measurements per device, forming a structured dataset where each sample corresponds to a unique pairing of a neural network architecture and an execution configuration. Each sample is represented by 13 input features, comprising a mix of architectural and execution-level descriptors. These features include:

- **Numerical variables** (*e.g.*, number of hidden layers, total number of neurons, batch size).
- **Categorical variables** (*e.g.*, activation function, processor type).
- **Binary variables** (*e.g.*, presence of batch normalization, quantization flag).

The preprocessing pipeline followed standard ML practices. Categorical and binary features were encoded using a combination of one-hot encoding (for nominal categories) and ordinal encoding (where appropriate). All numerical features were normalized to the [0, 1] range using min-max scaling to ensure feature value comparability and to promote model convergence during training. To assess generalization and device-specific variation, the experimental setup was replicated across two smartphones with differing hardware characteristics (*e.g.*, processors, available memory). This produced two distinct datasets, referred to as Dataset A and Dataset B, each with shape (21600, 13), providing a basis for evaluating model performance across heterogeneous mobile platforms.

6.2.2.2 Preliminary Results from Training

During the initial stage of experimentation, the focus was placed on training models tailored to individual devices to assess whether a compact MLP architecture could reliably predict inference latency based on device-specific characteristics. To enhance the granularity of the analysis, each dataset was further divided according to the type of processor used—CPU or GPU—thereby enabling the development of processor-specific models. This approach enabled the investigation of how effectively latency prediction could be specialized for distinct hardware components within a device.

Each dataset, along with its corresponding partitions, was randomly divided into training, validation, and test subsets using an 80/10/10 split ratio. Following a random search for hyperparameter optimization, the MLP regression models were trained and evaluated, with the results presented in Table 6.1. These results indicate the mean absolute error (MAE) on the respective test sets.

Table 6.1 Processor-Specific Latency Prediction Models

Processor	Test MAE
A	1.3116
A-CPU	0.5456
A-GPU	0.5847
B	4.6927
B-CPU	2.6073
B-GPU	2.0410

The results demonstrate that all models achieved relatively low MAE values, confirming that simple neural networks can effectively learn latency prediction functions tailored to specific devices and processors. The performance discrepancy between Device A and Device B is attributed to the differing latency scales: Device B exhibited significantly higher average

latencies (approximately 12 ms), compared to Device A (around 5 ms), making precise prediction more challenging. Notably, all models contained fewer than 10000 trainable parameters, underscoring their suitability for deployment in resource-constrained environments.

In a second line of investigation, the generalization capabilities of these models to unseen hyperparameter values were explored. For instance, all samples corresponding to an input size of 50 features were excluded from the training set and reserved exclusively for testing. The results indicated that the models could generalize reasonably well to previously unseen values, particularly when the missing values lay within the internal range of the training data distribution. For example, when intermediate batch size values (*e.g.*, 2, 4, 8) were excluded from training, the model produced better predictions than when the boundary values (*e.g.*, 1 and 16) were omitted. This suggests that the learned representations are more robust when interpolating rather than extrapolating.

Finally, the investigation was extended to a multi-device setting to assess whether a single model could generalize across devices. For this purpose, the input space was augmented with additional features characterizing the device's hardware, including CPU frequency, GPU frequency, and RAM capacity. After conducting hyperparameter tuning via random search, the best-performing model achieved a test MAE of 0.9395 with a total of 20449 trainable parameters, highlighting its capacity to effectively integrate cross-device information and maintain predictive accuracy. This result opens the possibility of creating unified latency predictors applicable to a broad range of mobile and embedded platforms.

6.2.2.3 Planned Extensions

Building upon the promising preliminary results, several directions are envisioned to extend and enhance this line of research on latency prediction in resource-constrained environments. These planned extensions aim to improve model robustness, generalization capability, and practical applicability across a broader range of hardware platforms and neural architectures.

1. **The dataset will be expanded to include a more diverse set of neural network models** beyond MLPs, such as convolutional neural networks and attention-based architectures. These architectures are more commonly deployed in mobile applications and pose additional challenges due to their more complex computational patterns and hardware dependencies. Including them will significantly increase the representativeness of the dataset and the applicability of the resulting prediction models.
2. **The data collection process will be expanded to incorporate a larger variety of hardware devices**, including additional smartphones, tablets, and embedded systems with different processor architectures (*e.g.*, ARM Cortex-A series, NPUs, and DSPs). This will facilitate the development of more generalized predictors capable of handling cross-platform latency estimation. The augmented device-specific features may also be enriched with other hardware descriptors, such as cache size, number of cores, and available hardware accelerators.
3. **Another planned extension is the use of more advanced learning techniques**, such as ensemble models or Transformer-based regressors, to further reduce prediction error and improve the model's capacity to generalize to unseen configurations. Additionally, the use of model-agnostic feature attribution techniques will be investigated (*e.g.*, SHAP or permutation importance) to better understand the contribution of each feature to the final latency prediction.

4. **The trained latency predictors are intended to be deployed as practical tools** for real-time performance estimation within neural architecture search pipelines or adaptive model selection systems. This integration would enable dynamic decision-making in environments where execution latency must be continuously optimized in response to changing hardware conditions or application requirements.

Overall, these extensions will reinforce the foundation laid by the initial work and support the broader objective of enabling efficient and predictive performance modeling for deep learning in mobile and embedded computing contexts.

6.2.3 Generative AI on Mobile and Embedded Platforms

The field of artificial intelligence has experienced significant advancements with the emergence of generative AI models, including large language models, large vision models, and diffusion models. These models have exhibited exceptional performance in areas like NLP, computer vision, and creative content generation. However, their capabilities come at the cost of heightened computational complexity, substantial memory demands, and considerable energy consumption. As a result, such models typically cannot operate outside high-performance computing and memory infrastructures.

In this context, the principal objective of efficient AI is the optimization of cloud-based deployment in order to enhance sustainability and reduce operational costs. This involves techniques such as model pruning, quantization, and hardware-aware neural architecture search, all of which are designed to reduce computational overhead without significantly compromising model performance. Additionally, it encompasses workload scheduling and resource management strategies tailored for energy-efficient inference and training in large-scale data centers. Conversely, deploying large-scale generative models on resource-constrained edge devices is generally regarded as a secondary concern due to the significant technical obstacles involved—challenges that, in many cases, render such deployment impractical. Addressing these obstacles would require transformative advances not only in model compression and optimization methods but also in the computational, memory, and energy efficiency of edge hardware itself.

6.2.3.1 Current State

Generative AI has made significant inroads into mobile and embedded systems, though the deployment landscape remains constrained by hardware limitations and model size. The current state is defined by a pragmatic shift toward smaller, highly optimized models that can run efficiently on modern mobile processors and edge devices:

- **Language models** such as LLaMA 2, Mistral, and Phi have been successfully compressed using quantization and pruning techniques, enabling them to run locally on smartphones and single-board computers. Libraries like *llama.cpp* and formats such as GGUF have made it possible to load and execute these models with acceptable performance on devices with as little as 4–8 GB of RAM, especially when paired with neural accelerators like the Apple Neural Engine or Qualcomm's Hexagon DSP. These models are already used in offline personal assistants, local text summarization tools, and chat applications that preserve user privacy by avoiding cloud inference.
- **In the realm of image generation**, applications like *Draw Things* on iOS have demonstrated that it is entirely feasible to run variants of Stable Diffusion on-device, particularly with hardware support from Apple's Core ML or Metal Performance Shaders. Although such implementations are typically slower and limited in resolution

compared to cloud-based services, they offer a viable path for privacy-conscious and latency-sensitive use cases like mobile art creation or content personalization.

- **Speech-related generative models** have also been adapted for mobile use. Projects like *whisper.cpp* bring real-time transcription to phones and edge devices, whereas compact text-to-speech models like Bark can synthesize human-like voices without needing to connect to external servers. These capabilities are increasingly being embedded into smart home assistants, accessibility tools, and voice-enabled mobile applications.

Despite these advances, the deployment of full-scale generative models—like GPT-4 or large diffusion models used in video generation—remains impractical for purely mobile or embedded systems. These models still require significant computational and memory resources, necessitating hybrid architectures where part of the inference is performed locally and part in the cloud. Nonetheless, a growing number of applications now adopt a local-first philosophy, using on-device generation for fast, lightweight tasks while falling back on cloud services only when necessary.

Overall, the state of generative AI on mobile and embedded systems is one of constrained but growing capability. What was once exclusively the domain of high-performance GPUs in data centers can now, in its lighter forms, operate effectively in users' pockets and at the network edge. This has opened the door to a new class of private, interactive, and personalized applications powered by generative AI—without always relying on a persistent internet connection.

6.2.3.2 Future Research Directions and Open Challenges

Despite the progress in deploying generative AI models on mobile and embedded platforms, several open challenges remain, shaping a vibrant landscape for future research.

- **Trade-off between model expressiveness and resource efficiency:** While compression techniques have enabled lightweight deployments, they often come at the cost of reduced generation quality, slower inference, or limited generalization. Developing new architectures specifically designed for low-power generative inference—rather than merely adapting existing large models—remains a key area of exploration.
- **Optimization of memory usage and latency under hardware constraints:** Generative models, particularly Transformers and diffusion models, are inherently resource-hungry, often requiring large context windows, multiple layers of attention, or iterative denoising steps. Efficient memory scheduling, smarter attention mechanisms, and reduced-precision computation will be critical to sustaining smooth user experiences in real-time applications.
- **Energy efficiency:** Continuous generation on battery-powered devices raises concerns about heat, battery life, and sustainability, particularly for models running in the background or interacting with real-world sensors. Hardware–software co-design—where model development is tightly coupled with the characteristics of target platforms—offers a promising avenue, especially when leveraging custom accelerators like NPUs and edge TPUs.
- **Privacy and security implications:** While local inference avoids sharing raw data with cloud services, it also raises questions of local model integrity and potential misuse. Research into secure model execution, tamper-proof inference pipelines, and

watermarking of locally generated content is essential, particularly in regulated domains like healthcare or finance.

- **Model personalization:** Current approaches to on-device fine-tuning are limited by available memory and compute, making continuous learning and user adaptation difficult. Finding methods to personalize generative models incrementally and efficiently—possibly through federated learning or sparse updates—remains a compelling direction for future work.
- **Cross-modal generation:** Tasks like text-to-video or speech-to-image remain largely undeveloped in the mobile context due to high resource demands and model complexity. Creating new paradigms for multi-modal generation that can operate across distributed compute nodes—some on the device, some in the edge cloud—could unlock richer interactive applications without overwhelming a single system.

In summary, while current solutions mark a promising beginning, the future of generative AI in mobile and embedded systems depends on interdisciplinary innovation across model design, hardware acceleration, and secure deployment strategies. Bridging the gap between performance and efficiency will define the next phase of intelligent computing at the edge.

6.2.4 AI-Enabled Networking

The convergence of AI and networking is rapidly emerging as a foundational pillar for the next generation of communication systems. As networks become more dynamic, heterogeneous, and large-scale—especially in the context of mobile computing, IoT, and edge-cloud architectures—the need for adaptive, data-driven decision-making is more critical than ever. AI, and in particular DL, offers the means to shift from rule-based, static configurations to systems that learn, predict, and optimize in real time.

Despite this promise, AI-enabled networking remains significantly less explored than other AI application areas such as image recognition or language understanding. This discrepancy can be attributed to several key factors. First, network data are often highly structured, temporal, and dynamic, involving packet flows, protocol stacks, topologies, and multivariate statistics that are difficult to encode effectively into formats suitable for standard AI pipelines. Second, unlike the abundance of publicly available image and text datasets, network data are scarce, heterogeneous, and often privacy-sensitive, limiting the pace of benchmarking and reproducibility. Moreover, ground-truth labels are costly to obtain and are frequently context-specific, further complicating model generalization.

Yet the potential for synergy is immense. AI can unlock unprecedented capabilities in areas such as traffic prediction, routing optimization, intrusion detection, and quality-of-service adaptation. Conversely, networking provides the infrastructure that AI systems depend on for distributed training, real-time inference, and federated learning across edge devices. The mutual dependency suggests that networking and AI are not merely complementary, but co-evolving disciplines, each enabling new levels of performance and scalability for the other.

Future research must address several open challenges. Developing domain-specific representations and architectures tailored to network data—such as graph neural networks, temporal attention mechanisms, and protocol-aware embeddings—is essential to improve the expressiveness and interpretability of AI models in this domain. Furthermore, online learning and adaptive systems capable of operating under distributional drift, data scarcity, and limited computational resources will be key to practical deployments. There is also a pressing need for benchmark datasets, simulation environments, and standardized evaluation frameworks to accelerate progress and collaboration in this space.

In conclusion, the synergy between AI and networking has the potential to fundamentally reshape the architecture and operation of future networks. Realizing this vision will require sustained interdisciplinary effort, bridging communication theory, systems design, and modern AI. As the demands on networks continue to grow—driven by ubiquitous connectivity, autonomous systems, and real-time applications—AI-enabled networking stands as a critical frontier for both innovation and impact.

6.2.5 Additional Directions

Beyond the targeted future work discussed in the previous sections, several broader directions hold significant potential for advancing the field of efficient deep learning in mobile and embedded computing environments:

- **Next-generation edge AI systems:** A promising avenue involves the dynamic partitioning of AI models across hybrid cloud-edge infrastructures. By enabling inference or training to be offloaded selectively to local or cloud resources depending on context (*e.g.*, latency constraints, energy availability, connectivity), such systems can achieve a balance between responsiveness and scalability. Further research is needed to develop adaptive model-splitting strategies, optimize runtime migration, and design coordination mechanisms that minimize overhead.
- **Energy-aware model optimization:** As energy efficiency remains a central constraint in mobile and embedded AI, future work could explore optimization frameworks that explicitly target energy consumption as a primary objective. This includes training models with energy profiles in mind, developing energy-aware loss functions, and integrating energy prediction models into deployment pipelines. Such efforts would align with the growing demand for sustainable and autonomous AI deployments.
- **Robustness and security of on-device models:** With AI models increasingly deployed at the edge, they become exposed to new classes of threats, including adversarial attacks and data poisoning. Future work should consider the development of lightweight adversarial defense mechanisms suitable for constrained environments. Moreover, incorporating trustworthiness and certification mechanisms for AI behavior under uncertainty is essential for safety-critical applications.
- **Self-supervised and continual learning in the wild:** Leveraging unlabeled data generated on-device through self-supervised learning can further reduce dependence on labeled datasets. Similarly, online learning approaches capable of long-term adaptation without catastrophic forgetting are essential for real-world, evolving deployments. This calls for efficient algorithms that can learn incrementally while being aware of concept drift and data distribution shifts.
- **Cross-disciplinary system co-design:** Achieving true efficiency in mobile and embedded AI will increasingly require holistic, co-designed solutions that span the algorithm-hardware-software stack. Future research could explore tighter integration of neural architecture design with compiler-level optimizations and hardware characteristics, leading to systems where models are not only functionally effective but natively optimized for their target execution environments.
- **AI for systems and systems for AI:** Lastly, the reciprocal relationship between AI and systems design continues to offer fertile ground for innovation. Future work can explore how AI techniques can be used to design better networks, operating systems, or resource allocation schemes, and conversely, how novel system architectures can empower new forms of adaptive and distributed intelligence at the edge.

Together, these directions complement the dissertation's contributions by expanding the scope from efficient on-device inference and training to a broader vision of intelligent, adaptive, and sustainable AI deployment across the entire spectrum of edge computing environments.

6.3 Final Thoughts

Conducting this research has been a journey through the fundamental trade-offs that define modern AI: power versus efficiency, accuracy versus latency, and complexity versus adaptability. While much of DL research focuses on maximizing accuracy, working within the constraints of mobile and embedded environments has revealed a deeper truth—true intelligence is not just about raw computational power but about how well a system can adapt, optimize, and function under real-world limitations. What has stood out throughout this process is that efficiency is not merely an afterthought or a secondary goal—it is a defining factor in whether AI can scale beyond academic research and high-performance computing clusters. The future of AI does not lie solely in creating larger and more complex models, but in ensuring that these models can function where they are needed most—in personal devices, autonomous systems, and real-time environments that shape everyday life.

This dissertation is not just the culmination of years of research, but a step toward a broader vision of AI that is ubiquitous, sustainable, and seamlessly integrated into the fabric of modern technology. The work presented here lays a foundation, but many questions remain unanswered. How can deep learning continue to evolve without unsustainable computational costs? How can AI be optimized for new and emerging hardware architectures? How can intelligence be made more adaptable, learning to function within constraints rather than being limited by them? These are questions that extend far beyond the scope of this dissertation, but they are the questions that will define the next era of AI. While this research provides some answers, it also serves as an invitation for further exploration—because the challenge of making AI both powerful and efficient is not just a research problem, but one of the most important technological challenges of our time.

6.4 Publications

The research conducted throughout this dissertation has led to several significant publications, contributing to the broader scientific discourse on efficient deep learning for mobile and embedded computing. These publications reflect the key findings, methodologies, and innovations developed in this work, showcasing the impact of this research within the academic and technological communities.

6.4.1 Journal Articles

- **I. Panopoulos**, S. I. Venieris, and I. S. Venieris, "CARIn: Constraint-Aware and Responsive Inference on Heterogeneous Devices for Single- and Multi-DNN Workloads," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 4, pp. 1–32, Jul. 2024, doi: 10.1145/3665868.

6.4.2 Peer-Reviewed Conference/Workshop Papers

- **I. Panopoulos**, S. Nikolaidis, S. I. Venieris, and I. S. Venieris, "Exploring the Performance and Efficiency of Transformer Models for NLP on Mobile Devices," in *2023 IEEE Symposium on Computers and Communications (ISCC)*, Gammarth, Tunisia: IEEE, Jul. 2023, pp. 1–4, doi: 10.1109/ISCC58397.2023.10217850.

- S. I. Venieris, **I. Panopoulos**, and I. S. Venieris, "OODIn: An Optimised On-Device Inference Framework for Heterogeneous Mobile Devices," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, Irvine, CA, USA: IEEE, Aug. 2021, pp. 1–8, doi: 10.1109/SMARTCOMP52413.2021.00021.
- S. I. Venieris, **I. Panopoulos**, I. Leontiadis, and I. S. Venieris, "How to Reach Real-Time AI on Consumer Devices? Solutions for Programmable and Custom Architectures," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, NJ, USA: IEEE, Jul. 2021, pp. 93–100, doi: 10.1109/ASAP52443.2021.00022.

6.4.3 Under Review/Accepted

- **I. Panopoulos**, M.-L. Bartsioka, S. Nikolaidis, S. I. Venieris, D. I. Kaklamani, and I. S. Venieris, "Dynamic Temporal Positional Encodings for Early Intrusion Detection in IoT," *10th International Conference on Smart and Sustainable Technologies (SpliTech 2025)*, (accepted), 2025.
- **I. Panopoulos**, M.-L. Bartsioka, S. Nikolaidis, S. I. Venieris, D. I. Kaklamani, and I. S. Venieris, "A-THENA: Early Intrusion Detection for IoT with Time-Aware Hybrid Encoding and Network-Specific Augmentation," *IEEE Transactions on Machine Learning in Communications and Networking (TMLCN)*, (under review), 2025.

Glossary

Term	Translation
accelerator	επιταχυντής
accessibility	προσβασιμότητα
accuracy	ακρίβεια
activation function	συνάρτηση ενεργοποίησης
adaptive	προσαρμοστικός
application	εφαρμογή
Artificial Intelligence (AI)	τεχνητή νοημοσύνη
artificial neural network	τεχνητό νευρωνικό δίκτυο
attack	επίθεση
attention mechanism	μηχανισμός προσοχής
augmentation	επαύξηση
baseline	γραμμή αναφοράς
batch	δέσμη
benchmarking	συγκριτική αξιολόγηση
Central Processing Unit (CPU)	κεντρική μονάδα επεξεργασίας
classification	κατηγοριοποίηση
cloud	νέφος
computational complexity	υπολογιστική πολυπλοκότητα
computational resources	υπολογιστικοί πόροι
Computer Vision	όραση υπολογιστών
computing environment	υπολογιστικό περιβάλλον
confidence	πεποίθηση
convolutional neural network (CNN)	συνελικτικό νευρωνικό δίκτυο
cross-entropy	διασταυρούμενη εντροπία
cybersecurity	κυβερνοασφάλεια
dataset	σύνολο δεδομένων
decision-making	λήψη αποφάσεων
decoder	αποκωδικοποιητής
Deep Learning (DL)	βαθιά μάθηση
deep neural network (DNN)	βαθύ νευρωνικό δίκτυο
delegate	εκπρόσωπος
deployment	ανάπτυξη
deterministic	ντετερμινιστικός
device	συσκευή
Digital Signal Processor (DSP)	επεξεργαστής ψηφιακού σήματος
distance	απόσταση
earliness	πρωιμότητα
early intrusion detection	έγκαιρη ανίχνευση εισβολών
Early Risk Detection Error (ERDE)	σφάλμα έγκαιρης ανίχνευσης κινδύνου
edge	παρυφές
edge computing	υπολογιστική παρυφών
edge intelligence	ευφυής υπολογιστική παρυφών
edge server	διακομιστής παρυφών
efficiency	αποδοτικότητα
efficient deep learning	αποδοτική βαθιά μάθηση
embedded computing	ενσωματωμένος υπολογισμός
embedding	διανυσματική αναπαράσταση
encoder	κωδικοποιητής

energy consumption	κατανάλωση ενέργειας
ensemble model	μοντέλο συνόλου
execution	εκτέλεση
false alarm rate (FAR)	ποσοστό ψευδών συναγερμών
false negative rate (FNR)	ποσοστό ψευδώς αρνητικών δειγμάτων
feedforward network	δίκτυο εμπρόσθιας τροφοδοσίας
filtering	φιλτράρισμα
flow	ροή
Fourier-based positional encoding	κωδικοποίηση θέσης Fourier
framework	πλαίσιο
fully connected layer	πλήρως συνδεδεμένο επίπεδο
generalization	γενίκευση
generative	παραγωγικός
Graphics Processing Unit (GPU)	μονάδα επεξεργασίας γραφικών
hardware	υλικό
heterogeneity	ετερογένεια
hyperparameter	υπερπαράμετρος
input	είσοδος
intelligence	ευφυία
Internet of Things (IoT)	διαδίκτυο των πραγμάτων
Intrusion Detection System (IDS)	σύστημα ανίχνευσης εισβολών
large language model (LLM)	μεγάλο γλωσσικό μοντέλο
latency	καθυστέρηση / χρόνος απόκρισης
loss function	συνάρτηση απώλειας
Machine Learning (ML)	μηχανική μάθηση
memory	μνήμη
memory footprint	αποτύπωμα μνήμης
metric	μετρική
microcontroller	μικροελεγκτής
mobile computing	κινητός υπολογισμός
model	μοντέλο
multi-head attention	προσοχή πολλαπλών κεφαλών
multi-objective optimization (MOO)	βελτιστοποίηση πολλαπλών στόχων
Natural Language Processing (NLP)	επεξεργασία φυσικής γλώσσας
network	δίκτυο
Neural Processing Unit (NPU)	μονάδα νευρωνικής επεξεργασίας
noise	θόρυβος
normalization	κανονικοποίηση
objective	στόχος
on-device inference	τοπική συμπερασματολογία
optimality	βελτιστότητα
optimization	βελτιστοποίηση
output	έξοδος
oversampling	υπερδειγματοληψία
packet	πακέτο
parameter	παράμετρος
pipeline	αγωγός
pooling	συγκέντρωση
positional encoding	κωδικοποίηση θέσης
preprocessing	προεπεξεργασία
privacy	ιδιωτικότητα
processing power	υπολογιστική ισχύς
processor	επεξεργαστής
quantization	κβαντοποίηση
raw data	ακατέργαστα δεδομένα

real-time	πραγματικού χρόνου
recurrent neural network (RNN)	επαναληπτικό νευρωνικό δίκτυο
rotary positional encoding	περιστροφική κωδικοποίηση θέσης
runtime	χρόνος εκτέλεσης
Runtime Manager (RM)	Διαχειριστής Εκτέλεσης
scaling	κλιμάκωση
security	ασφάλεια
sensor	αισθητήρας
sequence	ακολουθία
sequential data	ακολουθιακά δεδομένα
service-level objective (SLO)	στόχος επιπέδου υπηρεσίας
sinusoidal positional encoding	ημιτονοειδής κωδικοποίηση θέσης
size	μέγεθος
smartphones	ευφυές τηλέφωνο
subflow	υπορροή
sustainability	βιωσιμότητα
switching policy (SP)	πολιτική εναλλαγής
system	σύστημα
System-on-Chip (SoC)	σύστημα-σε-μικροτσίπ
task	διεργασία
testing	έλεγχος
thread	νήμα
throughput	ρυθμός διέλευσης
time-aware	χρονικά ευαίσθητος
timestamp	χρονική σφραγίδα
token	σύμβολο
trade-off	αντιστάθμιση
traffic data	δεδομένα κίνησης
training	εκπαίδευση
Transformer	μετασχηματιστής
use-case	σενάριο χρήσης
validation	επικύρωση
wearable device	φορετή συσκευή
workload	υπολογιστικό φορτίο
zero-padding	παραγέμισμα με μηδενικά

References

[1] H. Sheikh, C. Prins, and E. Schrijvers, *Mission AI: The New System Technology*. in Research for Policy, 2023. doi: 10.1007/978-3-031-21448-6.

[2] M. U. Hadi *et al.*, "Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects," *TechRxiv*, 2023. doi: 10.36227/techrxiv.23589741.v8.

[3] T. S. Ajani, A. L. Imoize, and A. A. Atayero, "An Overview of Machine Learning within Embedded and Mobile Devices–Optimizations and Applications," *Sensors*, vol. 21, no. 13, p. 4412, 2021, doi: 10.3390/s21134412.

[4] Y. Chen, B. Zheng, Z. Zhang, Q. Wang, C. Shen, and Q. Zhang, "Deep Learning on Mobile and Embedded Devices: State-of-the-art, Challenges, and Future Directions," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–37, 2021, doi: 10.1145/3398209.

[5] T. Guo, "Cloud-Based or On-Device: An Empirical Study of Mobile Deep Inference," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, Orlando, FL: IEEE, 2018, pp. 184–190. doi: 10.1109/IC2E.2018.00042.

[6] X. Jiang *et al.*, "MNN: A Universal and Efficient Inference Engine," in *Proceedings of Machine Learning and Systems*, 2020, pp. 1–13.

[7] J. Yuan *et al.*, "Mobile Foundation Model as Firmware," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, Washington D.C. DC USA: ACM, 2024, pp. 279–295. doi: 10.1145/3636534.3649361.

[8] C. Ni, J. Wu, H. Wang, W. Lu, and C. Zhang, "Enhancing cloud-based large language model processing with Elasticsearch and transformer models," in *International Conference on Image, Signal Processing, and Pattern Recognition (ISPP 2024)*, Guangzhou, China: SPIE, 2024, p. 66. doi: 10.1117/12.3033606.

[9] V. K. Singh, B. Moharana, T. Sarkar, A. Maan, A. Bhattacherjee, and M. Rakhra, "Practical Assessment of Large Language Models in Cloud Computing Using Real-World Data Applications," in *2024 International Conference on Cybernation and Computation (CYBERCOM)*, Dehradun, India: IEEE, 2024, pp. 356–362. doi: 10.1109/CYBERCOM63683.2024.10803266.

[10] M. Abbasi, A. Shahraki, and A. Taherkordi, "Deep Learning for Network Traffic Monitoring and Analysis (NTMA): A Survey," *Comput. Commun.*, vol. 170, pp. 19–41, 2021, doi: 10.1016/j.comcom.2021.01.021.

[11] I. Panopoulos, S. Venieris, and I. Venieris, "CARIn: Constraint-Aware and Responsive Inference on Heterogeneous Devices for Single- and Multi-DNN Workloads," *ACM Trans. Embed. Comput. Syst.*, vol. 23, no. 4, pp. 1–32, 2024, doi: 10.1145/3665868.

[12] I. Panopoulos, S. Nikolaidis, S. I. Venieris, and I. S. Venieris, "Exploring the Performance and Efficiency of Transformer Models for NLP on Mobile Devices," in *2023 IEEE Symposium on Computers and Communications (ISCC)*, Gammarth, Tunisia: IEEE, 2023, pp. 1–4. doi: 10.1109/ISCC58397.2023.10217850.

[13] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, “Transformers in Vision: A Survey,” *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 1–41, 2022, doi: 10.1145/3505244.

[14] I. Lauriola, A. Lavelli, and F. Aiolli, “An introduction to Deep Learning in Natural Language Processing: Models, techniques, and tools,” *Neurocomputing*, vol. 470, pp. 443–456, 2022, doi: 10.1016/j.neucom.2021.05.103.

[15] S. Zhai *et al.*, “Fine-Tuning Large Vision-Language Models as Decision-Making Agents via Reinforcement Learning,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2024, pp. 110935–110971.

[16] D. Kaul, H. Raju, and B. K. Tripathy, “Deep Learning in Healthcare,” in *Deep Learning in Data Analytics*, vol. 91, in Studies in Big Data, Cham: Springer International Publishing, 2022, pp. 97–115. doi: 10.1007/978-3-030-75855-4_6.

[17] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958, doi: 10.1037/h0042519.

[18] M. Minsky and S. Papert, *Perceptrons: an introduction to computational geometry*, Expanded ed. Cambridge, Mass: MIT Press, 1988.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, doi: 10.1038/323533a0.

[20] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995, doi: 10.1007/BF00994018.

[21] J. R. Quinlan, “Induction of decision trees,” *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986, doi: 10.1023/A:1022643204877.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2012.

[24] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[25] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[26] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” in Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 6105–6114.

[27] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[28] A. Vaswani *et al.*, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, 2017.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Association for Computational Linguistics, 2019, pp. 4171–4186. doi: 10.18653/V1/N19-1423.

[30] OpenAI *et al.*, “GPT-4 Technical Report,” 2023, *arXiv*. doi: 10.48550/ARXIV.2303.08774.

[31] C. Raffel *et al.*, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[32] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le, “XLNet: Generalized Autoregressive Pretraining for Language Understanding,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019, pp. 5754–5764.

[33] A. Dosovitskiy *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021.

[34] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-End Object Detection with Transformers,” in *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, in Lecture Notes in Computer Science, vol. 12346. Springer, 2020, pp. 213–229. doi: 10.1007/978-3-030-58452-8_13.

[35] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-Resolution Image Synthesis with Latent Diffusion Models,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, IEEE, 2022, pp. 10674–10685. doi: 10.1109/CVPR52688.2022.01042.

[36] A. Ramesh *et al.*, “Zero-Shot Text-to-Image Generation,” in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, in Proceedings of Machine Learning Research, vol. 139. PMLR, 2021, pp. 8821–8831.

[37] A. Radford *et al.*, “Learning Transferable Visual Models From Natural Language Supervision,” in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, in Proceedings of Machine Learning Research, vol. 139. PMLR, 2021, pp. 8748–8763.

[38] J.-B. Alayrac *et al.*, “Flamingo: a Visual Language Model for Few-Shot Learning,” in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[39] K. Cao, Y. Liu, G. Meng, and Q. Sun, “An Overview on Edge Computing Research,” *IEEE Access*, vol. 8, pp. 85714–85728, 2020, doi: 10.1109/ACCESS.2020.2991734.

[40] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, “A First Look at Deep Learning Apps on Smartphones,” in *The World Wide Web Conference*, San Francisco CA USA: ACM, 2019, pp. 2125–2136. doi: 10.1145/3308558.3313591.

[41] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, “Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence,” *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7457–7469, 2020, doi: 10.1109/JIOT.2020.2984887.

[42] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, “Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing,” *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019, doi: 10.1109/JPROC.2019.2918951.

[43] J. Chen and X. Ran, “Deep Learning With Edge Computing: A Review,” *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019, doi: 10.1109/JPROC.2019.2921977.

[44] M. Almeida, S. Laskaridis, A. Mehrotra, L. Dudziak, I. Leontiadis, and N. D. Lane, “Smart at what cost?: characterising mobile deep neural networks in the wild,” in *Proceedings of the 21st ACM Internet Measurement Conference*, Virtual Event: ACM, 2021, pp. 658–672. doi: 10.1145/3487552.3487863.

[45] C.-J. Wu *et al.*, “Machine Learning at Facebook: Understanding Inference at the Edge,” in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Washington, DC, USA: IEEE, 2019, pp. 331–344. doi: 10.1109/HPCA.2019.00048.

[46] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris, and N. D. Lane, “EmBench: Quantifying Performance Variations of Deep Neural Networks across Modern Commodity Devices,” in *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications*, Seoul Republic of Korea: ACM, 2019, pp. 1–6. doi: 10.1145/3325413.3329793.

[47] A. Ignatov *et al.*, “AI Benchmark: All About Deep Learning on Smartphones in 2019,” in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Seoul, Korea (South): IEEE, 2019, pp. 3617–3635. doi: 10.1109/ICCVW.2019.00447.

[48] A. K. Singh, S. Dey, K. McDonald-Maier, K. R. Basireddy, G. V. Merrett, and B. M. Al-Hashimi, “Dynamic Energy and Thermal Management of Multi-core Mobile Platforms: A Survey,” *IEEE Des. Test*, vol. 37, no. 5, pp. 25–33, Oct. 2020, doi: 10.1109/MDAT.2020.2982629.

[49] S. I. Venieris, C.-S. Bouganis, and N. D. Lane, “Multiple-Deep Neural Network Accelerators for Next-Generation Artificial Intelligence Systems,” *Computer*, vol. 56, no. 3, pp. 70–79, Mar. 2023, doi: 10.1109/MC.2022.3176845.

[50] J. S. Jeong *et al.*, “Band: coordinated multi-DNN inference on heterogeneous mobile processors,” in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, Portland Oregon: ACM, Jun. 2022, pp. 235–247. doi: 10.1145/3498361.3538948.

[51] B. Cox, J. Galjaard, A. Ghiassi, R. Birke, and L. Y. Chen, “Masa: Responsive Multi-DNN Inference on the Edge,” in *2021 IEEE International Conference on Pervasive*

Computing and Communications (PerCom), Kassel, Germany: IEEE, Mar. 2021, pp. 1–10. doi: 10.1109/PERCOM50583.2021.9439111.

[52] G. Menghani, "Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–37, Dec. 2023, doi: 10.1145/3578938.

[53] L. Wei, Z. Ma, C. Yang, and Q. Yao, "Advances in the Neural Network Quantization: A Comprehensive Review," *Appl. Sci.*, vol. 14, no. 17, p. 7445, Aug. 2024, doi: 10.3390/app14177445.

[54] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A White Paper on Neural Network Quantization," 2021, *arXiv*. doi: 10.48550/ARXIV.2106.08295.

[55] S. I. Venieris, I. Panopoulos, and I. S. Venieris, "OODIn: An Optimised On-Device Inference Framework for Heterogeneous Mobile Devices," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, Irvine, CA, USA: IEEE, Aug. 2021, pp. 1–8. doi: 10.1109/SMARTCOMP52413.2021.00021.

[56] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "μLayer: Low Latency On-Device Inference Using Cooperative Single-Layer Acceleration and Processor-Friendly Quantization," in *Proceedings of the Fourteenth EuroSys Conference 2019*, Dresden Germany: ACM, Mar. 2019, pp. 1–15. doi: 10.1145/3302424.3303950.

[57] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra, "High-Throughput CNN Inference on Embedded ARM Big.LITTLE Multicore Processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2254–2267, Oct. 2020, doi: 10.1109/TCAD.2019.2944584.

[58] S. I. Venieris, I. Panopoulos, I. Leontiadis, and I. S. Venieris, "How to Reach Real-Time AI on Consumer Devices? Solutions for Programmable and Custom Architectures," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, NJ, USA: IEEE, Jul. 2021, pp. 93–100. doi: 10.1109/ASAP52443.2021.00022.

[59] H. Cheng, M. Zhang, and J. Q. Shi, "A Survey on Deep Neural Network Pruning: Taxonomy, Comparison, Analysis, and Recommendations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 12, pp. 10558–10578, Dec. 2024, doi: 10.1109/TPAMI.2024.3447085.

[60] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge Distillation: A Survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, Jun. 2021, doi: 10.1007/s11263-021-01453-z.

[61] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 17, 2017, *arXiv*: arXiv:1704.04861. doi: 10.48550/arXiv.1704.04861.

[62] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT: IEEE, Jun. 2018, pp. 4510–4520. doi: 10.1109/CVPR.2018.00474.

[63] M. Tan and Q. Le, “EfficientNetV2: Smaller Models and Faster Training,” in *Proceedings of the 38th International Conference on Machine Learning*, in Proceedings of Machine Learning Research, vol. 139. PMLR, Jul. 2021, pp. 10096–10106.

[64] X. Jiao *et al.*, “TinyBERT: Distilling BERT for Natural Language Understanding,” in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Association for Computational Linguistics, Nov. 2020, pp. 4163–4174. doi: 10.18653/v1/2020.findings-emnlp.372.

[65] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” Mar. 01, 2020, *arXiv*: arXiv:1910.01108. doi: 10.48550/arXiv.1910.01108.

[66] M. Berman, L. Pishchulin, N. Xu, M. B. Blaschko, and G. Medioni, “AOWS: Adaptive and Optimal Network Width Search With Latency Constraints,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Jun. 2020, pp. 11214–11223. doi: 10.1109/CVPR42600.2020.01123.

[67] H. Bouzidi, M. Odema, H. Ouarnoughi, M. A. Al Faruque, and S. Niar, “HADAS: Hardware-Aware Dynamic Neural Architecture Search for Edge Performance Scaling,” in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium: IEEE, Apr. 2023, pp. 1–6. doi: 10.23919/DATE56975.2023.10137095.

[68] B. Wu *et al.*, “FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA: IEEE, Jun. 2019, pp. 10726–10734. doi: 10.1109/CVPR.2019.01099.

[69] L. L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu, “Fast Hardware-Aware Neural Architecture Search,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*, Computer Vision Foundation / IEEE, 2020, pp. 2959–2967. doi: 10.1109/CVPRW50498.2020.00354.

[70] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, “Once-for-All: Train One Network and Specialize it for Efficient Deployment,” in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.

[71] A. Kundu, L. Wynter, R. D. Lee, and L. Angel Bathen, “Transfer-Once-For-All: AI Model Optimization for Edge,” in *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, Chicago, IL, USA: IEEE, Jul. 2023, pp. 26–35. doi: 10.1109/EDGE60047.2023.00017.

[72] J. Lee, J. Rhim, D. Kang, and S. Ha, “SNAS: Fast Hardware-Aware Neural Architecture Search Methodology,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4826–4836, Nov. 2022, doi: 10.1109/TCAD.2021.3134843.

[73] H. Wen *et al.*, “AdaptiveNet: Post-deployment Neural Architecture Adaptation for Diverse Edge Environments,” in *Proceedings of the 29th Annual International*

Conference on Mobile Computing and Networking, Madrid Spain: ACM, Oct. 2023, pp. 1–17. doi: 10.1145/3570361.3592529.

[74] P. Dollar, M. Singh, and R. Girshick, "Fast and Accurate Model Scaling," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA: IEEE, Jun. 2021, pp. 924–932. doi: 10.1109/CVPR46437.2021.00098.

[75] J. Xie, X. Su, S. You, Z. Ma, F. Wang, and C. Qian, "ScaleNet: Searching for the Model to Scale," in *Computer Vision – ECCV 2022*, vol. 13681, in Lecture Notes in Computer Science, vol. 13681, Cham: Springer Nature Switzerland, 2022, pp. 104–120. doi: 10.1007/978-3-031-19803-8_7.

[76] J. Guo, S. Xia, and C. Peng, "OPA: One-Predict-All For Efficient Deployment," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, New York City, NY, USA: IEEE, May 2023, pp. 1–10. doi: 10.1109/INFOCOM53939.2023.10228928.

[77] B. Chen, M. Lin, R. Ji, and L. Cao, "Prioritized Subnet Sampling for Resource-Adaptive Supernet Training," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 9, pp. 11108–11119, Sep. 2023, doi: 10.1109/TPAMI.2023.3265198.

[78] R. Han, Q. Zhang, C. H. Liu, G. Wang, J. Tang, and L. Y. Chen, "LegoDNN: block-grained scaling of deep neural networks for mobile vision," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, New Orleans Louisiana: ACM, Oct. 2021, pp. 406–419. doi: 10.1145/3447993.3483249.

[79] T. Yang, S. Zhu, C. Chen, S. Yan, M. Zhang, and A. Willis, "MutualNet: Adaptive ConvNet via Mutual Learning from Network Width and Resolution," in *Computer Vision – ECCV 2020*, vol. 12346, in Lecture Notes in Computer Science, vol. 12346, Cham: Springer International Publishing, 2020, pp. 299–315. doi: 10.1007/978-3-030-58452-8_18.

[80] P. Guo, B. Hu, and W. Hu, "Mistify: Automating DNN Model Porting for On-Device Inference at the Edge," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, USENIX Association, Apr. 2021, pp. 705–719.

[81] S. Laskaridis, S. I. Venieris, H. Kim, and N. D. Lane, "HAPI: hardware-aware progressive inference," in *Proceedings of the 39th International Conference on Computer-Aided Design*, Virtual Event USA: ACM, Nov. 2020, pp. 1–9. doi: 10.1145/3400302.3415698.

[82] S. Liu, B. Guo, K. Ma, Z. Yu, and J. Du, "AdaSpring: Context-adaptive and Runtime-evolutionary Deep Model Compression for Mobile Applications," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 5, no. 1, pp. 1–22, Mar. 2021, doi: 10.1145/3448125.

[83] B. Kutukcu, S. Baidya, A. Raghunathan, and S. Dey, "Contention Grading and Adaptive Model Selection for Machine Vision in Embedded Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 21, no. 5, pp. 1–29, Sep. 2022, doi: 10.1145/3520134.

[84] Z. Xu, D. Yang, C. Yin, J. Tang, Y. Wang, and G. Xue, "A Co-Scheduling Framework for DNN Models on Mobile and Edge Devices with Heterogeneous Hardware," *IEEE Trans. Mob. Comput.*, pp. 1–1, 2021, doi: 10.1109/TMC.2021.3107424.

[85] H. Fan, S. I. Venieris, A. Kouris, and N. Lane, “Sparse-DySta: Sparsity-Aware Dynamic and Static Scheduling for Sparse Multi-DNN Workloads,” in *56th Annual IEEE/ACM International Symposium on Microarchitecture*, Toronto ON Canada: ACM, Oct. 2023, pp. 353–366. doi: 10.1145/3613424.3614263.

[86] A. Kouris, S. I. Venieris, S. Laskaridis, and N. D. Lane, “Fluid Batching: Exit-Aware Preemptive Serving of Early-Exit Neural Networks on Edge NPUs,” 2022, *arXiv*. doi: 10.48550/ARXIV.2209.13443.

[87] Z. Zhang, H. Li, Y. Zhao, C. Lin, and J. Liu, “BCEdge: SLO-Aware DNN Inference Services with Adaptive Batching on Edge Platforms,” 2023, *arXiv*. doi: 10.48550/ARXIV.2305.01519.

[88] Z. Zhang, Y. Zhao, and J. Liu, “Octopus: SLO-Aware Progressive Inference Serving via Deep Reinforcement Learning in Multi-tenant Edge Cluster,” in *Service-Oriented Computing*, vol. 14420, in Lecture Notes in Computer Science, vol. 14420, Cham: Springer Nature Switzerland, 2023, pp. 242–258. doi: 10.1007/978-3-031-48424-7_18.

[89] W. Seo, S. Cha, Y. Kim, J. Huh, and J. Park, “SLO-Aware Inference Scheduler for Heterogeneous Processors in Edge Platforms,” *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, pp. 1–26, Dec. 2021, doi: 10.1145/3460352.

[90] J. Yi and Y. Lee, “Heimdall: mobile GPU coordination platform for augmented reality applications,” in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, London United Kingdom: ACM, Sep. 2020, pp. 1–14. doi: 10.1145/3372224.3419192.

[91] X. He, X. Wang, Z. Zhou, J. Wu, Z. Yang, and L. Thiele, “On-Device Deep Multi-Task Inference via Multi-Task Zipping,” *IEEE Trans. Mob. Comput.*, vol. 22, no. 5, pp. 2878–2891, May 2023, doi: 10.1109/TMC.2021.3124306.

[92] M. Yuan, L. Zhang, Z. Zheng, Y.-N. Zhang, and X.-Y. Li, “MLink: Linking Black-Box Models From Multiple Domains for Collaborative Inference,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 10, pp. 12085–12097, Oct. 2023, doi: 10.1109/TPAMI.2023.3283780.

[93] A. Karatzas and I. Anagnostopoulos, “OmniBoost: Boosting Throughput of Heterogeneous Embedded Devices under Multi-DNN Workload,” in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA: IEEE, Jul. 2023, pp. 1–6. doi: 10.1109/DAC56929.2023.10247989.

[94] N. Ling, K. Wang, Y. He, G. Xing, and D. Xie, “RT-mDL: Supporting Real-Time Mixed Deep Learning Tasks on Edge Platforms,” in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, Coimbra Portugal: ACM, Nov. 2021, pp. 1–14. doi: 10.1145/3485730.3485938.

[95] S. Eyerman and L. Eeckhout, “System-Level Performance Metrics for Multiprogram Workloads,” *IEEE Micro*, vol. 28, no. 3, pp. 42–53, May 2008, doi: 10.1109/MM.2008.44.

[96] F. Yu, D. Wang, L. Shangguan, M. Zhang, C. Liu, and X. Chen, “A Survey of Multi-Tenant Deep Learning Inference on GPU,” 2022, *arXiv*. doi: 10.48550/ARXIV.2203.09040.

[97] Q. Yang *et al.*, “GMorph: Accelerating Multi-DNN Inference via Model Fusion,” in *Proceedings of the Nineteenth European Conference on Computer Systems*, Athens Greece: ACM, Apr. 2024, pp. 505–523. doi: 10.1145/3627703.3650074.

[98] F. Yu *et al.*, “Automated Runtime-Aware Scheduling for Multi-Tenant DNN Inference on GPU,” in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Munich, Germany: IEEE, Nov. 2021, pp. 1–9. doi: 10.1109/ICCAD51958.2021.9643501.

[99] N. Gunantara, “A review of multi-objective optimization: Methods and its applications,” *Cogent Eng.*, vol. 5, no. 1, p. 1502242, Jan. 2018, doi: 10.1080/23311916.2018.1502242.

[100] J. L. J. Pereira, G. A. Oliver, M. B. Francisco, S. S. Cunha, and G. F. Gomes, “A Review of Multi-objective Optimization: Methods and Algorithms in Mechanical Engineering Problems,” *Arch. Comput. Methods Eng.*, vol. 29, no. 4, pp. 2285–2308, Jun. 2022, doi: 10.1007/s11831-021-09663-x.

[101] S. Eyerman and L. Eeckhout, “System-Level Performance Metrics for Multiprogram Workloads,” *IEEE Micro*, vol. 28, no. 3, pp. 42–53, May 2008, doi: 10.1109/MM.2008.44.

[102] S. Sharma and V. Kumar, “A Comprehensive Review on Multi-objective Optimization Techniques: Past, Present and Future,” *Arch. Comput. Methods Eng.*, vol. 29, no. 7, pp. 5605–5633, Nov. 2022, doi: 10.1007/s11831-022-09778-9.

[103] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015, doi: 10.1007/s11263-015-0816-y.

[104] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollar, “Designing Network Design Spaces,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Jun. 2020, pp. 10425–10433. doi: 10.1109/CVPR42600.2020.01044.

[105] S. Mehta and M. Rastegari, “MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer,” in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net, 2022.

[106] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning Transferable Architectures for Scalable Image Recognition,” in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, Computer Vision Foundation / IEEE Computer Society, 2018, pp. 8697–8710. doi: 10.1109/CVPR.2018.00907.

[107] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A ConvNet for the 2020s,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, IEEE, 2022, pp. 11966–11976. doi: 10.1109/CVPR52688.2022.01167.

[108] E. Saravia, H.-C. T. Liu, Y.-H. Huang, J. Wu, and Y.-S. Chen, “CARER: Contextualized Affect Representations for Emotion Recognition,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 3687–3697. doi: 10.18653/v1/D18-1404.

[109] S. Mukherjee, A. H. Awadallah, and J. Gao, “XtremeDistilTransformers: Task Transfer for Task-agnostic Distillation,” Jun. 12, 2021, *arXiv*: arXiv:2106.04563. doi: 10.48550/arXiv.2106.04563.

[110] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, “MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, 2020, pp. 2158–2170. doi: 10.18653/v1/2020.acl-main.195.

[111] A. Quattoni and A. Torralba, “Recognizing indoor scenes,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL: IEEE, Jun. 2009, pp. 413–420. doi: 10.1109/CVPR.2009.5206537.

[112] J. F. Gemmeke *et al.*, “Audio Set: An ontology and human-labeled dataset for audio events,” in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA: IEEE, Mar. 2017, pp. 776–780. doi: 10.1109/ICASSP.2017.7952261.

[113] Z. Zhang, Y. Song, and H. Qi, “Age Progression/Regression by Conditional Adversarial Autoencoder,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI: IEEE, Jul. 2017, pp. 4352–4360. doi: 10.1109/CVPR.2017.463.

[114] C. Raffel *et al.*, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[115] Y. Liu *et al.*, “RoBERTa: A Robustly Optimized BERT Pretraining Approach,” 2019, *arXiv*. doi: 10.48550/ARXIV.1907.11692.

[116] L. Lu, C. Liu, J. Li, and Y. Gong, “Exploring Transformers for Large-Scale Speech Recognition,” in *Interspeech 2020*, ISCA, Oct. 2020, pp. 5041–5045. doi: 10.21437/Interspeech.2020-2638.

[117] Y. Feng, M. Xie, Z. Tian, S. Wang, Y. Lu, and J. Shu, “Mobius: Fine Tuning Large-Scale Models on Commodity GPU Servers,” in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, Vancouver BC Canada: ACM, Jan. 2023, pp. 489–501. doi: 10.1145/3575693.3575703.

[118] V. J. Reddi *et al.*, “MLPerf Inference Benchmark,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, Valencia, Spain: IEEE, May 2020, pp. 446–459. doi: 10.1109/ISCA45697.2020.00045.

[119] Q. Cao, A. E. Irimiea, M. Abdelfattah, A. Balasubramanian, and N. D. Lane, “Are Mobile DNN Accelerators Accelerating DNNs?,” in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, Virtual WI USA: ACM, Jun. 2021, pp. 7–12. doi: 10.1145/3469116.3470011.

[120] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, “A survey of modern deep learning based object detection models,” *Digit. Signal Process.*, vol. 126, p. 103514, Jun. 2022, doi: 10.1016/j.dsp.2022.103514.

[121] X. Wang, L. L. Zhang, Y. Wang, and M. Yang, “Towards efficient vision transformer inference: a first study of transformers on mobile devices,” in *Proceedings of the 23rd*

Annual International Workshop on Mobile Computing Systems and Applications, Tempe Arizona: ACM, Mar. 2022, pp. 1–7. doi: 10.1145/3508396.3512869.

[122] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2020, pp. 5776–5788.

[123] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.

[124] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," *OpenAI*, 2019.

[125] H. Kopetz and W. Steiner, "Internet of Things," in *Real-Time Systems*, Cham: Springer International Publishing, 2022, pp. 325–341. doi: 10.1007/978-3-031-11992-7_13.

[126] A. E. Omolara *et al.*, "The internet of things security: A survey encompassing unexplored areas and new insights," *Comput. Secur.*, vol. 112, p. 102494, 2022, doi: https://doi.org/10.1016/j.cose.2021.102494.

[127] L. Mohammadpour, T. C. Ling, C. S. Liew, and A. Aryanfar, "A Survey of CNN-Based Network Intrusion Detection," *Appl. Sci.*, vol. 12, no. 16, p. 8162, Aug. 2022, doi: 10.3390/app12168162.

[128] I. Ullah and Q. H. Mahmoud, "Design and Development of RNN Anomaly Detection Model for IoT Networks," *IEEE Access*, vol. 10, pp. 62722–62750, 2022, doi: 10.1109/ACCESS.2022.3176317.

[129] L. D. Manocchio, S. Layeghy, W. W. Lo, G. K. Kulatilleke, M. Sarhan, and M. Portmann, "FlowTransformer: A transformer framework for flow-based network intrusion detection systems," *Expert Syst. Appl.*, vol. 241, p. 122564, May 2024, doi: 10.1016/j.eswa.2023.122564.

[130] M. F. Saiyedand and I. Al-Anbagi, "Deep Ensemble Learning With Pruning for DDoS Attack Detection in IoT Networks," *IEEE Trans. Mach. Learn. Commun. Netw.*, vol. 2, pp. 596–616, 2024, doi: 10.1109/TMLCN.2024.3395419.

[131] M. Lopez-Vizcaino, F. J. Novoa, D. Fernandez, V. Carneiro, and F. Cacheda, "Early Intrusion Detection for OS Scan Attacks," in *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA, USA: IEEE, Sep. 2019, pp. 1–5. doi: 10.1109/NCA.2019.8935067.

[132] N. Patwardhan, S. Marrone, and C. Sansone, "Transformers in the Real World: A Survey on NLP Applications," *Information*, vol. 14, no. 4, p. 242, Apr. 2023, doi: 10.3390/info14040242.

[133] M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "CorrAUC: A Malicious Bot-IoT Traffic Detection Method in IoT Network Using Machine-Learning Techniques," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3242–3254, Mar. 2021, doi: 10.1109/JIOT.2020.3002255.

[134] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, “Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection,” in *Proceedings 2018 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2018. doi: 10.14722/ndss.2018.23204.

[135] K. Farhana, M. Rahman, and Md. T. Ahmed, “An intrusion detection system for packet and flow based networks using deep neural network approach,” *Int. J. Electr. Comput. Eng. IJECE*, vol. 10, no. 5, p. 5514, Oct. 2020, doi: 10.11591/ijece.v10i5.pp5514-5525.

[136] P. Toupas, D. Chamou, K. M. Giannoutakis, A. Drosou, and D. Tzovaras, “An Intrusion Detection System for Multi-class Classification Based on Deep Neural Networks,” in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, Boca Raton, FL, USA: IEEE, Dec. 2019, pp. 1253–1258. doi: 10.1109/ICMLA.2019.00206.

[137] M. A. Khan *et al.*, “A Deep Learning-Based Intrusion Detection System for MQTT Enabled IoT,” *Sensors*, vol. 21, no. 21, p. 7016, Oct. 2021, doi: 10.3390/s21217016.

[138] Y. Zhang, X. Chen, D. Guo, M. Song, Y. Teng, and X. Wang, “PCCN: Parallel Cross Convolutional Neural Network for Abnormal Network Traffic Flows Detection in Multi-Class Imbalanced Network Traffic Flows,” *IEEE Access*, vol. 7, pp. 119904–119916, 2019, doi: 10.1109/ACCESS.2019.2933165.

[139] A. Tekerek, “A novel architecture for web-based attack detection using convolutional neural network,” *Comput. Secur.*, vol. 100, p. 102096, Jan. 2021, doi: 10.1016/j.cose.2020.102096.

[140] G. T. Fox and R. V. Boppana, “On Early Detection of Anomalous Network Flows,” *IEEE Access*, vol. 11, pp. 68588–68603, 2023, doi: 10.1109/ACCESS.2023.3291686.

[141] M. Shahhosseini, H. Mashayekhi, and M. Rezvani, “A Deep Learning Approach for Botnet Detection Using Raw Network Traffic Data,” *J. Netw. Syst. Manag.*, vol. 30, no. 3, p. 44, Jul. 2022, doi: 10.1007/s10922-022-09655-7.

[142] X. Han, S. Cui, S. Liu, C. Zhang, B. Jiang, and Z. Lu, “Network intrusion detection based on n-gram frequency and time-aware transformer,” *Comput. Secur.*, vol. 128, p. 103171, May 2023, doi: 10.1016/j.cose.2023.103171.

[143] W. Wang *et al.*, “HAST-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection,” *IEEE Access*, vol. 6, pp. 1792–1806, 2018, doi: 10.1109/ACCESS.2017.2780250.

[144] X. Zhang, J. Chen, Y. Zhou, L. Han, and J. Lin, “A Multiple-Layer Representation Learning Model for Network-Based Attack Detection,” *IEEE Access*, vol. 7, pp. 91992–92008, 2019, doi: 10.1109/ACCESS.2019.2927465.

[145] Y. Zhu, D. Han, and X. Yin, “A hierarchical network intrusion detection model based on unsupervised clustering,” in *Proceedings of the 13th International Conference on Management of Digital EcoSystems*, Virtual Event Tunisia: ACM, Nov. 2021, pp. 22–29. doi: 10.1145/3444757.3485098.

[146] T. Ahmad and D. Truscan, “Early Detection with Explainability of Network Attacks Using Deep Learning,” in *2024 IEEE International Conference on Software Testing,*

Verification and Validation Workshops (ICSTW), Toronto, ON, Canada: IEEE, May 2024, pp. 161–167. doi: 10.1109/ICSTW60967.2024.00040.

[147] M. M. Islam, T. Ahmad, and D. Truscan, "An Evaluation of Transformer Models for Early Intrusion Detection in Cloud Continuum," in *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Naples, Italy: IEEE, Dec. 2023, pp. 279–284. doi: 10.1109/CloudCom59040.2023.00052.

[148] T. Ahmad, D. Truscan, and J. Vain, "Preliminary Results in Using Attention for Increasing Attack Identification Efficiency," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Dublin, Ireland: IEEE, Apr. 2023, pp. 159–164. doi: 10.1109/ICSTW58534.2023.00038.

[149] T. Ahmad, D. Truscan, J. Vain, and I. Porres, "Early Detection of Network Attacks Using Deep Learning," in *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Valencia, Spain: IEEE, Apr. 2022, pp. 30–39. doi: 10.1109/ICSTW55395.2022.00020.

[150] R. Mohammad, F. Saeed, A. A. Almazroi, F. S. Alsubaei, and A. A. Almazroi, "Enhancing Intrusion Detection Systems Using a Deep Learning and Data Augmentation Approach," *Systems*, vol. 12, no. 3, p. 79, Mar. 2024, doi: 10.3390/systems12030079.

[151] S. Menssouri and E. M. Amhoud, "A Conditional Tabular GAN-Enhanced Intrusion Detection System for Rare Attacks in IoT Networks," Feb. 09, 2025, *arXiv*: arXiv:2502.06031. doi: 10.48550/arXiv.2502.06031.

[152] Y. Zhang and Q. Liu, "On IoT intrusion detection based on data augmentation for enhancing learning on unbalanced samples," *Future Gener. Comput. Syst.*, vol. 133, pp. 213–227, Aug. 2022, doi: 10.1016/j.future.2022.03.007.

[153] S. Alabdulwahab, Y.-T. Kim, and Y. Son, "Privacy-Preserving Synthetic Data Generation Method for IoT-Sensor Network IDS Using CTGAN," *Sensors*, vol. 24, no. 22, p. 7389, Nov. 2024, doi: 10.3390/s24227389.

[154] C. Liu, R. Antypenko, I. Sushko, and O. Zakharchenko, "Intrusion Detection System After Data Augmentation Schemes Based on the VAE and CVAE," *IEEE Trans. Reliab.*, vol. 71, no. 2, pp. 1000–1010, Jun. 2022, doi: 10.1109/TR.2022.3164877.

[155] F. S. Melícias, T. F. R. Ribeiro, C. Rabadão, L. Santos, and R. L. D. C. Costa, "GPT and Interpolation-Based Data Augmentation for Multiclass Intrusion Detection in IIoT," *IEEE Access*, vol. 12, pp. 17945–17965, 2024, doi: 10.1109/ACCESS.2024.3360879.

[156] X. Jiang *et al.*, "NetDiffusion: Network Data Augmentation Through Protocol-Constrained Traffic Generation," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 8, no. 1, pp. 1–32, Feb. 2024, doi: 10.1145/3639037.

[157] N. Sivaroopan, D. Bandara, C. Madarasingha, G. Jourjon, A. P. Jayasumana, and K. Thilakarathna, "NetDiffus: Network traffic generation by diffusion models through time-series imaging," *Comput. Netw.*, vol. 251, p. 110616, Sep. 2024, doi: 10.1016/j.comnet.2024.110616.

[158] Y. Li, S. Si, G. Li, C.-J. Hsieh, and S. Bengio, "Learnable Fourier Features for Multi-dimensional Spatial Positional Encoding," in *Advances in Neural Information*

Processing Systems, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, Eds., Curran Associates, Inc., 2021, pp. 15816–15829.

[159] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, "RoFormer: Enhanced transformer with Rotary Position Embedding," *Neurocomputing*, vol. 568, p. 127063, Feb. 2024, doi: 10.1016/j.neucom.2023.127063.

[160] A. Rashed, S. Elsayed, and L. Schmidt-Thieme, "Context and Attribute-Aware Sequential Recommendation via Cross-Attention," in *Proceedings of the 16th ACM Conference on Recommender Systems*, Seattle WA USA: ACM, Sep. 2022, pp. 71–80. doi: 10.1145/3523227.3546777.

[161] Y. Ma *et al.*, "Non-stationary Time-aware Kernelized Attention for Temporal Event Prediction," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Washington DC USA: ACM, Aug. 2022, pp. 1224–1232. doi: 10.1145/3534678.3539470.

[162] J. Ji, Y. Cao, Y. Ma, and J. Yan, "TITD: enhancing optimized temporal position encoding with time intervals and temporal decay in irregular time series forecasting," *Appl. Intell.*, vol. 55, no. 6, p. 415, Apr. 2025, doi: 10.1007/s10489-025-06293-9.

[163] J. Zhou, G. Jiang, W. Du, and C. Han, "Profiling temporal learning interests with time-aware transformers and knowledge graph for online course recommendation," *Electron. Commer. Res.*, vol. 23, no. 4, pp. 2357–2377, Dec. 2023, doi: 10.1007/s10660-022-09541-z.

[164] H. Ryu, S. Yu, and K. Y. Lee, "TI-former: A Time-Interval Prediction Transformer for Timestamped Sequences," in *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*, Orlando, FL, USA: IEEE, May 2023, pp. 319–325. doi: 10.1109/SERA57763.2023.10197830.

[165] A. Sharma, T. Samon, A. Vellandurai, and V. Kumar, "TA-SAITS: Time Aware-Self Attention based Imputation of Time Series algorithm for Partially Observable Multi - Variate Time Series," in *2023 International Conference on Machine Learning and Applications (ICMLA)*, Jacksonville, FL, USA: IEEE, Dec. 2023, pp. 2228–2233. doi: 10.1109/ICMLA58977.2023.00336.

[166] P. Aitken, B. Claise, and B. Trammell, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," Internet Engineering Task Force, Request for Comments RFC 7011, Sep. 2013. doi: 10.17487/RFC7011.

[167] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, "CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment," *Sensors*, vol. 23, no. 13, 2023, doi: 10.3390/s23135941.

[168] H. Hindy, E. Bayne, M. Bures, R. Atkinson, C. Tachtatzis, and X. Bellekens, "Machine Learning Based IoT Intrusion Detection System: An MQTT Case Study (MQTT-IoT-IDS2020 Dataset)," in *Selected Papers from the 12th International Networking Conference*, vol. 180, in Lecture Notes in Networks and Systems, vol. 180, Cham: Springer International Publishing, 2021, pp. 73–84. doi: 10.1007/978-3-030-64758-2_6.

[169] I. Ullah and Q. H. Mahmoud, "A Scheme for Generating a Dataset for Anomalous Activity Detection in IoT Networks," in *Advances in Artificial Intelligence*, vol.

12109, in Lecture Notes in Computer Science, vol. 12109, Cham: Springer International Publishing, 2020, pp. 508–520. doi: 10.1007/978-3-030-47358-7_52.

[170] D. Fernandez, L. Vigoya, F. Cacheda, F. J. Novoa, M. F. Lopez-Vizcaino, and V. Carneiro, “A Practical Application of a Dataset Analysis in an Intrusion Detection System,” in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA: IEEE, Nov. 2018, pp. 1–5. doi: 10.1109/NCA.2018.8548316.

[171] P. Shaw, J. Uszkoreit, and A. Vaswani, “Self-Attention with Relative Position Representations,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 464–468. doi: 10.18653/v1/N18-2074.

[172] C.-Z. A. Huang *et al.*, “Music Transformer: Generating Music with Long-Term Structure,” presented at the International Conference on Learning Representations, Sep. 2018. Accessed: Mar. 31, 2025.