



ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μελέτη και Ανάπτυξη Ενεργειακά Αποδοτικών Μεθόδων Εκτέλεσης Generative AI (LLMs) σε Ενσωματωμένα Συστήματα Χαμηλών Πόρων

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ονοματεπώνυμο φοιτητή, ΑΕΜ:
Δημήτριος Νεκτάριος Βατούσης, 220007

Επιβλέπων Καθηγητής: Δημήτρης Καραμπατζάκης, Αναπληρωτής Καθηγητής,
Πληροφορικής, Δημοκρίτειο Πανεπιστήμιο Θράκης

Καβάλα, 2026

Μελέτη και Ανάπτυξη Ενεργειακά Αποδοτικών Μεθόδων Εκτέλεσης Generative AI (LLMs) σε Ενσωματωμένα Συστήματα Χαμηλών Πόρων

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ονοματεπώνυμο φοιτητή, ΑΕΜ:
Δημήτριος Νεκτάριος Βατούσης, 220007

Η παρούσα Πτυχιακή Εργασία υποβλήθηκε στο Τμήμα Πληροφορικής του Δημοκρίτειου Πανεπιστημίου Θράκης για την απόκτηση του πτυχίου του Τμήματος.

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Επιβλέπων καθηγητής: Δημήτριος Καραμπατζάκης, Αν. Καθηγητής, Τμήμα Πληροφορικής, ΔΠΘ

2ο Μέλος: Δρ. Πέτρος Αμανατίδης, Διδάσκων Ακαδ. Εμπειρία, Τμήμα Πληροφορικής, ΔΠΘ

3ο Μέλος: Βασίλειος Τσουκαλάς, ΕΤΕΠ, Τμήμα Πληροφορικής, ΔΠΘ

Καβάλα, 2026

Πνευματικά δικαιώματα

Copyright © Dimitris Vatousis, 2026

Η έγκριση της πτυχιακής εργασίας από το Τμήμα Πληροφορικής του Δημοκρίτειου Πανεπιστημίου Θράκης δε δηλώνει απαραίτητως την αποδοχή των απόψεων του/της συγγραφέα/ως.

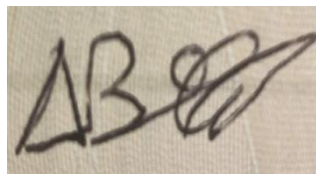
Τήρηση ακαδημαϊκών αρχών και δεοντολογίας

Η παρούσα εργασία είναι πρωτότυπη και εκπονήθηκε εξ ολοκλήρου από τον/την φοιτητή/φοιτήτρια Δημήτρη Βατούση του Προπτυχιακού Προγράμματος Σπουδών του Τμήματος Πληροφορικής, υπό την καθοδήγηση του/της επιβλέποντα/επιβλέπουσας.

Ο/Η φοιτητής/φοιτήτρια βεβαιώνει ότι κατά την εκπόνηση και τη συγγραφή της παρούσας πτυχιακής εργασίας τήρησε τα προβλεπόμενα από τον νόμο, τον εσωτερικό κανονισμό και τον κανονισμό εκπόνησης πτυχιακών εργασιών του Τμήματος, καθώς και ότι σεβάστηκε πλήρως τις Αρχές της Ακαδημαϊκής Ηθικής και του Κώδικα Δεοντολογίας, οι οποίες απαγορεύουν την παραποίηση των ερευνητικών/πειραματικών αποτελεσμάτων, την αναφορά ψευδών στοιχείων, την κατάχρηση της διανοητικής ιδιοκτησίας τρίτων και τη λογοκλοπή.

Ο/Η συγγραφέας

Δημήτρης Βατούσης

A handwritten signature in dark ink on a light-colored, textured background. The signature appears to be 'ΔΒ' followed by a stylized, cursive flourish.



DEMOCRITUS UNIVERSITY OF THRACE
SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS

Research and Development of Energy-Efficient Inference Methods for Generative AI (LLMs) on Low-Power Embedded Systems

UNDERGRADUATE DISSERTATION

Full Name of the author, RN:

Dimitrios Nektarios Vatousis, 220007

A dissertation submitted in partial fulfilment of the requirements for the undergraduate degree in the Department of Informatics, Democritus University of Thrace.

COMMITTEE OF EXAMINERS

Supervisor: Dimitrios Karampatzakis, Associate Professor, Department of Informatics, DUTH

Member 2: Dr. Petros Amanatidis, Adjunct Professor, Department of Informatics, DUTH

Member 3: Vasileios Tsoukalas, ETEP, Department of Informatics, DUTH

Kavala, 2026

All rights reserved

Copyright © Dimitris Vatousis, 2026

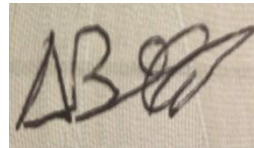
The approval of the undergraduate dissertation by the Department of Informatics, Democritus University of Thrace, does not necessarily indicate the acceptance of the views of the author.

Fulfillment of academic principles and ethics

The present undergraduate dissertation is original and was prepared entirely by the student Dimitris Vatousis of the Undergraduate Program of Studies of the Department of Informatics, under the guidance of the supervisor. The student certifies that during the preparation and writing of this undergraduate dissertation he/she complied with the provisions of the law and the corresponding internal regulations of the Department, as well as that he/she fully respected the Principles of Academic Ethics and the Code of Ethics, which prohibit falsification of research/experimental results, false reporting, misuse of third-party intellectual property and plagiarism.

The author

Dimitris Vatousis

A handwritten signature in dark ink on a light-colored, textured background. The signature appears to be 'ΔΒ' followed by a stylized, cursive flourish.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία επικεντρώνεται στη μελέτη, σύγκριση και ανάπτυξη μεθόδων βελτιστοποίησης για την εκτέλεση μοντέλων Γενετικής Τεχνητής Νοημοσύνης, και συγκεκριμένα Μεγάλων Γλωσσικών Μοντέλων (Large Language Models, LLMs), σε ενσωματωμένα συστήματα χαμηλών πόρων. Αρχικά, πραγματοποιείται εκτενής βιβλιογραφική ανασκόπηση μέσω των βάσεων δεδομένων Scopus και IEEE Xplore, με στόχο τη διερεύνηση σύγχρονων τεχνικών όπως ο κβαντισμός μοντέλων (quantization), η αποκοπή νευρωνικών δικτύων (pruning) και η δυναμική προσαρμογή ακρίβειας (dynamic precision scaling), οι οποίες συμβάλλουν στη βελτίωση της ενεργειακής αποδοτικότητας κατά την εκτέλεση inference.

Η πειραματική διαδικασία βασίζεται στην πλατφόρμα χαμηλής ισχύος Raspberry Pi 400, σε συνδυασμό με τον εξωτερικό μετρητή ισχύος Atorch UD18 για τη μέτρηση της κατανάλωσης ενέργειας σε επίπεδο υλικού. Παράλληλα, αναπτύσσεται λογισμικό πλαίσιο αξιολόγησης και παρακολούθησης της απόδοσης σε πραγματικό χρόνο, υλοποιημένο κυρίως σε Python, αξιοποιώντας τη βιβλιοθήκη llama-cpp-python για την εκτέλεση inference και το psutil για την παρακολούθηση του συστήματος, ενώ οι PyTorch και Hugging Face Transformers χρησιμοποιήθηκαν για την προετοιμασία των μοντέλων.

Κύριος στόχος της εργασίας είναι η διερεύνηση της εφικτότητας, των περιορισμών και των συμβιβασμών απόδοσης που προκύπτουν κατά την εκτέλεση LLMs σε περιβάλλοντα χαμηλής ισχύος, καθώς και η πρόταση στρατηγικών που επιτρέπουν σταθερή, ενεργειακά αποδοτική λειτουργία. Τα πειράματα σε τρία μοντέλα (TinyLlama-1.1B, Llama-3.2-1B, Qwen2-0.5B) και τέσσερις μεθόδους (fp16, INT8 Q8_0, unstructured pruning 30% και DPS) έδειξαν ότι η ισχύς του Raspberry Pi 400 παραμένει σχεδόν σταθερή (8.1 έως 8.5 W), οπότε η ενέργεια καθορίζεται κυρίως από τον χρόνο εκτέλεσης. Ο κβαντισμός INT8 μείωσε την ενέργεια ανά token κατά περίπου 45% σε όλα τα μοντέλα, το unstructured pruning χωρίς επανεκπαίδευση δεν απέφερε όφελος, ενώ η DPS προσαρμόστηκε διαφορετικά σε κάθε μοντέλο (0, 1 και 4 αλλαγές ακρίβειας). Τέλος, το μέγεθος του μοντέλου επηρέασε την ενέργεια περισσότερο από κάθε τεχνική βελτιστοποίησης. Τα αποτελέσματα αποτελούν πειραματική βάση για μελλοντική έρευνα στον τομέα του thermally-aware dynamic precision scaling.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη και Ενσωματωμένα Συστήματα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Μεγάλα Γλωσσικά Μοντέλα, Ενσωματωμένα Συστήματα, Ενεργειακά Αποδοτική Εκτέλεση, Κβαντισμός, Αποκοπή Νευρωνικών Δικτύων, Δυναμική Κλιμάκωση Ακρίβειας

ABSTRACT

This undergraduate thesis focuses on the study, comparison, and development of optimization techniques for the execution of Generative Artificial Intelligence models, specifically Large Language Models (LLMs), on low-resource embedded systems. The work begins with an extensive literature review conducted through the Scopus and IEEE Xplore databases, aiming to analyse state-of-the-art techniques such as model quantization, neural network pruning, and dynamic precision scaling, which contribute to improving energy efficiency during inference.

The experimental procedure is based on the low-power Raspberry Pi 400 platform, combined with an external Atorch UD18 power meter for hardware-level energy measurement. In parallel, a software framework for real-time performance evaluation and monitoring is developed, implemented mainly in Python, using the llama-cpp-python library for inference execution and psutil for system monitoring, while PyTorch and Hugging Face Transformers were used for model preparation.

The primary objective of this thesis is to investigate the feasibility, limitations, and performance trade-offs of executing LLMs in low-power environments, and to propose strategies that enable stable, energy-efficient operation. Experiments with three models (TinyLlama-1.1B, Llama-3.2-1B, Qwen2-0.5B) and four methods (fp16, INT8 Q8_0, 30% unstructured pruning and DPS) showed that the power draw of the Raspberry Pi 400 remains nearly constant (8.1 to 8.5 W), so energy is determined mainly by execution time. INT8 quantization reduced energy per token by about 45% across all models, unstructured pruning without retraining yielded no benefit, and DPS adapted differently to each model (0, 1 and 4 precision switches). Finally, model size affected energy more than any optimization technique. These results provide an experimental foundation for future research on thermally-aware dynamic precision scaling.

SUBJECT AREA: Artificial Intelligence and Embedded Systems

KEYWORDS: Large Language Models, Embedded Systems, Energy Efficient Inference, Quantization, Neural Network Pruning, Dynamic Precision Scaling

ΠΕΡΙΕΧΟΜΕΝΑ

1 ^Ο ΚΕΦΑΛΑΙΟ: ΕΙΣΑΓΩΓΗ	1
1.1 Εισαγωγή Στην Ενεργειακά Αποδοτική Εκτέλεση Generative AI	1
1.2 Κίνητρα Και Σημασία Της Ενεργειακά Αποδοτικής Εκτέλεσης Generative AI	1
1.3 Generative AI Και LLMs: Σύγχρονες Τάσεις Και Προκλήσεις	2
1.4 Εκτέλεση LLMs Σε Ενσωματωμένα Συστήματα Χαμηλών Πόρων	2
1.5 Ανασκόπηση Σχετικής Βιβλιογραφίας	3
1.5.1 Ενεργειακά Αποδοτικό Inference	3
1.5.2 Τεχνικές Βελτιστοποίησης Για Ενσωματωμένα Συστήματα	4
1.6 Περιορισμοί Υλικού Και Λογισμικού Σε Low-Power Περιβάλλοντα	5
2 ^Ο ΚΕΦΑΛΑΙΟ: ΘΕΩΡΗΤΙΚΕΣ ΈΝΝΟΙΕΣ ΚΑΙ ΜΕΘΟΔΟΙ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ	7
2.1 Βασικές Έννοιες Μηχανικής Μάθησης και Νευρωνικών Δικτύων	7
2.2 Large Language Models (LLMs)	7
2.2.1 Αρχιτεκτονική Transformer	7
2.2.2 Εκπαίδευση Και Inference Σε LLMs	8
2.3 Τεχνικές Βελτιστοποίησης για Ενεργειακά Αποδοτικό Inference	9
2.3.1 Κβαντισμός Μοντέλων (Quantization)	9
2.3.2 Αποκοπή Νευρωνικών Δικτύων (Pruning)	10
2.3.3 Dynamic Precision Scaling	11
2.4 Thermally-Aware και Energy-Aware Inference	11
3 ^Ο ΚΕΦΑΛΑΙΟ: ΠΕΡΙΓΡΑΦΗ ΣΥΣΤΗΜΑΤΟΣ ΥΛΙΚΟΥ (HARDWARE)	14
3.1 Υλικό του Συστήματος	14
3.1.1 Χαρακτηριστικά Υλικού	14
3.2 Υπολογιστικοί και Ενεργειακοί Περιορισμοί του Συστήματος	14
3.2.1 Υπολογιστικοί Περιορισμοί	15
3.2.2 Ενεργειακοί Περιορισμοί	15
3.3 Σύστημα Μέτρησης Κατανάλωσης Ενέργειας	16
3.3.1 Περιγραφή του Atorch UD18	16
3.4 Συνδεσμολογία και Διάγραμμα Συστήματος	17
4 ^Ο ΚΕΦΑΛΑΙΟ: ΠΕΡΙΓΡΑΦΗ ΛΟΓΙΣΜΙΚΟΥ (SOFTWARE)	20
4.1 Επιλογή Γλωσσών Προγραμματισμού	20
4.1.1 Python	20
4.1.2 YAML	20
4.2 Βιβλιοθήκες και Πλαίσιο Ανάπτυξης	20
4.2.1 Microsoft Visual Studio Code	21
4.2.2 Βασικές Βιβλιοθήκες Μηχανικής Μάθησης και Inference	21
4.2.3 Βιβλιοθήκες Παρακολούθησης και Μετρήσεων	21
4.2.4 Υποστηρικτικές Βιβλιοθήκες	22
4.3 Περιγραφή Κώδικα	22

4.3.1 run_method.py Επίπεδο: Υψηλού Επιπέδου	22
4.3.2 Προετοιμασία Μοντέλων (Windows PC) Επίπεδο: Χαμηλού Επιπέδου	23
4.3.3 dps_inference (λογική εντός run_method.py) Επίπεδο: Υψηλού Επιπέδου	23
4.3.4 performance_monitor.py Επίπεδο: Χαμηλού Επιπέδου	24
4.3.5 energy_logger.py Επίπεδο: Χαμηλού Επιπέδου	24
4.3.6 metrics.py Επίπεδο: Υψηλού Επιπέδου	25
4.3.7 graphs.py Επίπεδο: Υψηλού Επιπέδου	25
4.3.8 experiment_runner.py & config.yaml Επίπεδο: Υψηλού Επιπέδου	25
4.4 Διαγράμματα και Μοντελοποίηση (UML)	26
4.4.1 Διάγραμμα Περιπτώσεων Χρήσης	26
4.4.2 Διάγραμμα Στοιχείων	28
4.4.3 Διάγραμμα Κλάσης	28
4.5 Παραδείγματα Υλοποίησης Κώδικα & Περιγραφή Εφαρμογής	29
4.5.1 Λογική DPS Switching (run_method.py)	29
4.5.2 Ανάγνωση Αισθητήρα UD18 (energy_logger.py)	31
4.5.3 Παράλληλη Συλλογή Μετρικών (metrics.py)	32
4.5.4 Ορχήστρωση Πειραμάτων (experiment_runner.py)	35
4.5.5 Ενδεικτική Έξοδος Τερματικού	36
5^ο ΚΕΦΑΛΑΙΟ: ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ	38
5.1 Έλεγχος Καλής Λειτουργίας Συστήματος	38
5.1.1 Επαλήθευση Σύνδεσης Αισθητήρα UD18	38
5.1.2 Επαλήθευση Inference	38
5.1.3 Επαλήθευση Εκτέλεσης Πειράματος	40
5.1.4 Επαλήθευση Μηχανισμού DPS	40
5.2 Συνοπτικά Αποτελέσματα Πειράματος	41
5.3 Ανάλυση ανά Μέθοδο	41
5.3.1 Μέθοδος fp16	42
5.3.2 Μέθοδος INT8 (Q8_0)	44
5.3.3 Μέθοδος Pruned	46
5.3.4 Μέθοδος DPS	48
5.4 Συγκριτική Ανάλυση Μεθόδων	51
5.5 Επίδραση Μεγέθους Μοντέλου	52
5.6 Τελικά Συμπεράσματα	52
6^ο ΚΕΦΑΛΑΙΟ: ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΠΡΟΤΑΣΕΙΣ	54
6.1 Συνολικά Συμπεράσματα της Πτυχιακής	54
6.2 Προβλήματα που Αντιμετωπίστηκαν	54
6.2.1 Ανακατασκευή Λογικής DPS	54
6.2.2 Αποκωδικοποίηση Πρωτοκόλλου UD18	55
6.2.3 Διαχείριση Μνήμης στο Raspberry Pi	55
6.3 Κριτική Αξιολόγηση	55
6.3.1 Δυνατά Σημεία της Υλοποίησης	55
6.3.2 Περιορισμοί και Αδυναμίες	56
6.4 Μελλοντικές Προτάσεις και Επεκτάσεις	56
6.4.1 Εφαρμογή σε Μεγαλύτερα LLM	56
6.4.2 DPS σε Ταχύτερο Hardware	57
6.4.3 Έρευνα για Thermally-Aware DPS AI Inference	57

ΑΝΑΦΟΡΕΣ	58
ΠΑΡΑΡΤΗΜΑ 1. ΚΩΔΙΚΑΣ ΠΟΥ ΑΝΑΠΤΥΧΘΗΚΕ	60

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

Εικόνα 1: Atorch UD18	16
Εικόνα 2: Συνδεσμολογία & Διάγραμμα Συστήματος.....	17
Εικόνα 3: Σύστημα.....	18
Εικόνα 4: Διάγραμμα Περιπτώσεων Χρήσης.....	27
Εικόνα 5: Διάγραμμα Στοιχείων	28
Εικόνα 6: Διάγραμμα Κλάσης	28
Εικόνα 7: Επαλήθευση Εκτέλεσης Πειράματος	40
Εικόνα 8: tinyllama fp16 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.....	42
Εικόνα 9: Μέση Χρήση CPU & RAM tinyllama fp16.	42
Εικόνα 10: llama3.2 fp16 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.....	43
Εικόνα 11: Μέση Χρήση CPU & RAM llama3.2 fp16.	43
Εικόνα 12: qwen2 fp16 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.	43
Εικόνα 13: Μέση Χρήση CPU & RAM qwen2 fp16.	44
Εικόνα 14: tinyllama int8 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.....	44
Εικόνα 15: Μέση Χρήση CPU & RAM tinyllama int8.....	44
Εικόνα 16: llama3.2 int8 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.....	45
Εικόνα 17: Μέση Χρήση CPU & RAM llama3.2 int8.	45
Εικόνα 18: qwen2 int8 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.	45
Εικόνα 19: Μέση Χρήση CPU & RAM qwen2 int8.	46
Εικόνα 20: tinyllama pruned Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.....	46
Εικόνα 21: Μέση Χρήση CPU & RAM tinyllama pruned.	46
Εικόνα 22: llama3.2 pruned Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.....	47
Εικόνα 23: Μέση Χρήση CPU & RAM llama3.2 pruned.	47
Εικόνα 24: qwen2 pruned Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.....	47
Εικόνα 25: Μέση Χρήση CPU & RAM qwen2 pruned.....	48
Εικόνα 26: tinyllama dps Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.....	49
Εικόνα 27: Μέση Χρήση CPU & RAM tinyllama dps.....	49
Εικόνα 28: llama3.2 dps Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.....	50

Εικόνα 29: Μέση Χρήση CPU & RAM llama3.2 dps.	50
Εικόνα 30: qwen2 dps Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.	50
Εικόνα 31: Μέση Χρήση CPU & RAM qwen2 dps.	51
Εικόνα 32: Σύγκριση Μεθόδων Σε Όλα Τα Μοντέλα.....	51

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

Πίνακας 1: Χαρακτηριστικά Υλικού	14
Πίνακας 2: Ενδεικτικές αποκρίσεις ανά μοντέλο και μέθοδο.....	38
Πίνακας 3: Timeline εκτέλεσης DPS ανά run του TinyLlama.	40
Πίνακας 4: Συνοπτικά αποτελέσματα πειράματος.	41
Πίνακας 5: Συμπεριφορά DPS ανά μοντέλο	48
Πίνακας 6: Σχετική μεταβολή J/token έναντι fp16 baseline.....	51

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	Ελληνικός Όρος
Generative Artificial Intelligence	Γενετική Τεχνητή Νοημοσύνη
Large Language Model	Μεγάλο Γλωσσικό Μοντέλο
Inference	Εκτέλεση / Εξαγωγή Πρόβλεψης
Quantization	Κβαντισμός
Pruning	Αποκοπή Νευρωνικών Δικτύων
Dynamic Precision Scaling	Δυναμική Κλιμάκωση Ακρίβειας
Embedded System	Ενσωματωμένο Σύστημα
Energy-Efficient	Ενεργειακά Αποδοτικό
Thermal Throttling	Θερμικός Περιορισμός
Memory Footprint	Αποτύπωμα Μνήμης
Knowledge Distillation	Απόσταξη Γνώσης
Foundation Model	Θεμελιώδες Μοντέλο
Fine-tuning	Λεπτή Ρύθμιση
Pre-training	Προ-εκπαίδευση
Sparsity	Αραιότητα
Activation Function	Συνάρτηση Ενεργοποίησης
Backpropagation	Οπισθόδρομη Μετάδοση Σφάλματος
Overfitting	Υπεραρμογή
Attention Mechanism	Μηχανισμός Προσοχής
Decoder	Αποκωδικοποιητής
Encoder	Κωδικοποιητής
Latency	Καθυστέρηση
Throughput	Απόδοση / Ρυθμαπόδοση
Benchmark	Δοκιμή Αξιολόγησης
Layer	Επίπεδο (Νευρωνικού Δικτύου)
Batch Size	Μέγεθος Δέσμης
Learning Rate	Ρυθμός Εκμάθησης
Token	Λεκτική Μονάδα
Mixed Precision	Μικτή Ακρίβεια
Weight	Βάρος (Παράμετρος Μοντέλου)

ANN	Artificial Neural Network
API	Application Programming Interface
ARM	Advanced RISC Machine
AWQ	Activation-Aware Weight Quantization
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
DPS	Dynamic Precision Scaling
ΔΠΘ	Δημοκρίτειο Πανεπιστήμιο Θράκης
DVFS	Dynamic Voltage and Frequency Scaling
FFN	Feed-Forward Network
FP16	16-bit Floating Point
FP32	32-bit Floating Point
FP8	8-bit Floating Point
GPT	Generative Pre-trained Transformer
GPTQ	Generative Pre-trained Transformer Quantization
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
INT4	4-bit Integer
INT8	8-bit Integer
IoT	Internet of Things
KV-cache	Key-Value Cache
LLaMA	Large Language Model Meta AI
LLM	Large Language Model
LoRA	Low-Rank Adaptation
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
PMPD	Progressive Mixed-Precision Decoding

PTQ	Post-Training Quantization
QAT	Quantization-Aware Training
RAM	Random Access Memory
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
RoPE	Rotary Position Embedding
SGD	Stochastic Gradient Descent
SLO	Service Level Objective
TAPAS	Thermal- and Power-Aware Scheduling
TAWS	Thermally-Aware Workload Scheduling
TPU	Tensor Processing Unit
TTFT	Time to First Token
UML	Unified Modeling Language
vLLM	Virtual Large Language Model serving framework
YAML	YAML Ain't Markup Language

1^ο Κεφάλαιο: Εισαγωγή

1.1 Εισαγωγή Στην Ενεργειακά Αποδοτική Εκτέλεση Generative AI

Τα τελευταία χρόνια έχει υπάρξει ραγδαία εξέλιξη στον τομέα της τεχνητής νοημοσύνης, με τα μοντέλα Γενετικής Τεχνητής Νοημοσύνης (Generative Artificial Intelligence), και ιδιαίτερα τα Μεγάλα Γλωσσικά Μοντέλα (Large Language Models, LLMs), να βρίσκονται στο επίκεντρο. Τα μοντέλα αυτά χρησιμοποιούν αρχιτεκτονική βασισμένη σε αυτή των Transformers και έχουν επιδείξει ιδιαίτερη ικανότητα σε εργασίες επεξεργασίας φυσικής γλώσσας, όπως η παραγωγή κειμένου, η μετάφραση και η παραγωγή κώδικα, καθιστώντας τα θεμελιώδη εργαλεία για σύγχρονες έξυπνες εφαρμογές.

Τα LLMs έχουν αναδείξει εντυπωσιακές τεχνικές δυνατότητες με κόστος, βέβαια, αυξημένες απαιτήσεις σε υπολογιστικούς πόρους, μνήμη και κατανάλωση ενέργειας. Ο μεγάλος αριθμός παραμέτρων, ο μηχανισμός αυτοπροσοχής (self-attention) και η ανάγκη επεξεργασίας μεγάλων ακολουθιών δεδομένων καθιστούν την εκτέλεση (Inference) των μοντέλων αυτών ιδιαίτερα ενεργοβόρα, ακόμη και σε σύγχρονες υπολογιστικές πλατφόρμες υψηλών επιδόσεων. Παρατηρούμε, επομένως, ότι η ενεργειακή αποδοτικότητα είναι ένας κρίσιμος παράγοντας για τη βιώσιμη και κλιμακούμενη αξιοποίηση της Generative AI.

Το πρόβλημα γίνεται ακόμα πιο εμφανές όταν εξετάζεται η εκτέλεση LLMs σε ενσωματωμένα συστήματα χαμηλών πόρων (Low-Power Embedded Systems), όπως single-board computers, συσκευές στην άκρη του δικτύου και μικροελεγκτές. Τα συστήματα αυτά χαρακτηρίζονται από περιορισμένη υπολογιστική ισχύ, μικρή διαθέσιμη μνήμη, αυστηρά ενεργειακά όρια και συχνά αυξημένους θερμικούς περιορισμούς. Παρόλα αυτά, η αυξανόμενη ανάγκη για τοπική επεξεργασία δεδομένων, μειωμένο latency και προστασία της ιδιωτικότητας καθιστά αρκετά επιθυμητή ή ακόμα και αναγκαία την εκτέλεση Generative AI απευθείας στο edge.

Στο πλαίσιο αυτό, η ενεργειακά αποδοτική εκτέλεση (energy-efficient inference) των LLMs αποτελεί ένα ενεργό και ιδιαίτερα σημαντικό ερευνητικό πεδίο. Η ανάπτυξη και εφαρμογή τεχνικών που μειώνουν την κατανάλωση ενέργειας, χωρίς σημαντική υποβάθμιση της ακρίβειας και της λειτουργικότητας των μοντέλων, είναι καθοριστική για την πρακτική αξιοποίηση της Generative AI σε περιβάλλοντα χαμηλής ισχύος.

Η παρούσα εργασία τοποθετείται σε αυτό το ερευνητικό πλαίσιο, διερευνώντας μεθόδους και στρατηγικές που επιτρέπουν την αποδοτική και θερμικά ενήμερη εκτέλεση LLMs σε ενσωματωμένα συστήματα χαμηλών πόρων.

1.2 Κίνητρα Και Σημασία Της Ενεργειακά Αποδοτικής Εκτέλεσης Generative AI

Η αυξανόμενη υιοθέτηση των LLMs έχει μετατρέψει ριζικά τον τρόπο με τον οποίο σχεδιάζονται και υλοποιούνται σύγχρονες έξυπνες εφαρμογές. Αρχικά, τα μοντέλα αυτά αξιοποιήθηκαν κυρίως σε υποδομές υπολογιστικού νέφους (Cloud Computing), όπου οι αυξημένες απαιτήσεις τους μπορούσαν να μετριαστούν λόγω της μεγάλης διαθεσιμότητας υπολογιστικών πόρων. Ωστόσο, σύγχρονες τάσεις έχουν αυξήσει την ανάγκη για μεταφορά της επεξεργασίας στην άκρη του δικτύου, δημιουργώντας προκλήσεις και καθιστώντας, έτσι, αναγκαία τη μελέτη της εκτέλεσής τους σε περιβάλλοντα χαμηλών πόρων [1].

Ένα από τα σημαντικότερα εμπόδια στην υλοποίηση Generative AI σε ενσωματωμένα συστήματα είναι η κατανάλωση ενέργειας και συνεπώς οι θερμικοί περιορισμοί. Η εκτέλεση υπολογιστικά απαιτητικών μοντέλων σε συσκευές που λειτουργούν με μπαταρία ή διαθέτουν περιορισμένες δυνατότητες απαγωγής θερμότητας, μπορεί να οδηγήσει σε θερμικό περιορισμό

(thermal throttling), μείωση της απόδοσης ή ακόμα και αστάθεια του συστήματος. Παράλληλα, η αυξημένη ενεργειακή κατανάλωση ανεβάζει το λειτουργικό κόστος των συστημάτων, ιδίως σε εφαρμογές μεγάλης κλίμακας.

Παρότι υπάρχει πλήθος βιβλιογραφίας πάνω στην βελτιστοποίηση χρήσης LLMs σε συσκευές χαμηλών πόρων, είναι εμφανής η έλλειψη πηγών πάνω σε ενεργειακά και θερμικά αποδοτικές μεθόδους εκτέλεσης. Η παρούσα πτυχιακή έρχεται να καλύψει αυτό το κενό καθώς τέτοιες μέθοδοι συμβάλλουν σημαντικά στην επέκταση της Generative AI σε περιβάλλοντα στην άκρη του δικτύου, αυξάνοντας την αυτονομία, αξιοπιστία και βιωσιμότητα μελλοντικών low-power συσκευών.

1.3 Generative AI Και LLMs: Σύγχρονες Τάσεις Και Προκλήσεις

Τα Μεγάλα Γλωσσικά Μοντέλα του σήμερα βασίζονται σε αρχιτεκτονικές Transformer με εκατοντάδες εκατομμύρια έως εκατοντάδες δισεκατομμύρια παραμέτρους, πράγμα το οποίο τους επιτρέπει την εκμάθηση γενικών γλωσσικών αναπαραστάσεων και την προσαρμογή τους σε πληθώρα διαφορετικών εργασιών. Μοντέλα σαν αυτά βρίσκονται στον πυρήνα της ανάπτυξης της Generative AI, με αποτέλεσμα τη ραγδαία εξέλιξή τους τόσο στην πολυπλοκότητα όσο και στην κλίμακα. Η τάση στην χρήση των LLMs έχει οδηγήσει στη δημιουργία θεμελιωδών μοντέλων (foundation models), τα οποία λειτουργούν ως γενικού σκοπού μηχανές επεξεργασίας φυσικής γλώσσας.[2]

Με την αύξηση της κλίμακας παρατηρείται και αύξηση στον τρόπο αξιοποίησης των LLMs. Πέρα από τις εφαρμογές cloud-based υπηρεσιών υπάρχει, πλέον, μεγάλο ενδιαφέρον για τοπική εκτέλεση (on-device inference) και edge εφαρμογές, λόγω των πλεονεκτημάτων που συσχετίζονται με αυτά. Η μετατόπιση αυτή έχει στόχο την μειωμένη καθυστέρηση (latency), την προστασία της ιδιωτικότητας των δεδομένων και την βελτίωση της αξιοπιστίας των συστημάτων. Οι τάσεις αυτές καθιστούν τα LLMs ως βασικό στοιχείο σε έξυπνες συσκευές, συστήματα IoT (Internet of Things) και αυτόνομα συστήματα, διευρύνοντας σημαντικά το πεδίο εφαρμογών τους.

Από την άλλη, οι εξελίξεις αυτές συνοδεύονται από ιδιαίτερες τεχνικές προκλήσεις. Η εκτέλεση inference σε LLMs χαρακτηρίζεται από υψηλό υπολογιστικό φόρτο, αυξημένες απαιτήσεις μνήμης και έντονη κατανάλωση ενέργειας, ιδιαίτερα λόγω των πράξεων matrix multiplication και του μηχανισμού self-attention. Σε low-power περιβάλλοντα, όπως αυτά που αφορά η έρευνά μας, οι αυξημένες αυτές απαιτήσεις οδηγούν συχνά σε μειωμένη απόδοση, αυξημένη θερμική καταπόνηση και περιορισμένη βιωσιμότητα της συνεχούς λειτουργίας. Το χάσμα μεταξύ της συνεχώς αυξανόμενης πολυπλοκότητας των LLMs και των περιορισμένων δυνατοτήτων των ενσωματωμένων συστημάτων καθιστά αναγκαία την ανάπτυξη νέων προσεγγίσεων βελτιστοποίησης.

Μελέτες για τις σύγχρονες τάσεις της Generative AI υποστηρίζουν [3] ότι η απλή εφαρμογή υπαρχόντων μοντέλων σε πλατφόρμες περιορισμένων πόρων δεν είναι εφικτή χωρίς σημαντικούς συμβιβασμούς. Επομένως, η μελέτη τεχνικών που μειώνουν το ενεργειακό και θερμικό αποτύπωμα των LLMs αποτελεί κρίσιμο παράγοντα για τη βιώσιμη ενσωμάτωσή τους σε μελλοντικά embedded και edge συστήματα.

1.4 Εκτέλεση LLMs Σε Ενσωματωμένα Συστήματα Χαμηλών Πόρων

Η εκτέλεση Μεγάλων Γλωσσικών Μοντέλων σε ενσωματωμένα συστήματα χαμηλών πόρων απέχει αρκετά από την εκτέλεσή τους σε περιβάλλοντα υπολογιστικού νέφους ή σε υποδομές

υψηλών επιδόσεων. Με τον όρο ενσωματωμένα συστήματα εννοούμε πλατφόρμες όπως single-board computers, στις οποίες κατατάσσεται το Raspberry Pi 400, edge συσκευές που βασίζονται σε αρχιτεκτονικές ARM, καθώς και μικροελεγκτές, οι οποίοι χαρακτηρίζονται από περιορισμένη υπολογιστική ισχύ, μικρή διαθέσιμη μνήμη και αυστηρούς ενεργειακούς περιορισμούς[3].

Σε αντίθεση με τα data centers, όπου η εκτέλεση LLMs επιταχύνεται μέσω hardware accelerators όπως GPUs και TPUs και υποστηρίζεται από επαρκή συστήματα ψύξης και υψηλό ενεργειακό προϋπολογισμό, τα embedded συστήματα βασίζονται συνήθως σε γενικής χρήσης CPUs, με περιορισμένο αριθμό πυρήνων και χαμηλό θερμικό περιθώριο λειτουργίας[4]. Η απουσία εξειδικευμένου hardware επιτάχυνσης καθιστά την εκτέλεση πράξεων υψηλής πολυπλοκότητας, όπως οι πράξεις πολλαπλασιασμού πινάκων και ο μηχανισμός self-attention, ιδιαίτερα απαιτητική.

Ένας ακόμα σημαντικός περιορισμός είναι το αποτύπωμα μνήμης (memory footprint) των LLMs, καθώς η αποθήκευση των παραμέτρων ενός μοντέλου και των ενδιάμεσων ενεργοποιήσεων κατά το inference μπορεί να υπερβαίνει την διαθέσιμη RAM των low-power συστημάτων. Το πρόβλημα ενισχύεται σε εφαρμογές πραγματικού χρόνου, διότι απαιτείται χαμηλή καθυστέρηση απόκρισης και σταθερή απόδοση. Σε τέτοια περιβάλλοντα, η αυξημένη κατανάλωση ενέργειας οδηγεί συχνά σε θερμική επιβάρυνση και ενεργοποίηση των μηχανισμών thermal throttling, οι οποίοι μειώνουν την συχνότητα λειτουργίας του επεξεργαστή και επηρεάζουν αρνητικά την συνολική απόδοση.

Από τους παραπάνω περιορισμούς συμπεραίνουμε ότι η απλή μεταφορά ενός LLM από περιβάλλον cloud σε embedded πλατφόρμα δεν είναι εφικτή χωρίς σημαντικές τροποποιήσεις. Απαιτούνται τεχνικές βελτιστοποίησης προσαρμοσμένες στα χαρακτηριστικά του κάθε hardware με στόχο την μείωση της κατανάλωσης ενέργειας, του memory footprint και παράλληλα την διατήρηση αποδεκτού επιπέδου ακρίβειας. Η συστηματική μελέτη της συμπεριφοράς των LLMs σε low-power πλατφόρμες αποτελεί, συνεπώς, κρίσιμο βήμα για την πρακτική αξιοποίησή τους σε εφαρμογές edge Τεχνητής Νοημοσύνης.

1.5 Ανασκόπηση Σχετικής Βιβλιογραφίας

Στην ενότητα αυτή παρουσιάζεται μια συστηματική ανασκόπηση της υπάρχουσας βιβλιογραφίας που σχετίζεται με τους βασικούς άξονες της παρούσας εργασίας. Συγκεκριμένα, εξετάζονται δύο θεματικές περιοχές: η ενεργειακή αποδοτικότητα κατά την εκτέλεση μοντέλων γλωσσικής επεξεργασίας, και οι τεχνικές βελτιστοποίησης που έχουν προταθεί για την εκτέλεσή τους σε ενσωματωμένα συστήματα περιορισμένων πόρων. Μέσα από την ανάλυση των υφιστάμενων ερευνητικών εργασιών, αναδεικνύονται τόσο τα επιτεύγματα της τρέχουσας έρευνας όσο και τα ανοιχτά ζητήματα που δεν έχουν ακόμη αντιμετωπιστεί, τα οποία και αποτελούν το κίνητρο για την παρούσα μελέτη.

1.5.1 Ενεργειακά Αποδοτικό Inference

Με τον όρο ενεργειακά αποδοτικό inference αναφερόμαστε στην βελτιστοποίηση της διαδικασίας εξαγωγής προβλέψεων ενός machine learning μοντέλου, με σκοπό τη μείωση της κατανάλωσης ενέργειας, διατηρώντας παράλληλα αποδεκτά επίπεδα τόσο στην απόδοση όσο και στην ακρίβεια. Σε αντίθεση με την απλή επιτάχυνση της εκτέλεσης, η ενεργειακή αποδοτικότητα αξιολογείται μέσω μετρικών όπως η ενέργεια ανά εκτέλεση (joules per inference), η ενέργεια ανά παραγόμενο token (energy per token), και η απόδοση ανά watt (performance per watt).

Η ενεργειακή ανάλυση των LLMs έχει ξεχωρίσει ως κρίσιμο ερευνητικό ζήτημα στην διεθνή βιβλιογραφία, ιδιαίτερα στο πλαίσιο μεταφοράς τους από περιβάλλοντα υψηλών επιδόσεων σε

edge συστήματα. Η εργασία των Islam et al. [5] εξετάζει το ενεργειακό αποτύπωμα μικρών γλωσσικών μοντέλων σε edge πλατφόρμες, επισημαίνοντας ότι το inference μπορεί να παρουσιάζει σημαντικές διακυμάνσεις κατανάλωσης ισχύος, ακόμη και σε μοντέλα μειωμένου μεγέθους. Τα αποτελέσματα δείχνουν ότι η μείωση του αριθμού παραμέτρων δεν συνεπάγεται γραμμική μείωση της ενεργειακής κατανάλωσης, γεγονός που αναδεικνύει την πολυπλοκότητα της σχέσης μεταξύ αρχιτεκτονικής και αποδοτικότητας. Την ίδια στιγμή, η μελέτη των Martínez et al. [6] επικεντρώνεται στην ανάλυση κβαντισμένου (quantised) inference σε επεξεργαστές χαμηλής ισχύος, αναδεικνύοντας τη σημαντική επίδραση της ακρίβειας αριθμητικής αναπαράστασης στη συνολική κατανάλωση ενέργειας.

Τα αποτελέσματα της έρευνας δείχνουν ότι η χρήση μειωμένης ακρίβειας (πχ. INT8) μπορεί να οδηγήσει σε σημαντική βελτίωση της ενεργειακής απόδοσης, χωρίς ιδιαίτερη υποβάθμιση των αποτελεσμάτων, ιδίως σε transformer-based αρχιτεκτονικές.

Συνολικά, η υπάρχουσα βιβλιογραφία υποστηρίζει ότι το ενεργειακό κόστος των LLMs δεν οφείλεται απλώς στο μέγεθος του μοντέλου, αλλά ότι επηρεάζεται και από άλλους παράγοντες, συγκεκριμένα το είδος του hardware και την στρατηγική, ή αλλιώς την τεχνική, εκτέλεσης. Ωστόσο, οι περισσότερες έρευνες εστιάζουν είτε σε μικρότερα μοντέλα, είτε σε συγκεκριμένες τεχνικές βελτιστοποίησης, αντί για μια σφαιρική προσέγγιση που να συνδυάζει ενεργειακή και θερμική ανάλυση σε πραγματικά ενσωματωμένα περιβάλλοντα. Το κενό αυτό καθιστά αναγκαία την περαιτέρω διερεύνηση ενεργειακά αποδοτικών στρατηγικών εκτέλεσης LLMs σε low-power συστήματα.

1.5.2 Τεχνικές Βελτιστοποίησης Για Ενσωματωμένα Συστήματα

Η εκτέλεση LLMs σε ενσωματωμένα συστήματα χαμηλών πόρων απαιτεί την εφαρμογή εξειδικευμένων τεχνικών βελτιστοποίησης, οι οποίες στοχεύουν στη μείωση της κατανάλωσης ενέργειας, του αποτυπώματος μνήμης και του χρόνου εκτέλεσης, διατηρώντας παράλληλα αποδεκτή απόδοση του μοντέλου. Οι κυριότερες κατηγορίες τεχνικών που έχουν προταθεί στη βιβλιογραφία περιλαμβάνουν τον κβαντισμό (quantization), την αποκοπή νευρωνικών δικτύων (pruning), τη μεταφορά γνώσης (knowledge distillation) και τις δυναμικές στρατηγικές προσαρμογής ακρίβειας[7][8].

Η πρώτη τεχνική, ο κβαντισμός (quantisation), θεωρείται πλέον μία από τις πιο διαδεδομένες τεχνικές βελτιστοποίησης για συστήματα χαμηλών πόρων. Στο quantisation μειώνεται η αριθμητική ακρίβεια των βαρών και ενεργοποιήσεων (για παράδειγμα από FP16 σε INT4), επιτυγχάνοντας έτσι σημαντική μείωση του αποτυπώματος μνήμης και της απαιτούμενης υπολογιστικής ενέργειας. Η μελέτη των Martínez et al. [6] δείχνει ότι η κβαντισμένη εκτέλεση transformer-based μοντέλων σε επεξεργαστές χαμηλής ισχύος μπορεί να οδηγήσει σε αισθητή βελτίωση της ενεργειακής αποδοτικότητας, ιδιαίτερα όταν αξιοποιούνται κατάλληλες vectorized υλοποιήσεις.

Η επόμενη τεχνική είναι η αποκοπή (pruning), η οποία στοχεύει στην μείωση του αριθμού των παραμέτρων ενός μοντέλου μέσω της αφαίρεσης μη κρίσιμων συνδέσεων ή ολόκληρων δομικών στοιχείων. Οι προσεγγίσεις αυτές μπορεί να είναι μη δομημένες (unstructured sparsity) ή δομημένες (structured pruning), με τη δεύτερη να είναι συνήθως πιο φιλική προς το hardware.

Αν και η αποκοπή μπορεί να μειώσει το μέγεθος του μοντέλου και τη θεωρητική πολυπλοκότητα, η πρακτική ενεργειακή βελτίωση εξαρτάται από το κατά πόσο το hardware και το λογισμικό μπορούν να αξιοποιήσουν αποτελεσματικά τη δημιουργούμενη αραιότητα.

Τέλος, η τεχνική δυναμικής προσαρμογής ακρίβειας (dynamic precision scaling) λειτουργεί προσαρμόζοντας την αριθμητική ακρίβεια κατά την διάρκεια της εκτέλεσης, με βάση το φορτίο του συστήματος ή τους θερμικούς περιορισμούς. Η προσέγγιση αυτή συνδέει άμεσα την

Δημήτρης Βατούσης

ενεργειακή αποδοτικότητα με τη θερμική συμπεριφορά της πλατφόρμας και αποτελεί ιδιαίτερα υποσχόμενη κατεύθυνση για συστήματα με αυστηρό ενεργειακό προϋπολογισμό.

Συνολικά, οι τεχνικές βελτιστοποίησης για ενσωματωμένα συστήματα δεν λειτουργούν μεμονωμένα, αλλά συχνά συνδυάζονται, προκειμένου να επιτευχθεί βέλτιστη ισορροπία μεταξύ ακρίβειας, απόδοσης και κατανάλωσης ενέργειας. Η επιλογή της κατάλληλης στρατηγικής εξαρτάται τόσο από τα χαρακτηριστικά του μοντέλου όσο και από τους περιορισμούς του εκάστοτε hardware περιβάλλοντος.

1.6 Περιορισμοί Υλικού Και Λογισμικού Σε Low-Power Περιβάλλοντα

Η εκτέλεση Μεγάλων Γλωσσικών Μοντέλων σε συστήματα χαμηλής ισχύος επηρεάζεται άμεσα από ένα σύνολο περιορισμών τόσο σε επίπεδο υλικού όσο και σε επίπεδο λογισμικού. Οι περιορισμοί αυτοί καθορίζουν τα όρια απόδοσης, ενεργειακής κατανάλωσης και θερμικής συμπεριφοράς του συστήματος και αποτελούν βασικούς παράγοντες κατά τον σχεδιασμό ενεργειακά αποδοτικών μεθόδων inference.

Οι περιορισμοί στο επίπεδο υλικού προκύπτουν διότι οι low-power πλατφόρμες βασίζονται συνήθως σε επεξεργαστές γενικού σκοπού και χαμηλής κατανάλωσης, με περιορισμένη συχνότητα λειτουργίας και μικρό αριθμό πυρήνων.

Η έλλειψη εξειδικευμένων επιταχυντών, όπως GPUs ή TPUs, καθιστά τον μηχανισμό self-attention και τις πράξεις πολλαπλασιασμού πινάκων ιδιαίτερα απαιτητικές, τόσο σε χρόνο όσο και σε ενέργεια. Ένας ακόμα περιορισμός είναι η έλλειψη επαρκούς μνήμης RAM, με αποτέλεσμα αυστηρούς περιορισμούς στο μέγεθος των μοντέλων και στις ενδιάμεσες δομές δεδομένων κατά το inference.

Από την άλλη, οι περιορισμοί λογισμικού αφορούν κυρίως προβλήματα όπως ο περιορισμός της αποδοτικότητας λόγω της προεπιλεγμένης αριθμητικής κινητής υποδιαστολής υψηλής ακρίβειας (πχ. FP16), το overhead των δυναμικών γλωσσών προγραμματισμού και οι περιορισμοί των backend υλοποιήσεων. Επίσης, η εκτέλεση μοντέλων μέσω γενικών βιβλιοθηκών μηχανικής μάθησης (machine learning) σε CPU περιβάλλοντα συχνά δεν αξιοποιεί πλήρως τις δυνατότητες του hardware. Επιπλέον, σε συστήματα με περιορισμένους πόρους η διαχείριση της μνήμης και ο προγραμματισμός διεργασιών από το λειτουργικό σύστημα ενδέχεται να δημιουργήσουν μεταβλητότητα στην απόδοση.

Αξίζει επίσης να αναφέρουμε και τους ενεργειακούς περιορισμούς, καθώς είναι ένα από τα κύρια προβλήματα που αντιμετωπίζουν τα low-power συστήματα. Σε συστήματα που λειτουργούν με χαμηλό ενεργειακό προϋπολογισμό ή τροφοδοτούνται από μπαταρία, η στιγμιαία αύξηση κατανάλωσης κατά την εκτέλεση ενός LLM μπορεί να οδηγήσει σε μείωση της αυτονομίας ή σε ενεργοποίηση μηχανισμών διαχείρισης ισχύος.

Επιπλέον, η περιορισμένη δυνατότητα απαγωγής θερμότητας σημαίνει ότι η παρατεταμένη εκτέλεση υπολογιστικά απαιτητικών διεργασιών ενδέχεται να προκαλέσει θερμικό περιορισμό (thermal throttling), μειώνοντας δυναμικά την συχνότητα λειτουργίας του επεξεργαστή και επηρεάζοντας την σταθερότητα της απόδοσης.

Η κατανόηση αυτών των περιορισμών είναι απαραίτητη για την ανάπτυξη κατάλληλων τεχνικών βελτιστοποίησης, καθώς η δημιουργία ενεργειακά αποδοτικών μεθόδων εκτέλεσης LLMs απαιτεί προσέγγιση που λαμβάνει υπόψη την σχέση μεταξύ αρχιτεκτονικής μοντέλου, υλικού και λογισμικού.

2^ο Κεφάλαιο: Θεωρητικές Έννοιες Και Μέθοδοι Βελτιστοποίησης

2.1 Βασικές Έννοιες Μηχανικής Μάθησης και Νευρωνικών Δικτύων

Η Μηχανική Μάθηση (Machine Learning, ML) αποτελεί έναν κλάδο της Τεχνητής Νοημοσύνης που επιτρέπει σε υπολογιστικά συστήματα να αποκτούν γνώση από δεδομένα, χωρίς να έχουν προγραμματιστεί ρητά για κάθε εργασία. Τα Τεχνητά Νευρωνικά Δίκτυα (Artificial Neural Networks, ANNs) αποτελούν την κεντρική υπολογιστική δομή των σύγχρονων συστημάτων ML, εμπνευσμένα από τη βιολογική αρχιτεκτονία του ανθρώπινου εγκεφάλου.

Ένα νευρωνικό δίκτυο αποτελείται από επίπεδα (layers) τεχνητών νευρώνων: το επίπεδο εισόδου, ένα ή περισσότερα κρυφά επίπεδα (hidden layers) και το επίπεδο εξόδου. Κάθε νευρώνας λαμβάνει ένα σταθμισμένο άθροισμα εισόδων, εφαρμόζει μια συνάρτηση ενεργοποίησης (activation function) και διαβιβάζει το αποτέλεσμα στο επόμενο επίπεδο. Τα Βαθιά Νευρωνικά Δίκτυα (Deep Neural Networks, DNNs) επεκτείνουν αυτή την αρχιτεκτονία με πολλαπλά κρυφά επίπεδα, επιτρέποντας την αναπαράσταση εξαιρετικά σύνθετων μη γραμμικών συναρτήσεων[9].

Η εκπαίδευση ενός νευρωνικού δικτύου επιτυγχάνεται μέσω του αλγορίθμου οπισθόδρομης μετάδοσης του σφάλματος (backpropagation), ο οποίος υπολογίζει τις μερικές παραγώγους της συνάρτησης κόστους ως προς κάθε εκπαιδευόμενο παράμετρο, και στη συνέχεια ενημερώνει τα βάρη μέσω αλγορίθμων βαθμίδωσης, όπως ο Stochastic Gradient Descent (SGD) ή ο Adam. Βασικές έννοιες που διαμορφώνουν την εκπαίδευση περιλαμβάνουν τον ρυθμό εκμάθησης (learning rate), το batch size, τον αριθμό εποχών (epochs) και την κανονικοποίηση (regularization) για την αντιμετώπιση της υπεραρμογής (overfitting).

Οι Συνελικτικές Νευρωνικές Αρχιτεκτονικές (Convolutional Neural Networks, CNNs) εισήγαγαν ιεραρχική εξαγωγή χαρακτηριστικών για εφαρμογές απεικόνισης, ενώ οι Αναδρομικές Αρχιτεκτονικές (Recurrent Neural Networks, RNNs) και οι παραλλαγές τους LSTM/GRU χρησιμοποιήθηκαν ευρέως για σειριακά δεδομένα, όπως η φυσική γλώσσα.

Ωστόσο, οι περιορισμοί των RNNs ως προς την παράλληλη επεξεργασία και τη μακροχρόνια εξάρτηση δεδομένων οδήγησαν στην ανάπτυξη της αρχιτεκτονικής Transformer, η οποία παρουσιάζεται στην επόμενη ενότητα.

2.2 Large Language Models (LLMs)

Τα Large Language Models (LLMs) αποτελούν μια κατηγορία μεγάλων νευρωνικών δικτύων βαθιάς μάθησης, εκπαιδευμένων σε τεράστιες συλλογές κειμένου με σκοπό την κατανόηση και τη δημιουργία φυσικής γλώσσας[9]. Αρχιτεκτονικά, βασίζονται στο μοντέλο Transformer και αριθμούν από εκατοντάδες εκατομμύρια έως δισεκατομμύρια παραμέτρους. Ορόσημα όπως το GPT (Generative Pre-trained Transformer) της OpenAI, το LLaMA της Meta και το BERT της Google έχουν θέσει νέα πρότυπα απόδοσης σε εργασίες Επεξεργασίας Φυσικής Γλώσσας (NLP).

Η εκθετική αύξηση του μεγέθους αυτών των μοντέλων ανεβάζει ταυτόχρονα τις απαιτήσεις σε υπολογιστικούς πόρους και ενέργεια, ιδίως κατά τη φάση του inference.

2.2.1 Αρχιτεκτονική Transformer

Η αρχιτεκτονική Transformer εισήχθη το 2017 από τους Vaswani et al. στην εργασία «Attention is All You Need»[10] και αποτελεί τη θεμελιώδη δομή των σύγχρονων LLMs. Αντικατέστησε τα αναδρομικά δίκτυα στη μοντελοποίηση ακολουθιών, εισάγοντας έναν νέο μηχανισμό που επεξεργάζεται ολόκληρες ακολουθίες παράλληλα.

Η βασική αρχιτεκτονική Transformer αποτελείται από ένα τμήμα κωδικοποίησης (encoder) και ένα τμήμα αποκωδικοποίησης (decoder). Κάθε encoder layer περιλαμβάνει: (α) ένα μηχανισμό multi-head self-attention, (β) ένα feed-forward νευρωνικό δίκτυο (FFN/MLP), (γ) residual connections και (δ) layer normalization.

Τα σύγχρονα γενετικά LLMs (π.χ. GPT-4, LLaMA) χρησιμοποιούν κυρίως αρχιτεκτονική αποκωδικοποιητή (decoder-only). Ο μηχανισμός self-attention είναι η καρδιά του Transformer[11]. Για κάθε token στην ακολουθία εισόδου, υπολογίζει ένα διάνυσμα Query (Q), Key (K) και Value (V) μέσω εκπαιδευμένων μητρών προβολής. Η προσοχή (attention) υπολογίζεται ως:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{dk}) \cdot V$$

όπου dk είναι η διάσταση των διανυσμάτων Key. Ο μηχανισμός multi-head attention εκτελεί παράλληλα πολλαπλές λειτουργίες self-attention («κεφαλές»), επιτρέποντας στο μοντέλο να παρακολουθεί διαφορετικές σχέσεις μεταξύ tokens ταυτόχρονα.

Η positional encoding προστίθεται στις αναπαραστάσεις των tokens για να κωδικοποιηθεί η πληροφορία θέσης, καθώς ο μηχανισμός self-attention από μόνος του είναι αναλλοίωτος ως προς τη σειρά. Σύγχρονα LLMs χρησιμοποιούν τεχνικές όπως το Rotary Position Embedding (RoPE) ή το Absolute Position Embedding [12].

Η υπολογιστική πολυπλοκότητα του self-attention κλιμακώνεται τετραγωνικά με το μήκος της ακολουθίας εισόδου ($O(n^2)$)[11], γεγονός που συνιστά βασικό εμπόδιο για αποδοτικό inference σε μεγάλα πλαίσια κειμένου. Αυτό το χαρακτηριστικό οδηγεί σε αυξημένες υπολογιστικές απαιτήσεις, σημαντική κατανάλωση μνήμης και αυξημένη θερμική φόρτιση.

2.2.2 Εκπαίδευση Και Inference Σε LLMs

Η εκπαίδευση ενός LLM χωρίζεται συνήθως σε δύο φάσεις: την προ-εκπαίδευση (pre-training) και τη λεπτή ρύθμιση (fine-tuning). Κατά την προ-εκπαίδευση, το μοντέλο βελτιστοποιείται στην πρόβλεψη του επόμενου token σε τεράστια corpus κειμένου (self-supervised learning), αποκτώντας γενική γλωσσική γνώση. Αυτή η φάση απαιτεί τεράστιους υπολογιστικούς πόρους, συνήθως χιλιάδες GPU/TPU επί εβδομάδες ή μήνες, με κατανάλωση ενέργειας της τάξης των GWh.

Η λεπτή ρύθμιση (fine-tuning) προσαρμόζει το προ-εκπαιδευμένο μοντέλο σε εξειδικευμένες εργασίες ή domain-specific δεδομένα. Τεχνικές όπως το RLHF (Reinforcement Learning from Human Feedback) χρησιμοποιούνται για ευθυγράμμιση του μοντέλου με ανθρώπινες προτιμήσεις, ενώ το LoRA (Low-Rank Adaptation) μειώνει δραματικά τον αριθμό εκπαιδευόμενων παραμέτρων κατά τη φάση fine-tuning.

Το inference αναφέρεται στη φάση παραγωγής απόκρισης από ένα ήδη εκπαιδευμένο μοντέλο. Σε αυτοπαλινδρόμα LLMs (autoregressive LLMs), το inference ακολουθεί δύο φάσεις:

- Prefill (Φάση Προεπεξεργασίας): Επεξεργάζεται ολόκληρο το prompt εισόδου παράλληλα. Σε αυτή τη φάση υπολογίζονται και αποθηκεύονται οι KV-caches (Key-Value caches) για κάθε token εισόδου. Υπολογιστικά, είναι έντονη (compute-bound) λόγω μαζικών πράξεων πινάκων.
- Decode (Φάση Αποκωδικοποίησης): Παράγει ένα token τη φορά αυτοπαλινδρόμως, χρησιμοποιώντας τα αποθηκευμένα KV-caches. Χαρακτηρίζεται από έντονη εξάρτηση από το εύρος ζώνης μνήμης (memory-bound), γεγονός που επηρεάζει σημαντικά την κατανάλωση ενέργειας.

Στην πράξη, η φάση decode κυριαρχεί στη συνολική κατανάλωση ενέργειας ενός inference αιτήματος. Ο λόγος είναι ότι το decode εκτελείται μία φορά για κάθε παραγόμενο token, ενώ το prefill μία φορά ανά αίτημα, οπότε σε εργασίες με σύντομο prompt και μεγάλη απόκριση το decode καταλαμβάνει τη μεγάλη πλειονότητα του χρόνου και της ενέργειας (βλ. και ενότητα 2.4). Αυτό καθιστά την εύρεση αποδοτικών τεχνικών βελτιστοποίησης της φάσης decode κρίσιμης σημασίας.

2.3 Τεχνικές Βελτιστοποίησης για Ενεργειακά Αποδοτικό Inference

Καθώς η ανάπτυξη LLMs γίνεται ευρύτερη, η αναζήτηση τεχνικών που μειώνουν το υπολογιστικό και ενεργειακό κόστος του inference έχει λάβει έντονο ερευνητικό ενδιαφέρον. Βασικές κατηγορίες βελτιστοποίησης περιλαμβάνουν τον κβαντισμό μοντέλων (quantization), την αποκοπή (pruning), την απόσταξη γνώσης (knowledge distillation), τη δυναμική κλιμάκωση ακρίβειας (dynamic precision scaling) και τις αρχιτεκτονικές βελτιστοποιήσεις. Ο κβαντισμός, η αποκοπή και η δυναμική κλιμάκωση ακρίβειας, που εξετάζονται και πειραματικά στην παρούσα εργασία, αναλύονται διεξοδικά στις επόμενες υποενότητες [13].

2.3.1 Κβαντισμός Μοντέλων (Quantization)

Ο κβαντισμός (quantization) αποτελεί μια από τις πιο ευρέως χρησιμοποιούμενες τεχνικές βελτιστοποίησης inference. Βασίζεται στη μείωση της αριθμητικής ακρίβειας αναπαράστασης των παραμέτρων και ενεργοποιήσεων ενός μοντέλου, συνήθως από 32-bit ή 16-bit floating-point σε ακέραιους 8-bit (INT8) ή 4-bit (INT4), με στόχο τη μείωση της μνήμης, την επιτάχυνση των υπολογισμών και την εξοικονόμηση ενέργειας.

Υπάρχουν δύο κύριες κατηγορίες κβαντισμού:

- Κβαντισμός μετά την Εκπαίδευση (Post-Training Quantization, PTQ)
- Κβαντισμός με Επίγνωση Εκπαίδευσης (Quantization-Aware Training, QAT).

Η PTQ εφαρμόζεται σε ήδη εκπαιδευμένα μοντέλα χωρίς επανεκπαίδευση, μετατρέποντας τα βάρη FP32 σε χαμηλότερης ακρίβειας μορφές. Λόγω της απλότητας και της αποδοτικότητάς της, αποτελεί την κυρίαρχη μέθοδο στις παρούσες υλοποιήσεις LLM. Η QAT ενσωματώνει τα σφάλματα κβαντισμού στη διαδικασία εκπαίδευσης, επιτυγχάνοντας υψηλότερη ακρίβεια σε χαμηλότερο bitwidth, αλλά απαιτεί πρόσθετο υπολογιστικό κόστος.

Σημαντικές τεχνικές PTQ για LLMs περιλαμβάνουν το GPTQ, το AWQ (Activation-Aware Weight Quantization) και το SmoothQuant. Το GPTQ χρησιμοποιεί προσεγγιστικές βελτιστοποιήσεις δεύτερης τάξης για να ελαχιστοποιήσει τα σφάλματα ανακατασκευής ανά στρώμα, ενώ το AWQ λαμβάνει υπόψη την κατανομή των ενεργοποιήσεων για να εντοπίσει και να προστατεύσει τα κρίσιμα βάρη[14].

Ο κβαντισμός μπορεί να εφαρμοστεί σε διαφορετικά στοιχεία του μοντέλου: αποκλειστικά στα βάρη (weight-only quantization), στα βάρη και τις ενεργοποιήσεις (W+A quantization), ή και στον KV-cache. Ο κβαντισμός αποκλειστικά βαρών (weight-only) αποτελεί την πιο διαδεδομένη προσέγγιση σε edge και mobile συσκευές, όπου ο περιορισμός μνήμης είναι κρίσιμος.

Μια εκτεταμένη εμπειρική μελέτη αξιολόγησης 28 κβαντισμένων LLMs του Ollama framework σε Raspberry Pi 4 (4GB RAM) κατέδειξε ότι η PTQ με βάρη 4-bit (q4_K_M) επιτυγχάνει βέλτιστη ισορροπία μεταξύ κατανάλωσης ενέργειας, latency και ακρίβειας σε resource-constrained περιβάλλοντα[15]. Τα αποτελέσματα ανέδειξαν ότι μοντέλα με περισσότερες παραμέτρους μετά

από κβαντισμό μπορεί να υπερέχουν μικρότερων μη-κβαντισμένων μοντέλων σε ισορροπία ενέργειας-ακρίβειας.

Μια εξελιγμένη μορφή κβαντισμού είναι ο μικτός ακρίβειας κβαντισμός (mixed-precision quantization), όπου διαφορετικά στρώματα ή στοιχεία του μοντέλου χρησιμοποιούν διαφορετικό bitwidth ανάλογα με την ευαισθησία τους στη μείωση ακρίβειας. Αυτή η προσέγγιση επιτυγχάνει καλύτερη ισορροπία μεταξύ ακρίβειας και αποδοτικότητας σε σχέση με τον ομοιόμορφο κβαντισμό.

Ιδιαίτερης σημασίας είναι η πρόσφατη υιοθέτηση μορφών FP8 (8-bit floating-point) από σύγχρονους επιταχυντές, όπως οι GPU NVIDIA H100. Η μορφή FP8 προσφέρει διπλάσια απόδοση (throughput) σε σύγκριση με FP16, με μειωμένη κατανάλωση μνήμης έως 50%, διατηρώντας παράλληλα ικανοποιητική ακρίβεια μέσω τεχνικών σταθεροποίησης και δυναμικής κλιμάκωσης[16].

2.3.2 Αποκοπή Νευρωνικών Δικτύων (Pruning)

Η αποκοπή (pruning) νευρωνικών δικτύων αποτελεί μια κλασική τεχνική συμπίεσης μοντέλων που αποσκοπεί στην εξάλειψη περιττών ή ασήμαντων παραμέτρων, μειώνοντας έτσι το μέγεθος και το υπολογιστικό κόστος χωρίς σημαντική απώλεια απόδοσης. Ο αριθμός δημοσιεύσεων σχετικά με pruning έχει ξεπεράσει τις 3.000 για την περίοδο 2020–2024[17], αναδεικνύοντας το έντονο ερευνητικό ενδιαφέρον.

Οι τεχνικές pruning κατηγοριοποιούνται σε τρία κύρια είδη:

- Μη-δομημένη αποκοπή (Unstructured Pruning): Αφαιρεί μεμονωμένα βάρη οπουδήποτε στο δίκτυο, επιτυγχάνοντας υψηλούς λόγους αραιότητας (sparsity) με ελάχιστη απώλεια ακρίβειας. Ωστόσο, χωρίς εξειδικευμένη υποστήριξη υλικού, τα πρακτικά οφέλη επιτάχυνσης inference παραμένουν περιορισμένα.
- Δομημένη αποκοπή (Structured Pruning): Αφαιρεί ολόκληρα φίλτρα, κανάλια, κεφαλές attention ή στρώματα, παράγοντας πυκνά (dense) και άμεσα επιταχυνόμενα μοντέλα. Η μέθοδος LLM-Pruner, για παράδειγμα, εντοπίζει συστάδες νευρώνων με βάση τις εσωτερικές εξαρτήσεις και αφαιρεί όσες έχουν χαμηλή σημασία για το τελικό αποτέλεσμα.
- Ημι-δομημένη αποκοπή (Semi-Structured / N:M Sparsity): Αποτελεί συμβιβασμό μεταξύ των δύο προηγούμενων κατηγοριών. Επιβάλλει αραιότητα σε ομάδες διαδοχικών βαρών (π.χ. 2:4 sparsity: 2 μηδενικά ανά 4 βάρη), επιτυγχάνοντας πρακτική επιτάχυνση σε υλικό που υποστηρίζει αυτό το πρότυπο, όπως οι GPU NVIDIA Ampere και Hopper.

Εξέχουσες μέθοδοι unstructured pruning για LLMs περιλαμβάνουν το SparseGPT, το οποίο επιλύει το πρόβλημα αποκοπής ως προσεγγιστική ανακατασκευή αραιότητας χωρίς επανεκπαίδευση, και το Wanda, που αξιολογεί τη σημασία των βαρών λαμβάνοντας υπόψη τόσο το μέγεθός τους όσο και τις αντίστοιχες ενεργοποιήσεις εισόδου. Εμπειρικά, έχει βρεθεί ότι 50% αραιότητα σε μεγάλα μοντέλα (π.χ. LLaMA-65B) μπορεί να επιτευχθεί με ελάχιστη απώλεια ακρίβειας σε zero-shot εργασίες.

Στο πλαίσιο ενεργειακής αποδοτικότητας, το pruning μπορεί να μειώσει την κατανάλωση ενέργειας κατά 6–25% σε naïve λύσεις, αν και επιθετικές μορφές αποκοπής μπορεί να επιφέρουν σημαντική υποβάθμιση ακρίβειας.

Για τον λόγο αυτό έχουν προταθεί και προσεγγίσεις energy-aware pruning, οι οποίες επιλέγουν τα στοιχεία προς αφαίρεση με κριτήριο άμεσα την κατανάλωση ενέργειας του LLM [18].

2.3.3 Dynamic Precision Scaling

Η Δυναμική Κλιμάκωση Ακρίβειας (Dynamic Precision Scaling) αναφέρεται στη δυνατότητα προσαρμογής του bitwidth υπολογισμών κατά τη διάρκεια του inference, ανάλογα με τις απαιτήσεις ακρίβειας, τον διαθέσιμο υπολογιστικό πόρο ή τις θερμικές/ενεργειακές συνθήκες. Σε αντίθεση με τον στατικό κβαντισμό, αυτή η προσέγγιση επιτρέπει την προσαρμοστική εξισορρόπηση μεταξύ ακρίβειας και αποδοτικότητας σε πραγματικό χρόνο[19].

Σύγχρονα LLM inference frameworks (όπως το vLLM και το TensorRT-LLM) υποστηρίζουν μικτή ακρίβεια (mixed precision), όπου διαφορετικά επίπεδα ή φάσεις του inference εκτελούνται με διαφορετική ακρίβεια. Για παράδειγμα, η φάση prefill που είναι compute-bound μπορεί να εκτελεστεί με FP8, ενώ ευαίσθητες λειτουργίες όπως το Softmax στο attention διατηρούνται σε FP32 για λόγους αριθμητικής σταθερότητας.

Η μέθοδος PMPD (Progressive Mixed-Precision Decoding) αποτελεί ένα χαρακτηριστικό παράδειγμα δυναμικής κλιμάκωσης: χρησιμοποιεί υψηλότερη ακρίβεια στη φάση prefill και χαμηλότερη στη φάση decode, ενώ μειώνει προοδευτικά την ακρίβεια όσο προχωρά η παραγωγή της ακολουθίας, με χρονοπρογραμματιστές (schedulers) που αποφασίζουν πότε θα γίνει κάθε μετάβαση. Με τον τρόπο αυτό μειώνεται το μέσο bitwidth και αυξάνεται σημαντικά η ταχύτητα σε σχέση με μοντέλα fp16, χωρίς αισθητή απώλεια ποιότητας.

Η ανάπτυξη του FP8 ως τυποποιημένης μορφής σε σύγχρονους GPU επιταχυντές αποτελεί σημαντικό βήμα δυναμικής κλιμάκωσης ακρίβειας. Η μορφή E4M3 χρησιμοποιείται για τα εμπρόσθια βάρη και τις ενεργοποιήσεις (υψηλότερη ακρίβεια, περιορισμένο dynamic range), ενώ η E5M2 αποδίδεται στα οπίσθια gradients (ευρύτερο dynamic range, χαμηλότερη ακρίβεια). Αυτή η υβριδική στρατηγική επιτρέπει εκπαίδευση και inference με FP8 με αύξηση απόδοσης έως 2× και μειωμένη κατανάλωση μνήμης.

Η δυναμική κλιμάκωση ακρίβειας δεν περιορίζεται στο λογισμικό. Σε επίπεδο υλικού έχει εφαρμοστεί και σε μονάδες πολλαπλασιασμού-συσσώρευσης (MAC) επιταχυντών νευρωνικών δικτύων, όπου η ακρίβεια των πράξεων προσαρμόζεται δυναμικά ώστε να μειωθεί η κατανάλωση ενέργειας [20]. Συγγενής ιδέα είναι οι τεχνικές DVFS (Dynamic Voltage and Frequency Scaling), οι οποίες προσαρμόζουν την τάση και τη συχνότητα λειτουργίας του υλικού, για παράδειγμα των GPU, ανάλογα με το φορτίο και τη φάση του inference [21]. Η παρατήρηση ότι η φάση prefill είναι compute-bound και επομένως ευαίσθητη στη συχνότητα, ενώ η φάση decode είναι memory-bound και λιγότερο ευαίσθητη, επιτρέπει phase-aware DVFS πολιτικές με σημαντικά ενεργειακά οφέλη.

2.4 Thermally-Aware και Energy-Aware Inference

Πέραν των τεχνικών αλγοριθμικής βελτιστοποίησης, η θερμική διαχείριση και η ενεργειακή επίγνωση αποτελούν κρίσιμες διαστάσεις για αποδοτική ανάπτυξη LLMs, τόσο σε data center όσο και σε edge συσκευές.

Τα GPU συστήματα που εκτελούν LLM inference παράγουν σημαντική θερμότητα, η οποία πρέπει να παραμένει εντός ασφαλών ορίων. Για τις GPUs NVIDIA A100 και H100, το όριο θερμοκρασίας τυπικά κυμαίνεται στους 85°C. Υπέρβαση αυτού του ορίου ενεργοποιεί το λεγόμενο thermal throttling: η GPU μειώνει αυτόματα τη συχνότητα λειτουργίας για να μειώσει τη θερμοπαραγωγή, με άμεσες επιπτώσεις στην απόδοση και την ποιότητα υπηρεσίας.

Ένα σύστημα Thermally-Aware Workload Scheduling (TAWS) διαχειρίζεται τα inference workloads λαμβάνοντας υπόψη όχι μόνο τις υπολογιστικές απαιτήσεις, αλλά και τη θερμική κατάσταση κάθε GPU και το ψυκτικό περιβάλλον. Το TAWS διατυπώνεται ως διαδικασία απόφασης Markov (Markov Decision Process, MDP) που εξισορροπεί κόστος ψύξης, ενεργειακή κατανάλωση inference και διακύμανση SLO (Service Level Objectives).

Το σύστημα TAPAS (Thermal- and Power-Aware Scheduling for LLM Inference), που αναπτύχθηκε για cloud platforms της Microsoft Azure, επιδεικνύει πώς η παρακολούθηση θερμοκρασίας εισόδου σε χιλιάδες GPUs (A100 και H100) επί τρίμηνη περίοδο μπορεί να τροφοδοτήσει έξυπνη τοποθέτηση workloads [22]. Τα αποτελέσματά του αναδεικνύουν σημαντικές ανομοιομορφίες θερμοκρασίας μεταξύ διαδρόμων, racks και θέσεων εντός racks, γεγονός που επηρεάζει άμεσα την αποδοτικότητα του ψυκτικού συστήματος.

Σε συστήματα DVFS για energy-efficient LLM inference, έχει δειχθεί ότι η δυναμική ρύθμιση της συχνότητας των GPU, του αριθμού των instances και του παραλληλισμού του μοντέλου, με τήρηση των στόχων latency (SLO), μπορεί να μειώσει την κατανάλωση ενέργειας ενός cluster LLM inference κατά 53% [21]. Συγκεκριμένα, η παρατήρηση ότι η φάση decode είναι memory-bound επιτρέπει μείωση συχνότητας χωρίς σημαντική επίπτωση στο latency, ενώ η compute-bound prefill απαιτεί υψηλότερες συχνότητες.

Σε mobile και edge συστήματα, η δυναμική διαχείριση συχνότητας επεκτείνεται σε CPU, GPU και μνήμη ταυτόχρονα. Η ανεξάρτητη λειτουργία των τριών DVFS controllers (CPU, GPU, memory) χωρίς συντονισμό μπορεί να οδηγεί σε υποβέλτιστη ενεργειακή αποδοτικότητα. Έρευνες έχουν αναδείξει ότι ο συντονισμένος έλεγχος DVFS σε επίπεδο συστήματος μπορεί να μειώσει σημαντικά την κατανάλωση ενέργειας κατά το LLM inference σε κινητές συσκευές, χωρίς αισθητή επίπτωση στο latency. Σε κινητές συσκευές έχει επίσης μετρηθεί ότι η ενέργεια της φάσης decode είναι 16 έως 26 φορές μεγαλύτερη από αυτή του prefill, και ότι η κατάλληλη επιλογή πυρήνων CPU κατά το decode μειώνει την ενέργεια κατά 23% χωρίς επιβράδυνση [23].

Η energy-aware inference εκτείνεται και στη διαχείριση batch size, στον αριθμό ενεργών servers και στον τρόπο δρομολόγησης αιτημάτων. Η αύξηση batch size βελτιώνει την ενεργειακή αποδοτικότητα ανά token έως ένα σημείο κορεσμού: αύξηση από 32 σε 256 tokens ανά batch μπορεί να μειώσει την ενέργεια ανά token κατά ~25%, ενώ πέραν αυτού τα οφέλη εξαντλούνται. Παράλληλα, το σύστημα GreenLLM των Liu et al. (διαφορετικό από το ομώνυμο σύστημα energy-aware pruning της ενότητας 2.3.2) ελέγχει χωριστά τη συχνότητα της GPU στις φάσεις prefill και decode, μειώνοντας τη συνολική ενέργεια έως 34% σε σχέση με τον προεπιλεγμένο μηχανισμό DVFS, χωρίς απώλεια throughput [24].

Η θεμελιώδης κατανόηση των θερμικών και ενεργειακών δυναμικών κατά τη διάρκεια LLM inference, όπως παρουσιάστηκε σε αυτή την ενότητα, διαμορφώνει το υπόβαθρο για την ανάπτυξη συστημάτων Thermally-Aware και Energy-Aware Inference, κεντρικό θέμα της παρούσας εργασίας.

3^ο Κεφάλαιο: Περιγραφή Συστήματος Υλικού (Hardware)

3.1 Υλικό του Συστήματος

Στο πλαίσιο της παρούσας πτυχιακής εργασίας, για την υλοποίηση και αξιολόγηση ενεργειακά αποδοτικών μεθόδων εκτέλεσης Μεγάλων Γλωσσικών Μοντέλων (Large Language Models, LLMs) σε συστήματα περιορισμένων πόρων, χρησιμοποιήθηκε ως πλατφόρμα υλικού το Raspberry Pi 400. Η συγκεκριμένη συσκευή επιλέχθηκε λόγω του χαμηλού κόστους, της ευρείας διαθεσιμότητας, της περιορισμένης κατανάλωσης ενέργειας και της αντιπροσωπευτικότητάς της ως σύγχρονη embedded/edge πλατφόρμα.

Το Raspberry Pi 400 αποτελεί έναν πλήρη υπολογιστή ενσωματωμένο σε πληκτρολόγιο και βασίζεται σε αρχιτεκτονική ARM, γεγονός που το καθιστά ιδιαίτερα κατάλληλο για τη μελέτη εκτέλεσης υπολογιστικά απαιτητικών εφαρμογών, όπως τα LLMs, σε περιβάλλοντα με αυστηρούς ενεργειακούς και υπολογιστικούς περιορισμούς.

3.1.1 Χαρακτηριστικά Υλικού

Πίνακας 1: Χαρακτηριστικά Υλικού

Υλικό	Μοντέλο	Τεχνικά Χαρακτηριστικά
Επεξεργαστής (CPU)	Quad-core ARM Cortex-A72	- 4 Πυρήνες - Συχνότητα Λειτουργίας έως 1.8GHz
Μνήμη RAM	4 GB LPDDR4	- 4GB Μνήμη - 1600 MHz Συχνότητα - Χαμηλής Ισχύος
Αποθηκευτικός Χώρος	SanDisk Extreme Pro microSDXC 64GB	- 64GB Αποθηκευτικό Χώρο - 90MB/s Ταχύτητα Εγγραφής - 200MB/s Ταχύτητα Ανάγνωσης
Τροφοδοσία	Μέσω θύρας USB-C	- Είσοδος 5V

3.2 Υπολογιστικοί και Ενεργειακοί Περιορισμοί του Συστήματος

Η εκτέλεση LLMs σε συστήματα περιορισμένων πόρων όπως το Raspberry Pi 400, συνοδεύεται από σημαντικούς υπολογιστικούς και ενεργειακούς περιορισμούς. Οι περιορισμοί αυτοί αποτελούν βασικό παράγοντα που επηρεάζει τόσο την απόδοση όσο και τη βιωσιμότητα της εκτέλεσης τέτοιων μοντέλων σε περιβάλλοντα edge και embedded computing. Η κατανόηση και η αξιοποίηση αυτών των περιορισμών είναι κρίσιμη, καθώς το κύριο αντικείμενο μελέτης είναι η βελτιστοποίηση της ενεργειακής αποδοτικότητας και όχι η μέγιστη υπολογιστική απόδοση.

3.2.1 Υπολογιστικοί Περιορισμοί

- Περιορισμένη Υπολογιστική Ισχύς CPU

Το Raspberry Pi 400 διαθέτει τετραπύρρηνο επεξεργαστή ARM Cortex-A72, ο οποίος, αν και επαρκής για γενικής χρήσης εφαρμογές, δεν είναι σχεδιασμένος για εντατικές αριθμητικές πράξεις μεγάλης κλίμακας, όπως αυτές που απαιτούνται από τα LLMs. Η απουσία εξειδικευμένου επιταχυντή (π.χ. GPU ή NPU) καθιστά την εκτέλεση των νευρωνικών υπολογισμών αποκλειστικά ευθύνη της CPU.

Ως αποτέλεσμα:

- Η ταχύτητα inference είναι περιορισμένη,
- Η πλήρης αξιοποίηση όλων των πυρήνων οδηγεί σε αυξημένη κατανάλωση ενέργειας και παρατηρείται αυξημένος χρόνος απόκρισης σε σύγκριση με ισχυρότερα συστήματα.

- Περιορισμένη Διαθέσιμη Μνήμη RAM

Η διαθέσιμη μνήμη RAM του συστήματος ανέρχεται στα 4 GB, εκ των οποίων ένα σημαντικό μέρος καταναλώνεται από το λειτουργικό σύστημα και τις βασικές διεργασίες. Τα σύγχρονα LLMs, ακόμη και σε σχετικά μικρά μεγέθη παραμέτρων, απαιτούν σημαντικό αποτύπωμα μνήμης για:

- Τα βάρη του μοντέλου
- Τα intermediate activations
- Τα buffers κατά το inference

Ως Αποτέλεσμα:

- Υπάρχει αδυναμία φόρτωσης μοντέλων πλήρους ακρίβειας μεγάλου μεγέθους
- Υπάρχει ανάγκη χρήσης τεχνικών μείωσης μνήμης

- Περιορισμοί Αρχιτεκτονικής ARM

Η αρχιτεκτονική ARM, παρότι ενεργειακά αποδοτική, παρουσιάζει περιορισμούς σε σχέση με την υποστήριξη ορισμένων βιβλιοθηκών και βελτιστοποιήσεων που είναι περισσότερο ανεπτυγμένες για x86 συστήματα. Αυτό επηρεάζει:

- Την απόδοση των αριθμητικών πράξεων χαμηλής ακρίβειας
- Τη διαθεσιμότητα πλήρως βελτιστοποιημένων kernels για LLM inference

3.2.2 Ενεργειακοί Περιορισμοί

- Περιορισμένος Ενεργειακός Προϋπολογισμός

Το Raspberry Pi 400 λειτουργεί με χαμηλό ενεργειακό προϋπολογισμό, καθώς τροφοδοτείται μέσω θύρας USB-C στα 5V. Παρότι η συνολική κατανάλωση ενέργειας είναι χαμηλή σε σύγκριση με desktop συστήματα, η εκτέλεση LLM inference μπορεί να οδηγήσει σε:

- Σημαντικές στιγμιαίες αυξήσεις κατανάλωσης ισχύος
- Συνεχή υψηλό φορτίο CPU

- Μειωμένη ενεργειακή αποδοτικότητα
- Σχέση Απόδοσης και Κατανάλωσης Ενέργειας

Σε συστήματα όπως το Raspberry Pi 400, η αύξηση της υπολογιστικής ακρίβειας (π.χ. FP16 αντί INT8) οδηγεί σε:

- μεγαλύτερη χρήση μνήμης
- αυξημένη κατανάλωση ενέργειας ανά inference, λόγω μεγαλύτερου χρόνου εκτέλεσης

Αντίστοιχα, η μείωση της ακρίβειας μπορεί να μειώσει σημαντικά την κατανάλωση ενέργειας, εις βάρος όμως της ποιότητας ή της σταθερότητας των αποτελεσμάτων. Αυτή η σχέση δημιουργεί την ανάγκη για δυναμικές προσεγγίσεις προσαρμογής ακρίβειας.

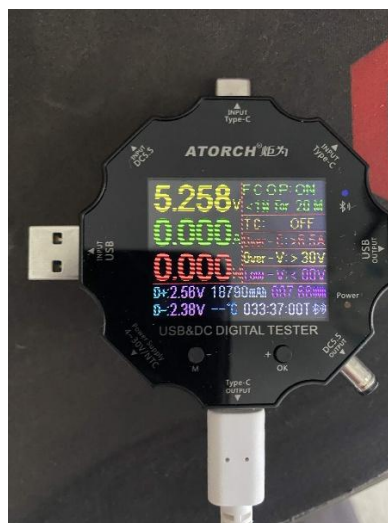
3.3 Σύστημα Μέτρησης Κατανάλωσης Ενέργειας

Για την αξιόπιστη αξιολόγηση της ενεργειακής αποδοτικότητας των διαφορετικών μεθόδων εκτέλεσης LLM, είναι απαραίτητη η ακριβής μέτρηση της κατανάλωσης ενέργειας κατά τη διάρκεια του inference. Στην παρούσα εργασία χρησιμοποιείται εξωτερικός αισθητήρας μέτρησης ισχύος, ο Atorch UD18, ο οποίος επιτρέπει τη μέτρηση της στιγμιαίας τάσης (Voltage) και ρεύματος (Current) που καταναλώνει το Raspberry Pi 400 σε πραγματικό χρόνο.

Η χρήση εξωτερικού αισθητήρα επιλέχθηκε ώστε να διασφαλιστεί η ανεξαρτησία και η ακρίβεια των μετρήσεων, αποφεύγοντας την επίδραση λογισμικών εργαλείων εκτίμησης κατανάλωσης που συχνά βασίζονται σε προσεγγίσεις ή μοντέλα. Με τον αισθητήρα να λειτουργεί ανεξάρτητα από το λογισμικό inference, οι μετρήσεις αντικατοπτρίζουν την πραγματική κατανάλωση της συσκευής συνολικά.

3.3.1 Περιγραφή του Atorch UD18

Ο Atorch UD18 είναι ένας εξελιγμένος αισθητήρας μέτρησης ισχύος υψηλής ακρίβειας με ενσωματωμένη επικοινωνία Bluetooth. Συνδέεται στο USB-C τροφοδοτικό καλώδιο μεταξύ του τροφοδοτικού και της θύρας τροφοδοσίας USB-C του Raspberry Pi 400, μετρώντας την συνολική κατανάλωση ισχύος της συσκευής. Η επικοινωνία γίνεται ασύρματα μέσω Bluetooth RFCOMM, επιτρέποντας στο λογισμικό παρακολούθησης να λαμβάνει μετρήσεις απευθείας από τη συσκευή χωρίς καμία φυσική σύνδεση στο Raspberry Pi.



Εικόνα 1: Atorch UD18

Η συσκευή παρέχει τις ακόλουθες βασικές μετρήσεις:

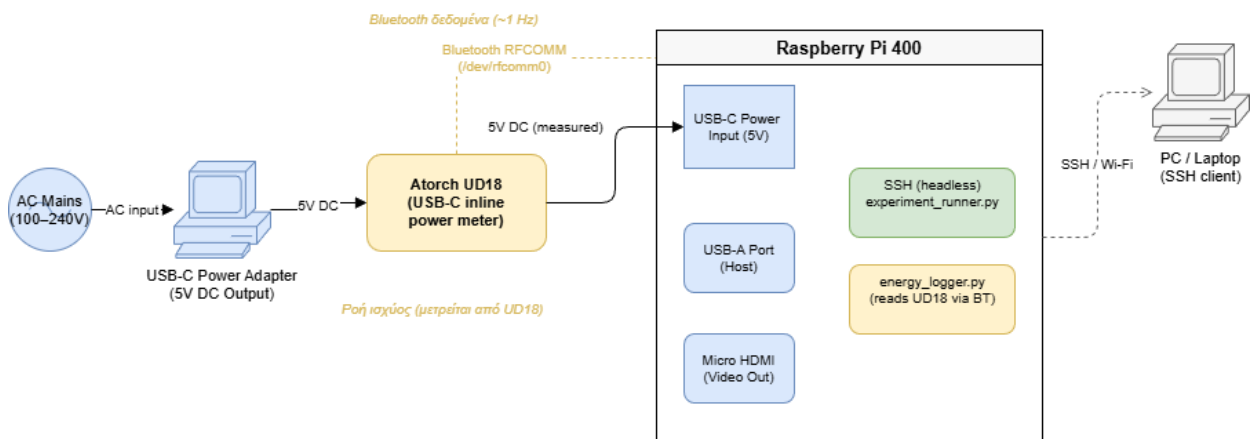
- Τάση Τροφοδοσίας (Volt)
- Ένταση Ρεύματος (Ampere)
- Στιγμιαία Ηλεκτρική Ισχύς (Watt)

Τα δεδομένα μετρήσεων μεταδίδονται σε ειδικά δυαδικά πακέτα μέσω του Bluetooth. Το λογισμικό παρακολούθησης (energy_logger.py) διαβάζει κάθε πακέτο ως raw bytes και εξάγει τις τιμές ισχύος μέσω συγκεκριμένων byte offsets: η τάση λαμβάνεται από τα bytes 4–6 (uint24/100), το ρεύμα από τα bytes 7–9 (uint24/100) και η ισχύς από τα bytes 22–25 (uint32/1000).

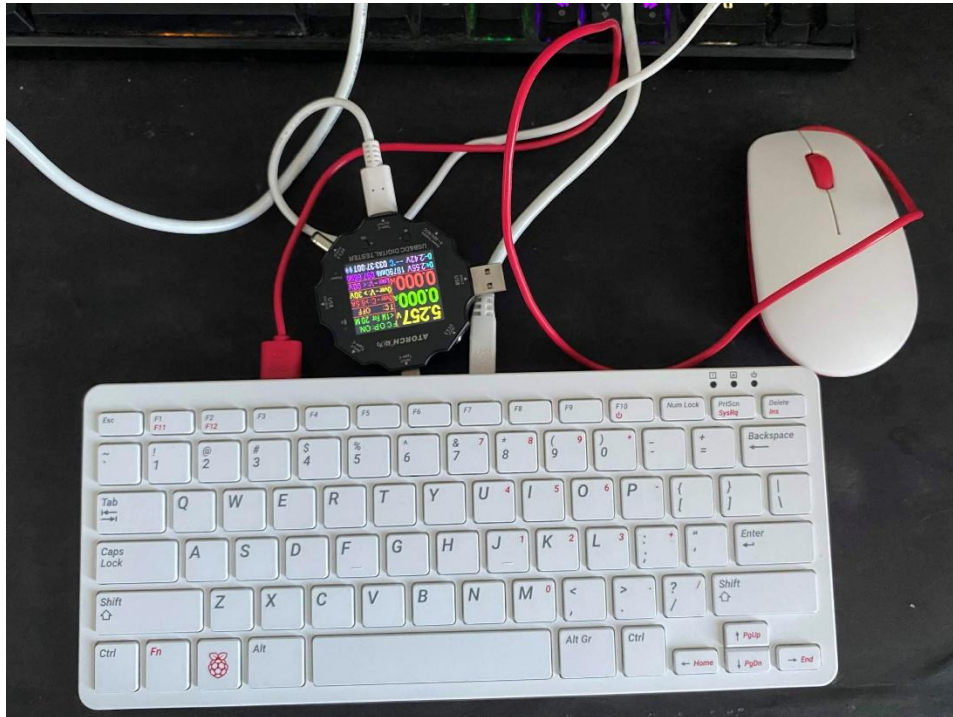
Η δειγματοληψία γίνεται με ρυθμό περίπου 1 Hz και οι τιμές ενέργειας υπολογίζονται με τραπεζοειδή ολοκλήρωση της ισχύος ως προς τον πραγματικό χρόνο μεταξύ των δειγμάτων για να ληφθεί η συνολική κατανάλωση ενέργειας σε Joule.

3.4 Συνδεσμολογία και Διάγραμμα Συστήματος

Η συνδεσμολογία του συστήματος υλοποιείται ώστε να επιτρέπει τη μέτρηση της κατανάλωσης ενέργειας χωρίς να επηρεάζει την κανονική τροφοδοσία της συσκευής. Ο αισθητήρας Atorch UD18 τοποθετείται μεταξύ του τροφοδοτικού USB-C και της θύρας τροφοδοσίας του Raspberry Pi 400, επιτρέποντας τη μέτρηση ολόκληρης της κατανάλωσης ισχύος της συσκευής, συμπεριλαμβανομένων της CPU, της μνήμης και των λοιπών συστατικών. Η επικοινωνία γίνεται μέσω Bluetooth RFCOMM (sudo rfcomm connect 0 AA:B4:0A:65:EA:6D 1), επιτρέποντας τη συνεχή λήψη μετρήσεων σε πραγματικό χρόνο κατά τη διάρκεια κάθε inference run. Η κύρια τροφοδοσία του Raspberry Pi παρέχεται κανονικά μέσω τροφοδοτικού USB-C, το οποίο τροφοδοτεί εν σειρά τον αισθητήρα UD18 και στη συνέχεια το Raspberry Pi. Έτσι ο αισθητήρας μετρά την πλήρη κατανάλωση του συστήματος, ενώ η λήψη των δεδομένων γίνεται ασύρματα μέσω Bluetooth στο ίδιο το μηχάνημα χωρίς επιβάρυνση στο USB δίαυλο. Η συγκεκριμένη διάταξη επιτρέπει τη σύγκριση διαφορετικών μεθόδων inference υπό σταθερές συνθήκες λειτουργίας, διασφαλίζοντας την αξιοπιστία και την αναπαραγωγιμότητα των αποτελεσμάτων.



Εικόνα 2: Συνδεσμολογία & Διάγραμμα Συστήματος



Εικόνα 3: Σύστημα

4ο Κεφάλαιο: Περιγραφή Λογισμικού (Software)

4.1 Επιλογή Γλωσσών Προγραμματισμού

Για την υλοποίηση και αξιολόγηση LLMs σε embedded περιβάλλον, επιλέχθηκαν οι γλώσσες Python και YAML (YAML Ain't Markup Language). Η επιλογή αυτή βασίστηκε σε κριτήρια που σχετίζονται άμεσα με τη φύση του προβλήματος, τους υπολογιστικούς και ενεργειακούς περιορισμούς του συστήματος, καθώς και τις απαιτήσεις πειραματικής αξιολόγησης και επαναληψιμότητας.

4.1.1 Python

Η Python είναι η πιο διαδεδομένη γλώσσα προγραμματισμού στον χώρο της Μηχανικής Μάθησης και των LLM. Βιβλιοθήκες όπως οι PyTorch, Transformers (Hugging Face) και NumPy προσφέρουν πλήρη υποστήριξη για φόρτωση, εκτέλεση, κβαντισμό (quantization) και pruning μοντέλων. Η ύπαρξη αυτών των εργαλείων καθιστά εφικτή την υλοποίηση προηγμένων τεχνικών ενεργειακής βελτιστοποίησης χωρίς την ανάγκη χαμηλού επιπέδου προγραμματισμού.

Παρότι η Python δεν είναι γλώσσα χαμηλού επιπέδου, είναι ιδιαίτερα διαδεδομένη σε embedded Linux συστήματα όπως το Raspberry Pi. Η ευκολία εγκατάστασης, η συμβατότητα με ARM αρχιτεκτονική και η διαθεσιμότητα έτοιμων βιβλιοθηκών επιτρέπουν γρήγορη ανάπτυξη και αξιόπιστη λειτουργία, ακόμα και σε περιβάλλοντα με περιορισμένους πόρους. Επιπρόσθετα προσφέρει βιβλιοθήκες όπως το psutil για την παρακολούθηση CPU και RAM, καθώς και βιβλιοθήκες για επικοινωνία με εξωτερικές συσκευές μέτρησης ενέργειας (Atorch UD18). Αυτό επέτρεψε την υλοποίηση ενός ολοκληρωμένου συστήματος συλλογής μετρήσεων χωρίς την ανάγκη εξωτερικών εργαλείων ή πολύπλοκων drivers.

Τέλος, δεδομένου ότι η παρούσα πτυχιακή έχει ερευνητικό χαρακτήρα, η καθαρή δομή και η αναγνωσιμότητα του κώδικα είναι ιδιαίτερα σημαντικές. Η Python επιτρέπει την υλοποίηση σύνθετης λογικής (π.χ. Dynamic Precision Scaling) με σαφή και κατανοητό τρόπο, διευκολύνοντας τόσο τη συντήρηση όσο και την αναπαραγωγή των πειραμάτων.

4.1.2 YAML

Η χρήση αρχείου configuration YAML επιτρέπει τον καθαρό διαχωρισμό της λογικής του κώδικα από τις παραμέτρους των πειραμάτων. Πληροφορίες όπως τα μοντέλα που θα χρησιμοποιηθούν, οι μέθοδοι inference, τα prompts, ο αριθμός επαναλήψεων και τα paths αποθήκευσης ορίζονται εξωτερικά, χωρίς να απαιτείται τροποποίηση του κώδικα.

Επιπλέον, η δυνατότητα αποθήκευσης όλων των παραμέτρων ενός πειράματος σε ένα αρχείο YAML συμβάλλει καθοριστικά στην επιστημονική εγκυρότητα της πτυχιακής. Κάθε πείραμα μπορεί να επαναληφθεί ακριβώς με τις ίδιες ρυθμίσεις, κάτι ιδιαίτερα σημαντικό για τη σύγκριση ενεργειακών και υπολογιστικών μετρικών.

4.2 Βιβλιοθήκες και Πλαίσιο Ανάπτυξης

Για την υλοποίηση του κώδικα επιλέχθηκε ένα σύνολο εργαλείων και βιβλιοθηκών που καλύπτουν τόσο τις ανάγκες ανάπτυξης λογισμικού όσο και τις απαιτήσεις εκτέλεσης, παρακολούθησης και αξιολόγησης των LLM. Ιδιαίτερη έμφαση δόθηκε στη σταθερότητα, τη συμβατότητα με ARM αρχιτεκτονική και τη δυνατότητα λεπτομερούς ενεργειακής ανάλυσης.

4.2.1 Microsoft Visual Studio Code

Ως βασικό πλαίσιο ανάπτυξης επιλέχθηκε το Microsoft Visual Studio Code (VS Code). Το VS Code αποτελεί ένα ελαφρύ αλλά ισχυρό περιβάλλον ανάπτυξης, ιδιαίτερα διαδεδομένο στον χώρο της Python και της Μηχανικής Μάθησης. Παρέχει προηγμένες δυνατότητες όπως διαχείριση έργων, ενσωματωμένο τερματικό, έλεγχο εκδόσεων και υποστήριξη εικονικών περιβαλλόντων (virtual environments).

Η επιλογή του VS Code κρίθηκε κατάλληλη καθώς επιτρέπει την ανάπτυξη και τον έλεγχο του κώδικα τόσο σε υπολογιστή όσο και σε απομακρυσμένο σύστημα, όπως το Raspberry Pi, μέσω SSH. Με τον τρόπο αυτό διευκολύνεται η ανάπτυξη, η αποσφαλμάτωση (debugging) και η συντήρηση του κώδικα, χωρίς να απαιτείται άμεση εργασία πάνω στη συσκευή-στόχο.

4.2.2 Βασικές Βιβλιοθήκες Μηχανικής Μάθησης και Inference

- **llama-cpp-python** Η βιβλιοθήκη llama-cpp-python αποτέλεσε το κεντρικό framework για την εκτέλεση inference στο Raspberry Pi 400. Πρόκειται για Python bindings της βιβλιοθήκης llama.cpp, η οποία υλοποιεί βελτιστοποιημένο CPU inference για μοντέλα σε μορφή GGUF. Η εγκατάσταση έγινε με ενεργοποιημένη τη σημαία CMAKE_ARGS="-DGGML_NATIVE=on", επιτρέποντας τη χρήση των NEON διανυσματικών εντολών του ARM Cortex-A72 για μέγιστη απόδοση. Η βιβλιοθήκη υποστηρίζει εγγενώς τόσο FP16 όσο και Q8_0 (INT8) μοντέλα μέσω του format GGUF, χωρίς να απαιτείται GPU ή εξειδικευμένος επιταχυντής.
- **PyTorch (torch) και torch.nn.utils.prune** Το PyTorch χρησιμοποιήθηκε αποκλειστικά στο στάδιο προετοιμασίας των μοντέλων σε Windows PC, όχι στο Raspberry Pi. Συγκεκριμένα, η υποβιβλιοθήκη torch.nn.utils.prune χρησιμοποιήθηκε για την εφαρμογή unstructured L1 pruning στα γραμμικά επίπεδα των μοντέλων με ποσοστό 30%, μηδενίζοντας τις λιγότερο σημαντικές παραμέτρους χωρίς επανεκπαίδευση. Τα pruned μοντέλα στη συνέχεια μετατράπηκαν σε μορφή GGUF και μεταφέρθηκαν στο Pi για inference, εξαιρώντας έτσι την επιβάρυνση του pruning από τις μετρήσεις ενέργειας.
- **Transformers (Hugging Face)** Η βιβλιοθήκη transformers χρησιμοποιήθηκε επίσης αποκλειστικά στο PC για τη λήψη και φόρτωση των pretrained μοντέλων (TinyLlama-1.1B, Llama-3.2-1B, Qwen2-0.5B) από το Hugging Face Hub, καθώς και για τη μετατροπή τους σε μορφή GGUF μέσω του εργαλείου convert_hf_to_gguf.py της llama.cpp. Στο Raspberry Pi δεν χρησιμοποιήθηκε καθόλου, καθώς το GGUF format είναι πλήρως φορητό και ανεξάρτητο αρχιτεκτονικής.

4.2.3 Βιβλιοθήκες Παρακολούθησης και Μετρήσεων

Για τη συλλογή μετρικών απόδοσης και κατανάλωσης ενέργειας χρησιμοποιήθηκαν εξειδικευμένες βιβλιοθήκες συστήματος. Η παρακολούθηση CPU γίνεται μέσω της βιβλιοθήκης psutil (cpu_percent) σε πραγματικό χρόνο, ενώ η μέτρηση RAM γίνεται μέσω του /proc/meminfo, χρησιμοποιώντας τα MemTotal και MemFree για τον υπολογισμό του ποσοστού χρησιμοποιούμενης μνήμης. Η επικοινωνία με τον αισθητήρα Atorch UD18 γίνεται μέσω Bluetooth RFCOMM που εκθέτει μια σειριακή σύνδεση στο /dev/rfcomm0, την οποία το λογισμικό ανοίγει μέσω της βιβλιοθήκης pyserial (9600 baud).

Κάθε πακέτο μετρήσεων αναλύεται ως raw bytes με συγκεκριμένα byte offsets: τάση από bytes 4–6 (uint24/100), ρεύμα από bytes 7–9 (uint24/100) και ισχύς από bytes 22–25 (uint32/1000). Οι μετρήσεις αυτές αποτέλεσαν τη βάση για τον υπολογισμό των ενεργειακών μετρικών, όπως η συνολική κατανάλωση ενέργειας σε Joule και η ενέργεια ανά παραγόμενο token (J/token).

4.2.4 Υποστηρικτικές Βιβλιοθήκες

Συμπληρωματικά, χρησιμοποιήθηκαν βιβλιοθήκες γενικού σκοπού όπως οι time, threading, subprocess και yaml. Η βιβλιοθήκη time χρησιμοποιήθηκε για τη μέτρηση της διάρκειας κάθε inference run, ενώ η threading επέτρεψε τη συλλογή μετρήσεων ενέργειας και απόδοσης σε παράλληλο νήμα κατά τη διάρκεια του inference, χωρίς να επηρεάζεται η εκτέλεση του μοντέλου. Η βιβλιοθήκη subprocess αξιοποιήθηκε από τον orchestrator (experiment_runner.py) για την εκκίνηση κάθε inference run ως ανεξάρτητη διεργασία, διασφαλίζοντας την πλήρη αποδέσμευση μνήμης μεταξύ διαδοχικών μεθόδων.

Η yaml χρησιμοποιήθηκε για τη διαχείριση της πειραματικής διαμόρφωσης (μοντέλα, prompts, αριθμός runs, max tokens) μέσω του αρχείου config.yaml. Τέλος, η matplotlib χρησιμοποιήθηκε για την οπτικοποίηση των αποτελεσμάτων, παράγοντας γραφήματα σύγκρισης ενεργειακών και υπολογιστικών μετρικών μεταξύ μεθόδων και μοντέλων.

4.3 Περιγραφή Κώδικα

Στην παρούσα ενότητα περιγράφεται αναλυτικά το λογισμικό που αναπτύχθηκε στο πλαίσιο της παρούσας πτυχιακής εργασίας. Το σύστημα αποτελείται από διακριτά αρχεία κώδικα, οργανωμένα σε υποφακέλους, τα οποία συνεργάζονται για την εκτέλεση πειραμάτων inference LLMs σε embedded περιβάλλον, με ταυτόχρονη παρακολούθηση υπολογιστικών και ενεργειακών μετρικών.

Η αρχιτεκτονική του λογισμικού ακολουθεί αρθρωτή (modular) προσέγγιση, ώστε κάθε συνιστώσα να επιτελεί έναν σαφώς ορισμένο ρόλο, διευκολύνοντας την επεκτασιμότητα, τη συντήρηση και την πειραματική αξιολόγηση.

Κρίσιμη αρχιτεκτονική απόφαση αποτέλεσε ο διαχωρισμός των σταδίων προετοιμασίας μοντέλων και inference: ο κβαντισμός (quantization) και το pruning εκτελέστηκαν εκ των προτέρων σε Windows PC και τα μοντέλα αποθηκεύτηκαν σε μορφή GGUF, ενώ το Raspberry Pi 400 χρησιμοποιήθηκε αποκλειστικά για καθαρό inference, διασφαλίζοντας ότι οι μετρήσεις ενέργειας αντικατοπτρίζουν μόνο το κόστος εκτέλεσης.

4.3.1 run_method.py Επίπεδο: Υψηλού Επιπέδου

Το αρχείο run_method.py αποτελεί τον κεντρικό worker της πειραματικής διαδικασίας. Εκκινείται ως ανεξάρτητη διεργασία από τον experiment_runner.py για κάθε συνδυασμό μοντέλου/μεθόδου, εκτελεί όλα τα runs της μεθόδου και τερματίζει αποδεσμεύοντας πλήρως τη μνήμη.

- Λειτουργία
 - Φορτώνει το GGUF μοντέλο μέσω llama-cpp-python εκ νέου σε κάθε run, διασφαλίζοντας καθαρό context window
 - Εκτελεί inference με create_chat_completion() (TinyLlama) ή plain completion (Llama3.2, Qwen2)
 - Για DPS: αξιολογεί το s/token του προηγούμενου run και αποφασίζει αλλαγή precision
 - Αποθηκεύει averaged metrics CSV, responses CSV και γράφημα ανά μέθοδο

- Απαιτήσεις
 - GGUF αρχεία διαθέσιμα στον φάκελο models/gguf/
 - Εγκατεστημένη llama-cpp-python με GGML_NATIVE=on
 - Ενεργή σύνδεση Bluetooth με τον αισθητήρα UD18
- Προδιαγραφές
 - Ανεξάρτητη διεργασία ανά μέθοδο, με πλήρη αποδέσμευση μνήμης μεταξύ μεθόδων
 - temperature=0.1 για αναπαραγωγικότητα αποτελεσμάτων
 - DPS threshold: 0.80 s/token (slow), 0.65 s/token (fast)

4.3.2 Προετοιμασία Μοντέλων (Windows PC) Επίπεδο: Χαμηλού Επιπέδου

Η προετοιμασία των μοντέλων εκτελέστηκε εκτός Raspberry Pi, σε Windows PC με Python 3.11, και περιλαμβάνει δύο στάδια: pruning και μετατροπή σε GGUF. Για το pruning χρησιμοποιήθηκε η torch.nn.utils.prune με unstructured L1 pruning ποσοστού 30% στα γραμμικά επίπεδα κάθε μοντέλου, χωρίς επανεκπαίδευση. Τα βάρη μηδενίζονται μόνιμα με prune.remove(), διατηρώντας την αρχιτεκτονική αμετάβλητη.

Η μετατροπή σε GGUF έγινε μέσω του convert_hf_to_gguf.py της llama.cpp, ενώ ο κβαντισμός Q8_0 εφαρμόστηκε με το εργαλείο llama-quantize.exe. Τα εννέα τελικά GGUF αρχεία (3 μοντέλα × 3 παραλλαγές: fp16, q8, pruned) μεταφέρθηκαν στο Pi στον φάκελο ~/Software/models/gguf/.

- Προδιαγραφές
 - Unstructured L1 pruning, ποσοστό 30%, χωρίς επανεκπαίδευση
 - Μορφή εξόδου: GGUF (fp16, Q8_0, pruned fp16)
 - Εκτέλεση εκτός Pi, χωρίς επίδραση στις μετρήσεις ενέργειας

4.3.3 dps_inference (λογική εντός run_method.py) Επίπεδο: Υψηλού Επιπέδου

Η λογική Dynamic Precision Scaling υλοποιείται εντός του run_method.py. Πριν από κάθε run (εκτός του πρώτου), το σύστημα αξιολογεί το s/token του προηγούμενου run και αποφασίζει δυναμικά αν το επόμενο inference θα εκτελεστεί σε FP16 ή Q8_0 ακρίβεια, φορτώνοντας το αντίστοιχο GGUF αρχείο.

- Λειτουργία
 - Run 1: εκκίνηση πάντα σε FP16 (πλήρης ακρίβεια)
 - Κάθε επόμενο run: αξιολόγηση s/token προηγούμενου run
 - Αν s/token > 0.80: μετάβαση σε Q8_0 (throughput πολύ χαμηλό)
 - Αν s/token < 0.65: επιστροφή σε FP16 (throughput αρκετά υψηλό)
 - Καταγραφή κάθε αλλαγής στο <μοντέλο>_dps_switch_log.json

- Προδιαγραφές
 - Threshold-based λογική με hysteresis (0.80 / 0.65 s/token)
 - Απόφαση βασισμένη σε μετρημένο throughput, όχι εκτίμηση
 - Software-only υλοποίηση, δεν απαιτεί hardware υποστήριξη

4.3.4 performance_monitor.py Επίπεδο: Χαμηλού Επιπέδου

Το performance_monitor.py είναι υπεύθυνο για τη συλλογή μετρικών απόδοσης του συστήματος σε πραγματικό χρόνο. Η χρήση CPU υπολογίζεται μέσω της βιβλιοθήκης psutil (cpu_percent), ενώ η κατανάλωση RAM διαβάζεται απευθείας από το /proc/meminfo ως (MemTotal – MemFree) / MemTotal, ώστε να περιλαμβάνεται και η μνήμη των βαρών του μοντέλου που διατηρεί το Linux στο buff/cache.

- Παρεχόμενες Μετρικές
 - CPU usage (%)
 - RAM usage (%)
- Προδιαγραφές
 - Ανάγνωση RAM απευθείας από /proc/meminfo με χαμηλό overhead
 - Real-time μετρήσεις CPU μέσω psutil
 - Επαναχρησιμοποίηση από το metrics.py

4.3.5 energy_logger.py Επίπεδο: Χαμηλού Επιπέδου

Το energy_logger.py επικοινωνεί με τον αισθητήρα Atorch UD18 μέσω Bluetooth RFCOMM και καταγράφει στιγμιαίες τιμές τάσης και ρεύματος. Το αρχείο λειτουργεί ως abstraction layer μεταξύ hardware και λογισμικού, παρέχοντας αξιόπιστες ενεργειακές μετρήσεις χωρίς άμεση εμπλοκή στο inference.

- Λειτουργία
 - Σύνδεση στο /dev/rfcomm0 μέσω pyserial (9600 baud)
 - Ανάλυση raw bytes κάθε πακέτου με συγκεκριμένα offsets: τάση bytes 4–6 (uint24/100), ρεύμα bytes 7–9 (uint24/100), ισχύς bytes 22–25 (uint32/1000)
 - Δειγματοληψία ~1 Hz κατά τη διάρκεια κάθε inference run
- Προδιαγραφές
 - Ανάγνωση Volt και Ampere
 - Απομόνωση hardware logic από inference
 - Αξιόπιστη μέτρηση χωρίς παρεμβολή στην εκτέλεση

4.3.6 metrics.py Επίπεδο: Υψηλού Επιπέδου

Το metrics.py υλοποιεί τη συλλογή, αποθήκευση και επεξεργασία όλων των μετρικών αξιολόγησης μέσω της κλάσης MetricsCollector. Κατά τη διάρκεια του inference εκτελεί περιοδική δειγματοληψία ισχύος, CPU και RAM σε ξεχωριστό thread. Μετά το πέρας της εκτέλεσης υπολογίζει τις τελικές μετρικές.

- Παρεχόμενες Μετρικές
 - Συνολικός χρόνος inference (s)
 - Μέση ισχύς (W) και συνολική ενέργεια (J)
 - Joule ανά token (J/token)
 - Μέση χρήση CPU (%) και RAM (%)
 - Αριθμός παραγόμενων tokens
- Προδιαγραφές
 - Παράλληλη συλλογή μετρικών με threading χωρίς παρεμβολή στο inference
 - Χρήση ως context manager
 - Ανεξαρτησία από τη μέθοδο inference

4.3.7 graphs.py Επίπεδο: Υψηλού Επιπέδου

Το graphs.py αναλαμβάνει τη γραφική απεικόνιση των αποτελεσμάτων μέσω δύο συναρτήσεων. Η plot_metrics() παράγει γράφημα ράβδων για τα runs μιας μεθόδου, με χρωματική κωδικοποίηση για DPS (μπλε = fp16, πορτοκαλί = Q8_0). Η plot_comparison() παράγει το κεντρικό συγκριτικό γράφημα όλων των μοντέλων και μεθόδων, το οποίο καλείται αυτόματα από τον experiment_runner μετά την ολοκλήρωση όλων των runs.

- Απαιτήσεις
 - Διαθέσιμα CSV metrics στον φάκελο Results/
 - Εγκατεστημένη matplotlib
- Προδιαγραφές
 - Αποθήκευση γραφημάτων σε PNG
 - Σύγκριση μεταξύ μεθόδων και μοντέλων
 - Αναπαραγωγικότητα από τα ίδια CSV δεδομένα

4.3.8 experiment_runner.py & config.yaml Επίπεδο: Υψηλού Επιπέδου

Το experiment_runner.py αποτελεί τον κεντρικό orchestrator των πειραμάτων. Διαβάζει τις παραμέτρους από το config.yaml, εκκινεί κάθε συνδυασμό μοντέλου/μεθόδου ως ανεξάρτητη διεργασία subprocess, διαχειρίζεται την αποδέσμευση μνήμης μεταξύ μεθόδων με drop_caches και στο τέλος καλεί την plot_comparison() για τη δημιουργία του συγκριτικού γραφήματος.

Το config.yaml ορίζει τα μοντέλα, τις μεθόδους inference, τα prompts ανά μοντέλο, τον αριθμό επαναλήψεων (runs_per_method: 5), τον μέγιστο αριθμό tokens (max_new_tokens: 100) και τις διαδρομές των GGUF αρχείων και αποτελεσμάτων. Επιτρέπει την πλήρη αναπαραγωγή του πειράματος χωρίς τροποποίηση του πηγαίου κώδικα.

- Προδιαγραφές
 - ο Εκκίνηση κάθε μεθόδου ως subprocess για εγγυημένη αποδέσμευση μνήμης
 - ο Αυτόματο drop_caches πριν και μετά από κάθε μέθοδο
 - ο Αποθήκευση αποτελεσμάτων σε Results/ ως CSV και PNG
 - ο Κεντρικός έλεγχος ροής για διαχωρισμό πειραματικής λογικής από κώδικα

4.4 Διαγράμματα και Μοντελοποίηση (UML)

Στην παρούσα ενότητα παρουσιάζεται η μοντελοποίηση του λογισμικού συστήματος μέσω διαγραμμάτων UML (Unified Modeling Language), με σκοπό τη σαφή και δομημένη αποτύπωση της αρχιτεκτονικής, της λειτουργικότητας και των αλληλεπιδράσεων μεταξύ των επιμέρους συνιστωσών του συστήματος. Τα UML διαγράμματα λειτουργούν συμπληρωματικά προς την περιγραφή του κώδικα, επιτρέποντας την κατανόηση της συνολικής λειτουργίας του λογισμικού σε υψηλό επίπεδο αφάιρεσης, χωρίς να απαιτείται άμεση μελέτη της υλοποίησης.

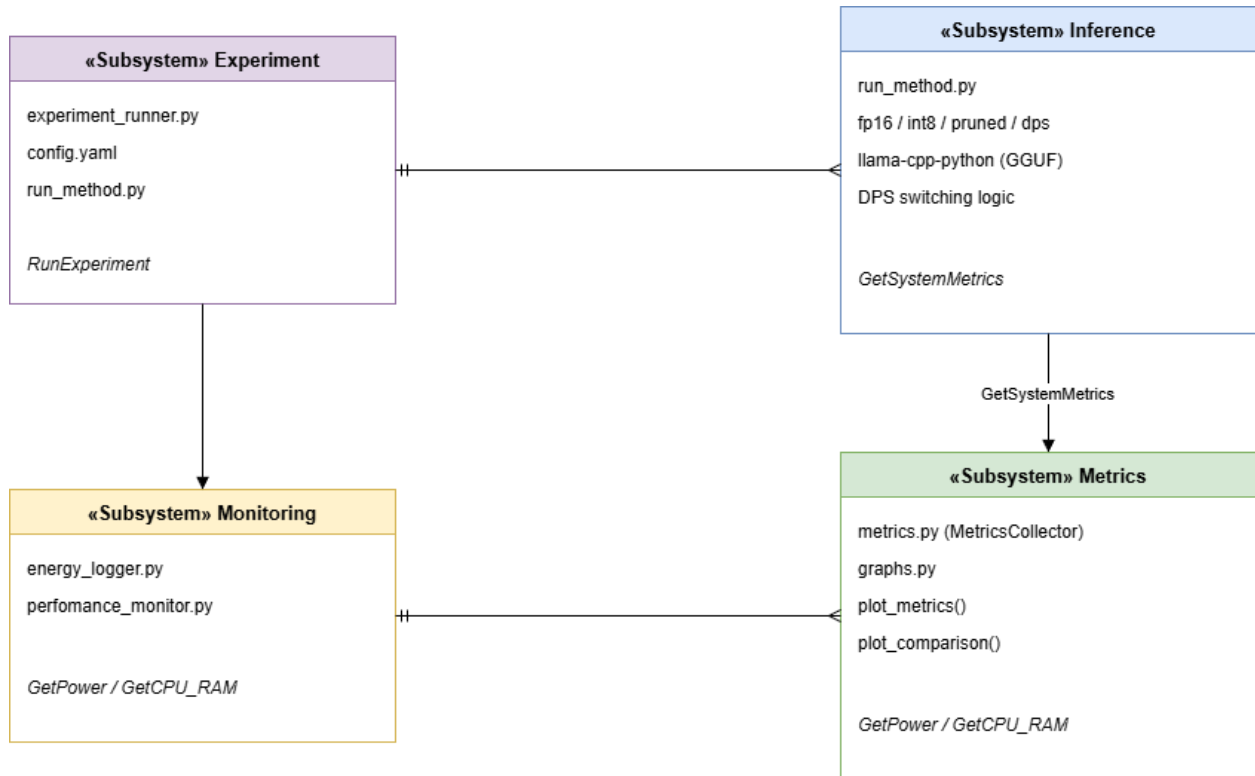
Μέσω αυτών, καθίσταται ευκολότερη η ανάλυση, η επεκτασιμότητα και η τεκμηρίωση του συστήματος που αναπτύχθηκε για αυτήν την πτυχιακή εργασία.

4.4.1 Διάγραμμα Περιπτώσεων Χρήσης



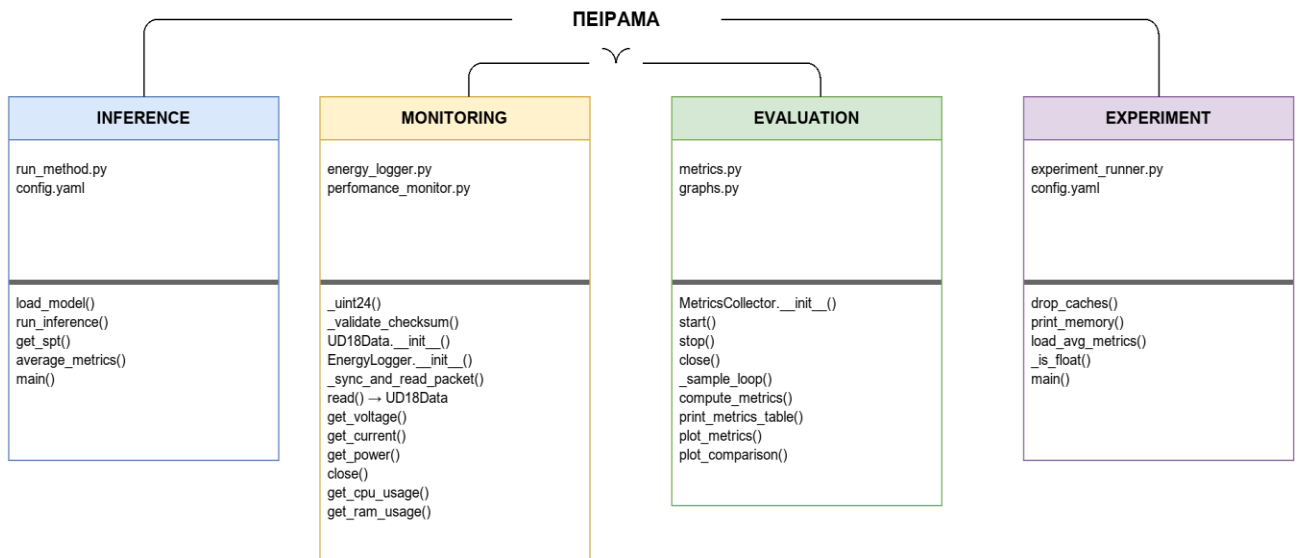
Εικόνα 4: Διάγραμμα Περιπτώσεων Χρήσης

4.4.2 Διάγραμμα Στοιχείων



Εικόνα 5: Διάγραμμα Στοιχείων

4.4.3 Διάγραμμα Κλάσης



Εικόνα 6: Διάγραμμα Κλάσης

4.5 Παραδείγματα Υλοποίησης Κώδικα & Περιγραφή Εφαρμογής

Στην παρούσα ενότητα παρουσιάζονται ενδεικτικά αποσπάσματα κώδικα από τα βασικά υποσυστήματα της εφαρμογής, με στόχο την καλύτερη κατανόηση της υλοποίησης και της λειτουργικής δομής του λογισμικού. Σκοπός δεν είναι η αναπαραγωγή ολόκληρης της βάσης κώδικα, αλλά η ανάδειξη των κρίσιμων μηχανισμών: η λογική DPS switching, η ανάγνωση δεδομένων ισχύος από τον αισθητήρα UD18, η παράλληλη συλλογή μετρικών μέσω MetricsCollector, και η ορχήστρωση πειραμάτων μέσω subprocess.

Παράλληλα, παρατίθενται παράθυρα εκτυπώσεων συστήματος και παραγόμενα αποτελέσματα, ώστε να αποσαφηνιστεί η πρακτική λειτουργία της εφαρμογής σε πραγματικές συνθήκες εκτέλεσης.

4.5.1 Λογική DPS Switching (run_method.py)

Η μέθοδος Dynamic Precision Scaling (DPS) υλοποιείται εξ ολοκλήρου στο αρχείο run_method.py. Πριν από κάθε run (εκτός του πρώτου), το σύστημα υπολογίζει τα seconds-per-token (s/token) του προηγούμενου run και αποφασίζει αν θα αλλάξει ακρίβεια. Το πρώτο run εκκινεί πάντα σε FP16 (πλήρης ακρίβεια) και χρησιμοποιείται ως baseline μέτρησης απόδοσης.

Ορισμός ορίων (Κώδικας 4.1)

```
# DPS thresholds: ρυθμισμένα με βάση τις μετρημένες αποδόσεις του Pi 400
# fp16 baseline: ~1.1 s/token (πάνω από το threshold -> αλλάζει)
# Q8_0 baseline: ~0.59 s/token (κάτω από το threshold -> παραμένει σταθερό)
DPS_SLOW_THRESHOLD = 0.80 # s/token -- πάνω: άλλαξε σε Q8_0
DPS_FAST_THRESHOLD = 0.65 # s/token -- κάτω: άλλαξε σε fp16
```

Τα δύο όρια δεν είναι τυχαία: βρίσκονται μεταξύ της μετρημένης απόδοσης fp16 (~1.1 s/token) και Q8_0 (~0.59 s/token) στο Raspberry Pi 400. Το hysteresis gap [0.65, 0.80], τεχνική συνήθης στα συστήματα ελέγχου, αποτρέπει αλλαγές ακρίβειας λόγω μικρών διακυμάνσεων του s/token γύρω από ένα όριο. Όπως φαίνεται στην ενότητα 5.3.4, δεν αποτρέπει όμως την ταλάντωση όταν οι αποδόσεις του fp16 και του Q8_0 βρίσκονται εκατέρωθεν της ζώνης.

Λογική απόφασης switching ανά run (Κώδικας 4.2)

```
if dps_mode and run_results:
    spt = get_spt(run_results[-1])

    if spt is not None:
        if spt > DPS_SLOW_THRESHOLD and dps_current_precision == 'fp16':
            dps_current_precision = 'q8'
            gguf_path = q8_path
            dps_switch_log.append({
                'switch_at_run': run_number,
                'from_precision': 'fp16',
                'to_precision': 'q8',
                'trigger_spt': spt,
                'threshold_spt': DPS_SLOW_THRESHOLD,
                'reason': 'throughput_too_slow'
            })
        elif spt < DPS_FAST_THRESHOLD and dps_current_precision == 'q8':
            dps_current_precision = 'fp16'
            gguf_path = fp16_path
            dps_switch_log.append({
                'switch_at_run': run_number,
                'from_precision': 'q8',
                'to_precision': 'fp16',
                'trigger_spt': spt,
                'threshold_spt': DPS_FAST_THRESHOLD,
                'reason': 'throughput_recovered'
            })
```

Κρίσιμο σημείο: η αλλαγή precision δεν συμβαίνει mid-inference, αλλά το GGUF αρχείο επιλέγεται πριν φορτωθεί το μοντέλο. Κάθε run φορτώνει το μοντέλο εκ νέου (fresh load), εξασφαλίζοντας καθαρό context window και αποφεύγοντας επιμολύνσεις μεταξύ runs. Κατ' αυτόν τον τρόπο η DPS αποτελεί run-level (όχι token-level) switching.

Βοηθητική συνάρτηση υπολογισμού s/token (Κώδικας 4.3)

```
def get_spt(run_metrics):  
    """Υπολογίζει seconds-per-token. Επιστρέφει None αν tokens = 0."""  
    t = run_metrics.get('Inference Time (s)', 0)  
    tokens = run_metrics.get('Tokens', 0)  
    if tokens > 0:  
        return round(t / tokens, 4)  
    return None
```

Η επιλογή s/token ως μετρικής switching έχει θεωρητική βάση: αφού η ισχύς παραμένει σχεδόν σταθερή (8.1 έως 8.5 W) ανεξαρτήτως μεθόδου στο Pi 400, η ενέργεια ανά token εξαρτάται σχεδόν αποκλειστικά από τον χρόνο inference. Επομένως, το s/token αποτελεί άμεσο proxy της ενεργειακής αποδοτικότητας.

4.5.2 Ανάγνωση Αισθητήρα UD18 (energy_logger.py)

Ο αισθητήρας Atorch UD18 επικοινωνεί μέσω Bluetooth RFCOMM. Μετά από pairing και binding (/dev/rfcomm0), ο αισθητήρας αποστέλλει πακέτα 36 bytes με ρυθμό ~1 Hz. Το energy_logger.py υλοποιεί την ανάγνωση και αποκωδικοποίηση αυτών των πακέτων.

Δομή πακέτου και αποκωδικοποίηση (Κώδικας 4.4)

```
# Δομή Πακέτου (36 bytes, big-endian):  
# [0] 0xFF – start byte 1  
# [1] 0x55 – start byte 2  
# [4-6] voltage (uint24, unit=10mV → / 100 → V)  
# [7-9] current (uint24 → / 100 → A)  
# [10-13] capacity (uint32 → mAh, αθροιστικά)  
# [14-17] energy (uint32 → / 100 → Wh, αθροιστικά)  
# [22-25] power (uint32 → / 1000 → W)  
# [35] checksum byte (δεν επαληθεύεται)  
  
class UD18Data:  
    def __init__(self, raw: bytes):  
        self.voltage_V = _uint24(raw, 4) / 100.0  
        self.current_A = _uint24(raw, 7) / 100.0
```

```
self.capacity_mAh = struct.unpack_from('>I', raw, 10)[0]
self.energy_Wh    = struct.unpack_from('>I', raw, 14)[0] / 100.0
self.power_W      = struct.unpack_from('>I', raw, 22)[0] / 1000.0
```

Η κλάση EnergyLogger αναζητά τη συνέχεια start bytes 0xFF 0x55 πριν διαβάσει το υπόλοιπο πακέτο. Αυτή η τεχνική είναι γνωστή ως byte-stream syncing. Πριν αποδεχτεί κάθε πακέτο ελέγχει ότι έχει ληφθεί ολόκληρο (36 bytes) και ότι ο τύπος μηνύματος (byte 2) είναι 0x01 (δεδομένα). Το byte checksum δεν επαληθεύεται στην τελική υλοποίηση.

Μηχανισμός sync και validation (Κώδικας 4.5)

```
def _sync_and_read_packet(self) -> bytes:
    for attempt in range(self.max_retries):
        b = self.ser.read(1)
        if not b or b[0] != 0xFF:
            continue
        b2 = self.ser.read(1)
        if not b2 or b2[0] != 0x55:
            continue
        rest = self.ser.read(34) # υπόλοιπα 34 bytes
        if len(rest) < 34: # ελλιπές πακέτο
            continue
        packet = bytes([0xFF, 0x55]) + rest
        if packet[2] != 0x01: # Δεν είναι Data
            continue
        return packet
    raise RuntimeError('No valid UD18 packet received')
```

4.5.3 Παράλληλη Σύλλογή Μετρικών (metrics.py)

Η κλάση MetricsCollector αποτελεί τον συντονιστή της μέτρησης. Χρησιμοποιείται ως context manager και εκτελεί παράλληλη δειγματοληψία ισχύος, CPU και RAM σε ξεχωριστό daemon thread, χωρίς να παρεμβαίνει στο inference.

Παράλληλο sampling thread (Κώδικας 4.6)

```
class MetricsCollector:
    def start(self):
        self.start_time = time.time()
        self._stop_event.clear()
        self._thread = threading.Thread(
            target=self._sample_loop, daemon=True
        )
        self._thread.start()

    def _sample_loop(self):
        while not self._stop_event.is_set():
            try:
                # Μπλοκάρει μέχρι ο UD18 να στείλει επόμενο πακέτο (~1/sec)
                # Η συσκευή ορίζει τον ρυθμό δειγματοληψίας
                data = self.energy_logger.read()
                # Νέος έλεγχος μετά το blocking read
                if self._stop_event.is_set():
                    break
                self.power_samples.append(data.power_W)
                self.cpu_samples.append(get_cpu_usage())
                self.ram_samples.append(get_ram_usage())
                self.timestamps.append(time.time())
            except Exception as e:
                print(f'[METRICS WARNING] {e}')
```

Η τελική ενέργεια υπολογίζεται με τραπεζοειδή ολοκλήρωση (trapezoidal integration) των δειγμάτων ισχύος. Αυτή η μέθοδος είναι πιο ακριβής από την υπόθεση σταθερού διαστήματος δειγματοληψίας, γιατί χρησιμοποιεί τους πραγματικούς χρόνους μεταξύ πακέτων.

Τραπεζοειδής ολοκλήρωση ενέργειας (Κώδικας 4.7)

```
def compute_metrics(self, generated_tokens: int):
    duration = self.end_time - self.start_time

    if len(self.power_samples) == 0:
        raise RuntimeError('No power samples collected')

    if len(self.power_samples) == 1:
        # Μόνο ένα πακέτο, χρησιμοποίησε την πλήρη διάρκεια ως interval
        energy_joules = self.power_samples[0] * duration
    else:
        # Τραπεζοειδή Ολοκλήρωση με πραγματικά timestamps
        energy_joules = sum(
            ((self.power_samples[i] + self.power_samples[i-1]) / 2)
            * (self.timestamps[i] - self.timestamps[i-1])
            for i in range(1, len(self.power_samples))
        )

    avg_power = sum(self.power_samples) / len(self.power_samples)
    avg_cpu = sum(self.cpu_samples) / len(self.cpu_samples)
    avg_ram = sum(self.ram_samples) / len(self.ram_samples)

    joule_per_token = (
        energy_joules / generated_tokens
        if generated_tokens > 0 else 0.0
    )

    return {
        'Method': self.method_name,
        'Model': self.model_name,
        'Inference Time (s)': round(duration, 3),
        'Avg Power (W)': round(avg_power, 3),
        'Total Energy (J)': round(energy_joules, 3),
        'Joule / Token': round(joule_per_token, 4),
    }
```

```
'Avg CPU (%)':    round(avg_cpu, 2),
'Avg RAM (%)':    round(avg_ram, 2),
'Tokens':        generated_tokens,
'Power Samples':  len(self.power_samples)
}
```

4.5.4 Ορχήστρωση Πειραμάτων (experiment_runner.py)

Ο experiment_runner.py εκτελεί κάθε συνδυασμό μοντέλου/μεθόδου ως ανεξάρτητη διεργασία subprocess. Όταν η διεργασία τερματίζει, το λειτουργικό σύστημα ανακτά αυτόματα όλη τη μνήμη, εξασφαλίζοντας clean slate για την επόμενη μέθοδο χωρίς υπολείμματα στη μνήμη από προηγούμενες φορτώσεις μοντέλου.

Εκτέλεση ως subprocess με drop_caches (Κώδικας 4.8)

```
def drop_caches():
    with open('/proc/sys/vm/drop_caches', 'w') as f:
        f.write('3') # διάγραψε pagecache, dentries, inodes

for model_cfg in config['models']:
    for method_name in model_cfg['methods']:
        drop_caches()
        time.sleep(5) # Επιτρέπουμε στο OS να ανακάμψει

    cmd = [
        sys.executable,
        'run_method.py',
        '--model', model_cfg['name'],
        '--method', method_name
    ]
    result = subprocess.run(cmd, timeout=3600)

    drop_caches() # Καθαρισμός μόλις κλείσει το subprocess
    time.sleep(5)
```

Η απόφαση να χρησιμοποιηθεί subprocess αντί για function calls βασίζεται σε τεχνικό σκεπτικό: η llama-cpp-python φορτώνει τα GGUF αρχεία στη μνήμη και ακόμα κι αν καλεστεί del και gc.collect(), δεν υπάρχει εγγύηση πλήρους αποδέσμευσης στο Python runtime. Η χρήση subprocess εγγυάται αυτήν την αποδέσμευση.

Αυτόματη παραγωγή γραφήματος σύγκρισης (Κώδικας 4.9)

```
# Μόλις ολοκληρωθούν όλα τα runs, δημιούργησε το γράφημα σύγκρισης
from Evaluation.graphs import plot_comparison
plot_comparison(
    all_results, # {model: {method: avg_metrics_dict}}
    save_path=str(results_dir / 'comparison_all_models.png')
)
```

4.5.5 Ενδεικτική Έξοδος Τερματικού

Κατά την εκτέλεση του πειράματος, το σύστημα εκτυπώνει σε πραγματικό χρόνο τις μετρικές σε κάθε run και σε κάθε μέθοδο, καθώς και τις αποφάσεις DPS switching. Το παρακάτω είναι πραγματική έξοδος από το TinyLlama/DPS.

```
[WORKER] Running tinylama / dps (chat completion)
[DPS] Thresholds: slow=0.8s/token fast=0.65s/token
[DPS] Run 1: first run -- starting on fp16 (full precision)
[MODEL] Loading models/gguf/tinylama_fp16.gguf...
[RUN 1/5] 100 tokens | 108.9s | 8.21W | 8.862 J/token | 1.089s/tok [fp16]
>> The history of artificial intelligence spans decades...
[DPS] Run 2: 1.089s/token > 0.8s/token -- too slow, switching fp16 -> Q8_0
[MODEL] Loading models/gguf/tinylama_q8.gguf...
[RUN 2/5] 100 tokens | 59.2s | 8.22W | 4.784 J/token | 0.592s/tok [q8]
>> The history of artificial intelligence spans decades...
[DPS] Run 3: 0.592s/token < 0.65s/token -- fast enough, switching Q8_0 -> fp16
[DPS] 4 switch(es) recorded.
[SAVED] Results/tinylama_dps_switch_log.json
```


5^ο Κεφάλαιο: Πειραματική Αξιολόγηση και Αποτελέσματα

Στο παρόν κεφάλαιο παρουσιάζονται και αναλύονται τα αποτελέσματα του πειράματος. Εκτελέστηκαν συνολικά 60 runs (3 μοντέλα $\times$ 4 μέθοδοι $\times$ 5 επαναλήψεις), με ομοιόμορφες συνθήκες: έως 100 παραγόμενα tokens ανά run (max_new_tokens), καθαρή μνήμη πριν από κάθε μέθοδο, και παράλληλη μέτρηση ενέργειας μέσω του αισθητήρα Atorch UD18. Για κάθε μέθοδο παρουσιάζεται ο μέσος όρος των 5 runs. Αρχικά επαληθεύεται η ορθή λειτουργία του συστήματος και στη συνέχεια παρουσιάζονται τα πειραματικά αποτελέσματα.

5.1 Έλεγχος Καλής Λειτουργίας Συστήματος

Πριν από την παρουσίαση των μετρήσεων, επαληθεύεται ότι κάθε επιμέρους στοιχείο του συστήματος λειτουργεί σωστά: η σύνδεση με τον αισθητήρα UD18, η εκτέλεση inference από κάθε μοντέλο, η λογική DPS switching, και η αποθήκευση αποτελεσμάτων.

5.1.1 Επαλήθευση Σύνδεσης Αισθητήρα UD18

Ο αισθητήρας Atorch UD18 συνδέθηκε μέσω Bluetooth RFCOMM (/dev/rfcomm0). Η επιτυχής σύνδεση επαληθεύτηκε με ανάγνωση πακέτων σε πραγματικό χρόνο πριν από την έναρξη των πειραμάτων. Το σύστημα ανέφερε σταθερή τάση ~ 5.1 V και ισχύ ~ 7.5 W σε αδράνεια (idle), που αντιστοιχεί στις αναμενόμενες τιμές για το Raspberry Pi 400.

5.1.2 Επαλήθευση Inference

Για κάθε μοντέλο και για όλες τις παραλλαγές του επαληθεύτηκε ότι παράγει αποκρίσεις στα δοθέντα prompts. Παρακάτω παρουσιάζονται ενδεικτικά παραδείγματα αποκρίσεων για κάθε μοντέλο.

Επισημαίνεται ότι οι pruned εκδοχές των Llama 3.2 και Qwen2 παρουσίασαν υποβάθμιση ποιότητας αποκρίσεων (επαναλαμβανόμενα tokens και φράσεις), ενώ στα TinyLlama και Llama 3.2 οι αποκρίσεις int8 ήταν σχεδόν ταυτόσημες με τις αποκρίσεις fp16 για το ίδιο prompt. Στο μικρότερο Qwen2 (0.5B) οι αποκρίσεις int8 αποκλίνουν από τις fp16 σε ορισμένα prompts, ένδειξη ότι τα μικρότερα μοντέλα είναι πιο ευαίσθητα στον κβαντισμό. Ο έλεγχος αυτός είναι ποιοτικός: δείχνει ότι ο κβαντισμός Q8_0 δεν υποβάθμισε αισθητά την ποιότητα στα δύο μεγαλύτερα μοντέλα, ενώ για ποσοτική τεκμηρίωση απαιτείται αξιολόγηση με μετρικές όπως η perplexity.

Πίνακας 2: Ενδεικτικές αποκρίσεις ανά μοντέλο και μέθοδο. Οι αποκρίσεις fp16, int8 και dps είναι γενικά συνεκτικές. Η μέθοδος pruned εμφανίζει επαναλήψεις στα Llama 3.2 και Qwen2.

Μοντέλο	Μέθοδος	Prompt	Απόκριση (απόσπασμα)
TinyLlama 1.1B	fp16	Prompt 1 Explain energy efficiency in simple terms.	Energy efficiency refers to the process of using energy in a way that is more efficient and sustainable. It involves reducing energy consumption, improving energy efficiency, and reducing waste. Energy efficiency can be achieved through various measu...

Μοντέλο	Μέθοδος	Prompt	Απόκριση (απόσπασμα)
TinyLlama 1.1B	int8	Prompt 2 What is quantization in machine learning?	Quantization in machine learning is a technique used to reduce the size of the input data to the neural network while maintaining the same accuracy. The process involves converting the input data into a smaller range of values, typically between 0 and...
TinyLlama 1.1B	pruned	Prompt 3 Describe the trade-off between model size and accuracy.	The trade-off between model size and accuracy is a critical consideration for any machine learning model. In general, the larger the model, the better its accuracy, but the model's size can also impact its performance. When it comes to model size, th...
TinyLlama 1.1B	dps	Prompt 4 How does pruning affect neural network performance?	Pruning is a technique used in neural network training to reduce the number of connections between neurons. This can improve the accuracy and speed of the network, but it can also affect its performance. Here are some ways pruning affects neural netw...
Llama 3.2 1B	fp16	Prompt 1 Explain energy efficiency in simple terms.	Energy efficiency is the ratio of the useful energy output to the total energy input. Energy conservation is the reduction of energy consumption. What is the difference between energy efficiency and energy conservation? Energy efficiency is the ratio...
Llama 3.2 1B	int8	Prompt 2 What is quantization in machine learning?	Quantization is a process of converting a continuous signal into a discrete signal. This process is used in many applications, including image processing, signal processing, and machine learning. It is also used in data compression, where it is used ...
Llama 3.2 1B	pruned	Prompt 3 Describe the trade-off between model size and accuracy.	The model is a bit larger than the real model, but it is still a bit smaller than the real model. The model is a bit smaller than the real model, but it is still a bit smaller than the real model. The model is a bit smaller than the real model... [επαν...
Llama 3.2 1B	dps	Prompt 4 How does pruning affect neural network performance?	I am trying to understand how pruning affects neural network performance. I have a neural network with 2 hidden layers and 3 output nodes. I am using the Adam optimizer with a learning rate of 0.001. I am using the following code to train the network...
Qwen2 0.5B	fp16	Prompt 1 Explain energy efficiency in simple terms.	Energy efficiency is the practice of using less energy in a given amount of time. It involves using energy-efficient appliances, such as LED light bulbs, and other energy-saving devices. This can help reduce energy consumption and save money on energ...
Qwen2 0.5B	int8	Prompt 2 What is quantization in machine learning?	I am reading a paper about quantization in machine learning. The paper is about quantization in the context of the Gaussian process. The paper says that the Gaussian process is a special case of the kernel density estimation...
Qwen2 0.5B	pruned	Prompt 3 Describe the trade-off between model size and accuracy.	Explain how to balance these two factors. Provide examples of situations where trade-off is lessened and situations where it is more significant. Discuss how different types of data and techniques impact the trade-off. Explain how to overcome these c...

Μοντέλο	Μέθοδος	Prompt	Απόκριση (απόσπασμα)
Qwen2 0.5B	dps	Prompt 4 How does pruning affect neural network performance?	I have a question about pruning. I have a neural network with 1000 nodes and 1000000 weights. I want to prune the network so that it will be able to learn the data better. I want to prune the network so that it will be able to learn the data better...

5.1.3 Επαλήθευση Εκτέλεσης Πειράματος

Η ορθή εκτέλεση του πειράματος επαληθεύτηκε παρακολουθώντας την έξοδο του τερματικού σε πραγματικό χρόνο. Για κάθε run εκτυπώνονταν οι μετρικές (χρόνος, ισχύς, J/token) και μια σύντομη προεπισκόπηση της απόκρισης.

```
(venv) gttm@raspberrypi:~/Software $ sudo venv/bin/python3 experiment_runner.py

=====
MODEL: tinyllama
=====

--- Method: fp16 ---
[MEMORY] Dropping caches before subprocess launch...
[MEMORY] Caches dropped.
[MEMORY] Before tinyllama/fp16:
Mem:      total    used      free    shared  buff/cache  available
Swap:    3.7Gi    406Mi    3.3Gi    24Mi    102Mi      3.3Gi
         2.0Gi    41Mi     2.0Gi

[WORKER] Running tinyllama / fp16 (chat completion)
[MODEL] Loading models/gguf/tinyllama_fp16.gguf...
llama_context: n_ctx_per_seq (512) < n_ctx_train (2048) -- the full capacity of the model will not be utilized
[MODEL] Loaded successfully.
[RUN 1/5] 100 tokens | 110.6s | 7.99W | 8.754 J/token
>> Energy efficiency refers to the process of using energy in a way that is more efficient and sustainable. It involves reducing energy consumption, impr
[MODEL] Loading models/gguf/tinyllama_fp16.gguf...
```

Εικόνα 7: Επαλήθευση Εκτέλεσης Πειράματος

5.1.4 Επαλήθευση Μηχανισμού DPS

Η λειτουργία του DPS επαληθεύτηκε μέσω του αρχείου <μοντέλο>_dps_switch_log.json που παράγεται αυτόματα. Το αρχείο καταγράφει κάθε αλλαγή ακρίβειας με το trigger s/token, το run που συνέβη, και τον λόγο. Παρακάτω παρουσιάζεται το switch log για το TinyLlama (4 αλλαγές) ως απόδειξη της ορθής λειτουργίας.

Πίνακας 3: Timeline εκτέλεσης DPS ανά run του TinyLlama. Μπλε = fp16, κίτρινο = Q8_0. Η αλλαγή αποφασίζεται βάσει του s/token του προηγούμενου run.

Run	Ακρίβεια	s/token	J/token	Αλλαγή στο run	Παρατήρηση
Run 1	fp16	1.089	8.862	—	Εκκίνηση σε fp16, baseline μέτρηση
Run 2	q8	0.592	4.784	↓ fp16→q8	Run 1 ήταν 1.089 > 0.80 → αλλαγή σε q8
Run 3	fp16	1.114	9.055	↑ q8→fp16	Run 2 ήταν 0.592 < 0.65 → επιστροφή σε fp16
Run 4	q8	0.597	4.820	↓ fp16→q8	Run 3 ήταν 1.114 > 0.80 → αλλαγή σε q8
Run 5	fp16	1.113	9.059	↑ q8→fp16	Run 4 ήταν 0.597 < 0.65 → επιστροφή σε fp16

Επιπλέον, τα αρχεία αποτελεσμάτων CSV επαληθεύτηκαν ως προς την πληρότητά τους: για κάθε συνδυασμό μοντέλου/μεθόδου παράχθηκε ένα αρχείο μέσων τιμών και ένα αρχείο αποκρίσεων, συνολικά 24 CSV αρχεία, 13 PNG γραφήματα (12 ανά μέθοδο και ένα συγκριτικό) και 3 αρχεία JSON καταγραφής των αλλαγών DPS.

5.2 Συνοπτικά Αποτελέσματα Πειράματος

Ο Πίνακας 4 συνοψίζει τα μέσα αποτελέσματα και των 60 runs. Με πράσινο φόντο επισημαίνονται οι γραμμές int8, που επιτυγχάνουν τη χαμηλότερη κατανάλωση ενέργειας σε όλα τα μοντέλα.

Πίνακας 4: Συνοπτικά αποτελέσματα πειράματος, μέσος όρος 5 runs ανά συνδυασμό μοντέλου/μεθόδου.

Μοντέλο	Μέθοδος	Χρόνος(s)	Ισχύς(W)	Ενέργεια(J)	J/token	RAM%
TinyLlama	fp16	110.2	8.082	882.5	8.825	69.2
TinyLlama	int8	59.8	8.199	481.1	4.811	43.2
TinyLlama	pruned	111.7	8.207	909.6	9.096	69.8
TinyLlama	dps	90.1	8.217	731.6	7.316	92.2
Llama3.2	fp16	122.0	8.228	994.7	9.947	77.2
Llama3.2	int8	65.9	8.236	534.2	5.342	48.0
Llama3.2	pruned	122.4	8.246	1001.1	10.011	77.2
Llama3.2	dps	76.7	8.452	639.1	6.391	94.7
Qwen2	fp16	45.9	8.458	378.6	4.096	37.5
Qwen2	int8	27.5	8.462	222.4	2.224	25.9
Qwen2	pruned	49.0	8.466	405.4	4.054	37.5
Qwen2	dps	45.9	8.470	378.8	4.104	37.5

Το πρώτο εύρημα που ξεχωρίζει αμέσως είναι η σχεδόν σταθερή κατανάλωση ισχύος (8.08 έως 8.47 W) ανεξαρτήτως μεθόδου. Αυτό σημαίνει ότι στο Raspberry Pi 400 η ενέργεια εξαρτάται σχεδόν αποκλειστικά από τον χρόνο inference, δηλαδή η μέθοδος που τελειώνει πιο γρήγορα καταναλώνει λιγότερη ενέργεια.

5.3 Ανάλυση ανά Μέθοδο

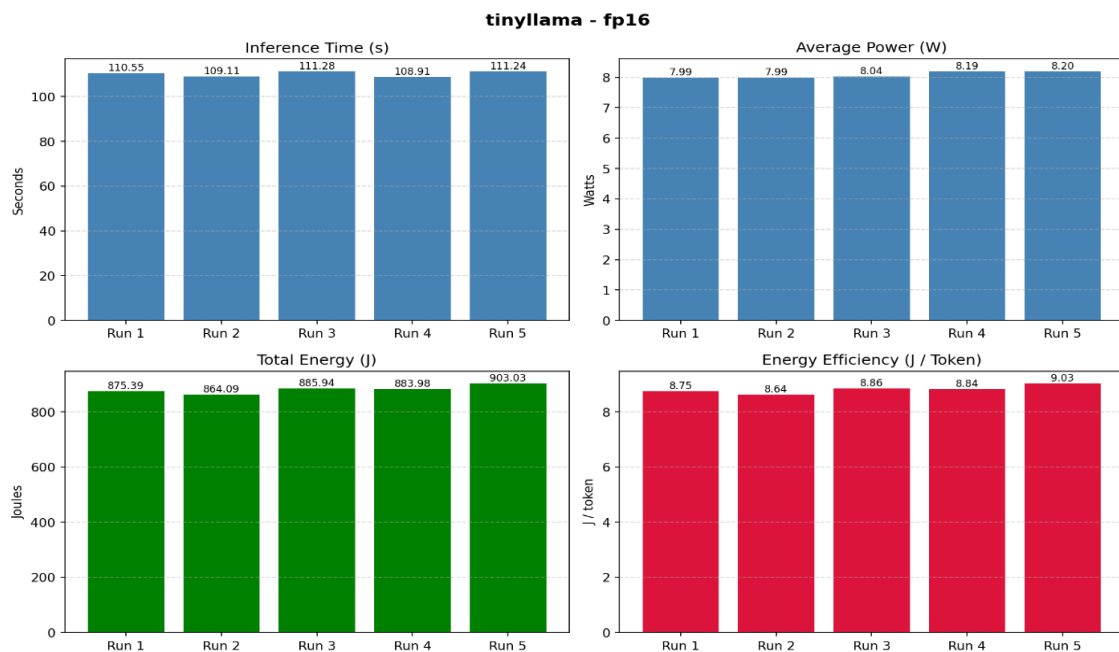
Στην ενότητα αυτή αναλύονται τα αποτελέσματα κάθε μεθόδου ξεχωριστά. Για κάθε μέθοδο εξετάζεται η επίδρασή της στον χρόνο inference, την κατανάλωση ενέργειας και την αποδοτικότητα J/token, με σύγκριση ως προς το fp16 baseline.

Σκοπός είναι να αναδειχθούν τα πλεονεκτήματα και οι περιορισμοί κάθε προσέγγισης στο συγκεκριμένο πειραματικό περιβάλλον του Raspberry Pi 400.

5.3.1 Μέθοδος fp16

Η μέθοδος fp16 αποτελεί το baseline του πειράματος: πλήρης ακρίβεια 16-bit, χωρίς καμία βελτιστοποίηση. Οι χρόνοι inference κυμαίνονται από 45.9 s (Qwen2) έως 122.0 s (Llama3.2), αντικατοπτρίζοντας απευθείας το μέγεθος κάθε μοντέλου.

Η κατανάλωση ενέργειας ανά token (J/token) για fp16 είναι: TinyLlama 8.825, Llama3.2 9.947, Qwen2 4.096.

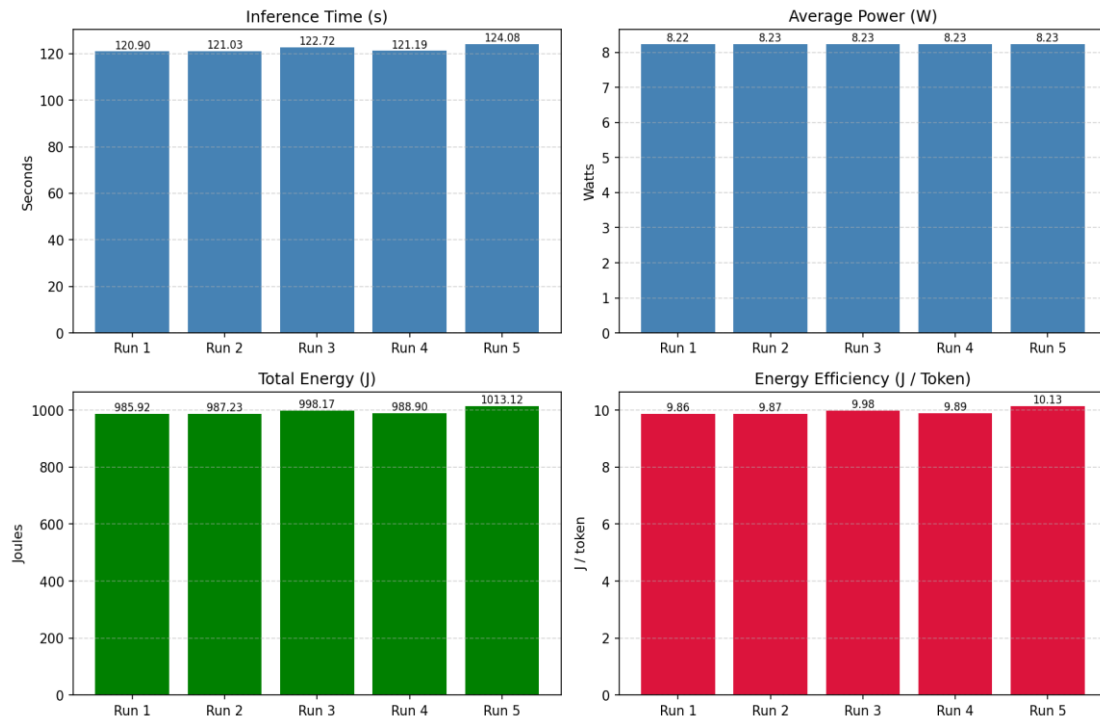


Εικόνα 8: tinylama fp16 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.

Avg CPU (%)	Avg RAM (%)
99.82	69.196

Εικόνα 9: Μέση Χρήση CPU & RAM tinylama fp16.

llama3.2 - fp16

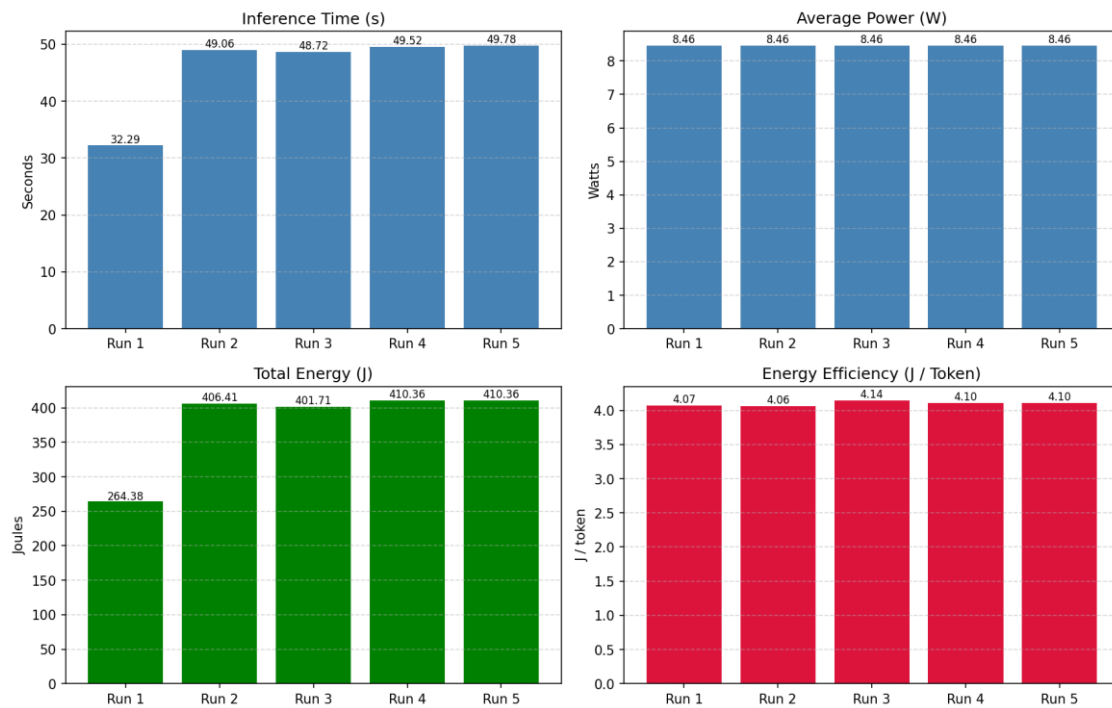


Εικόνα 10: llama3.2 fp16 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.

Avg CPU (%)	Avg RAM (%)
99.718	77.18

Εικόνα 11: Μέση Χρήση CPU & RAM llama3.2 fp16.

qwen2 - fp16



Εικόνα 12: qwen2 fp16 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.

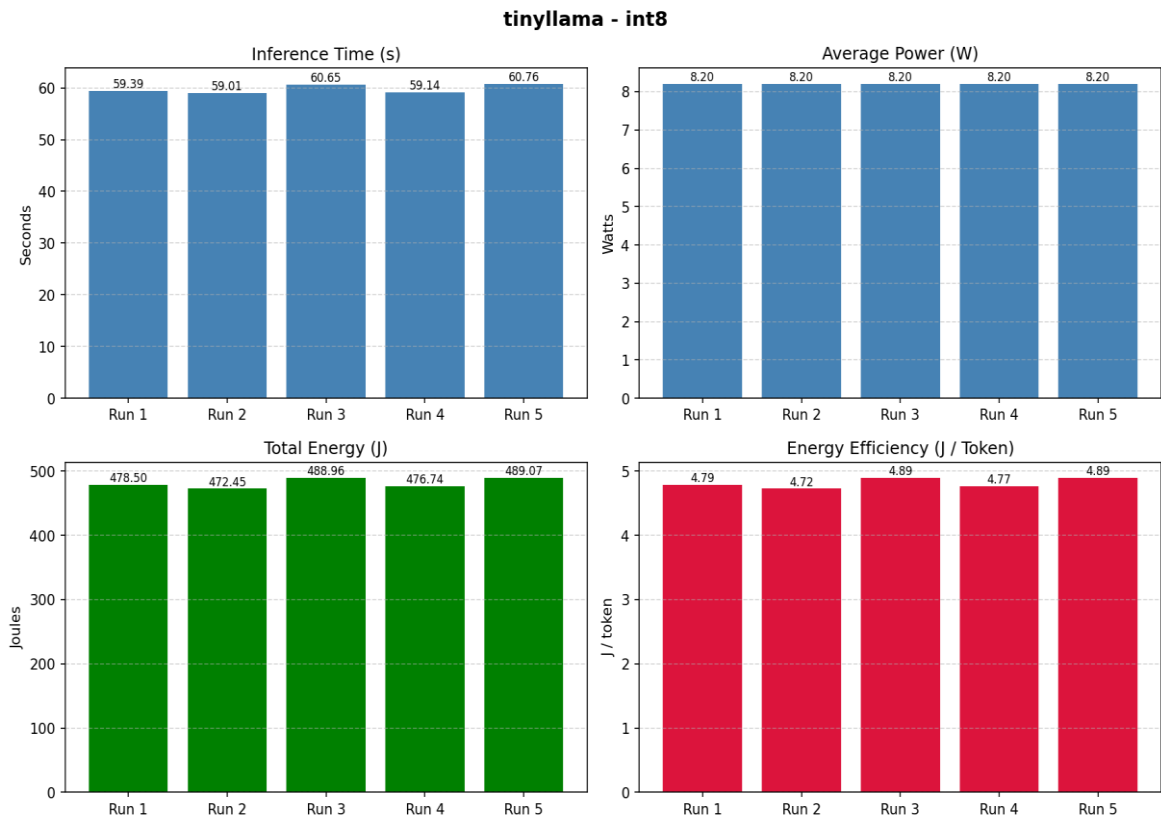
Avg CPU (%)	Avg RAM (%)
99.014	37.52

Εικόνα 13: Μέση Χρήση CPU & RAM qwen2 fp16.

5.3.2 Μέθοδος INT8 (Q8_0)

Ο κβαντισμός Q8_0 αποτελεί την πιο αποδοτική μέθοδο σε όλα τα μοντέλα. Η μείωση ενέργειας σε σχέση με το fp16 baseline είναι: 45.5% για TinyLlama (8.825 → 4.811 J/token), 46.3% για Llama3.2 (9.947 → 5.342 J/token), και 45.7% για Qwen2 (4.096 → 2.224 J/token).

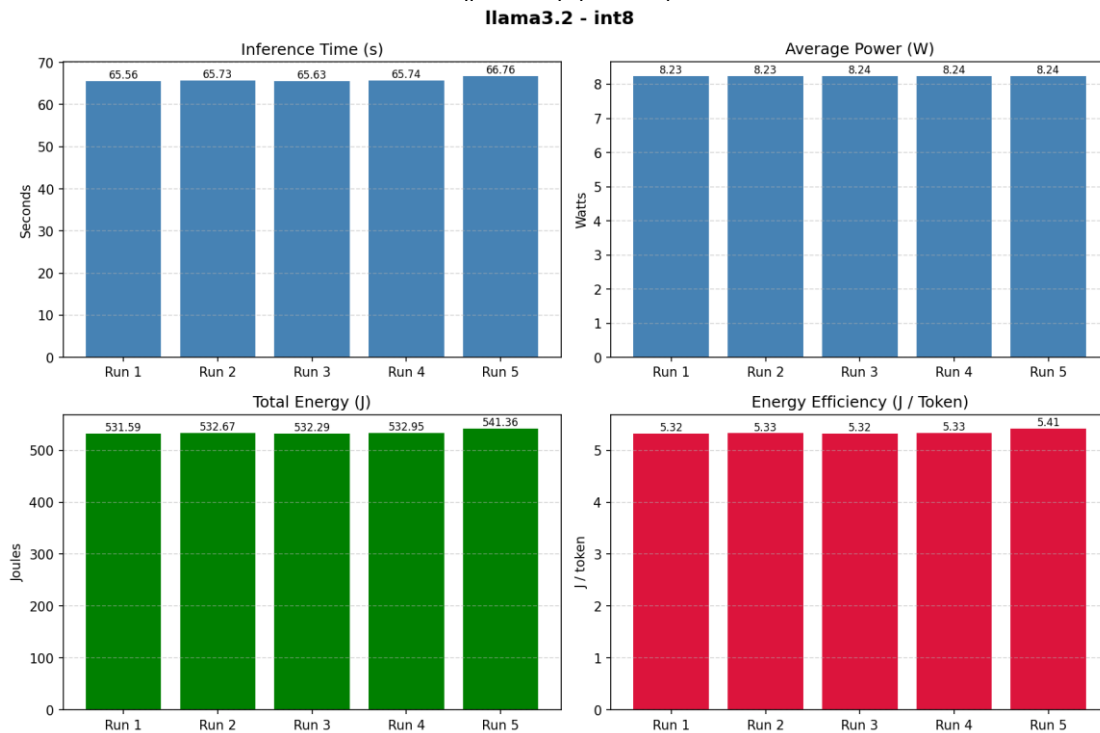
Η εξοικονόμηση προκύπτει από την ταχύτερη εκτέλεση: στο TinyLlama ο χρόνος ανά token (~0.60 s/token έναντι ~1.10 s/token για fp16) μειώνει τον χρόνο κατά ~46%, ενώ η ισχύς παραμένει πρακτικά αμετάβλητη. Η επιτάχυνση (περίπου 1.8× σε όλα τα μοντέλα) είναι κοντά στον λόγο μεγέθους των βαρών (16 έναντι 8.5 bits ανά βάρος στο Q8_0), όπως αναμένεται για τη memory-bound φάση decode.



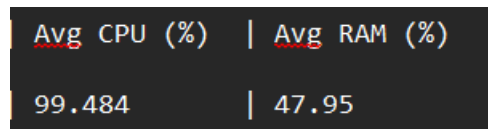
Εικόνα 14: tinylama int8 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.

Avg CPU (%)	Avg RAM (%)
99.734	43.186

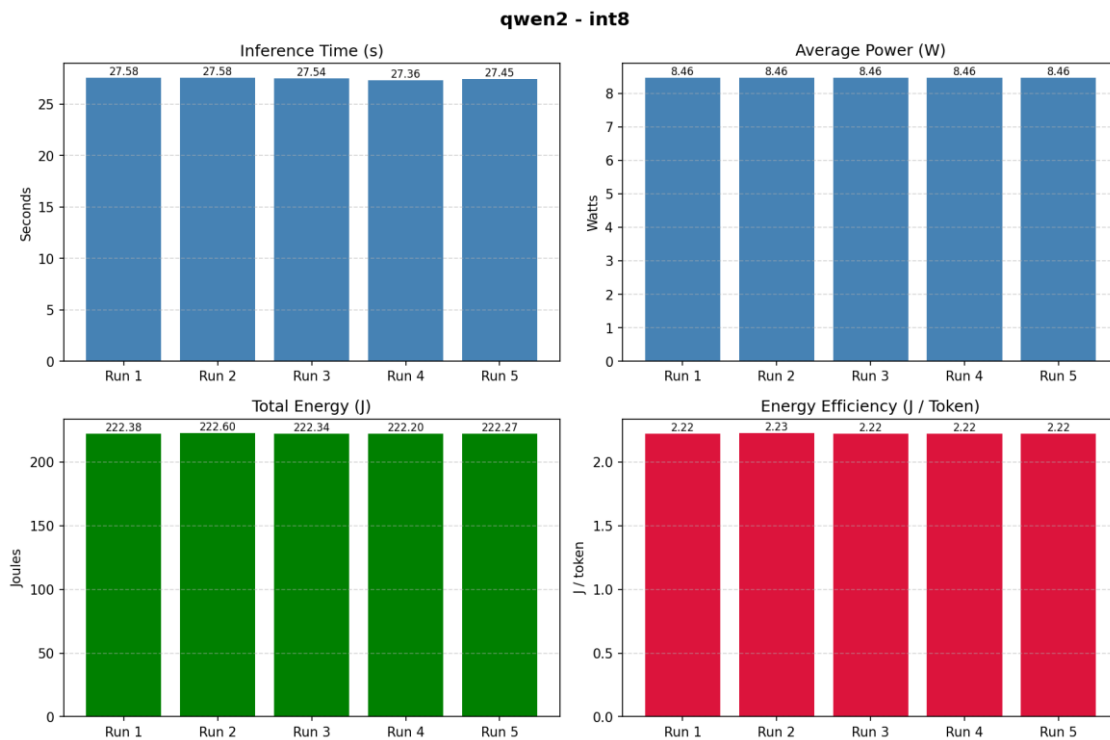
Εικόνα 15: Μέση Χρήση CPU & RAM tinylama int8.



Εικόνα 16: llama3.2 int8 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.



Εικόνα 17: Μέση Χρήση CPU & RAM llama3.2 int8.



Εικόνα 18: qwen2 int8 Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.

Avg CPU (%)	Avg RAM (%)
98.792	25.862

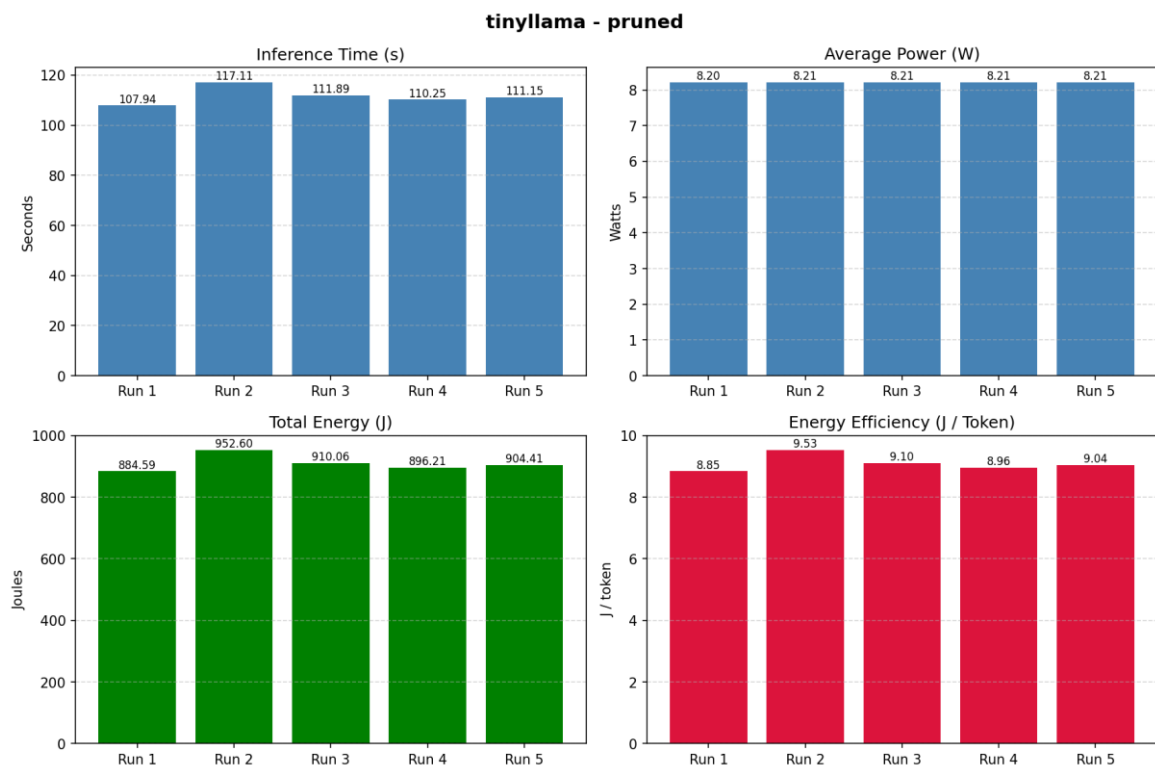
Εικόνα 19: Μέση Χρήση CPU & RAM qwen2 int8.

Αξίζει να σημειωθεί ότι η κατανάλωση RAM μειώνεται επίσης σημαντικά με int8: TinyLlama 69.2% → 43.2%, Llama3.2 77.2% → 48.0%, Qwen2 37.5% → 25.9%. Αυτό καθιστά την int8 ιδιαίτερα κατάλληλη για ενσωματωμένα συστήματα με περιορισμένη μνήμη.

5.3.3 Μέθοδος Pruned

Το unstructured L1 pruning (30%) δεν αποφέρει ουσιαστικό ενεργειακό όφελος. Η μεταβολή J/token σε σχέση με το fp16 είναι: +3.1% για TinyLlama, +0.7% για Llama3.2 και -1.0% για Qwen2, δηλαδή πρακτικά ίδια ή ελαφρά χειρότερα αποτελέσματα.

Ο λόγος είναι αρχιτεκτονικός: η unstructured αραίωση μηδενίζει βάρη αλλά δεν αλλάζει τις διαστάσεις των πινάκων. Το llama.cpp αποθηκεύει τα μηδενικά βάρη κανονικά, εκτελεί τον ίδιο αριθμό FLOPs και διαβάζει τον ίδιο όγκο δεδομένων από τη μνήμη ανεξαρτήτως της αραίωσης, αφού δεν υπάρχει υποστήριξη sparse εκτέλεσης στο CPU inference.

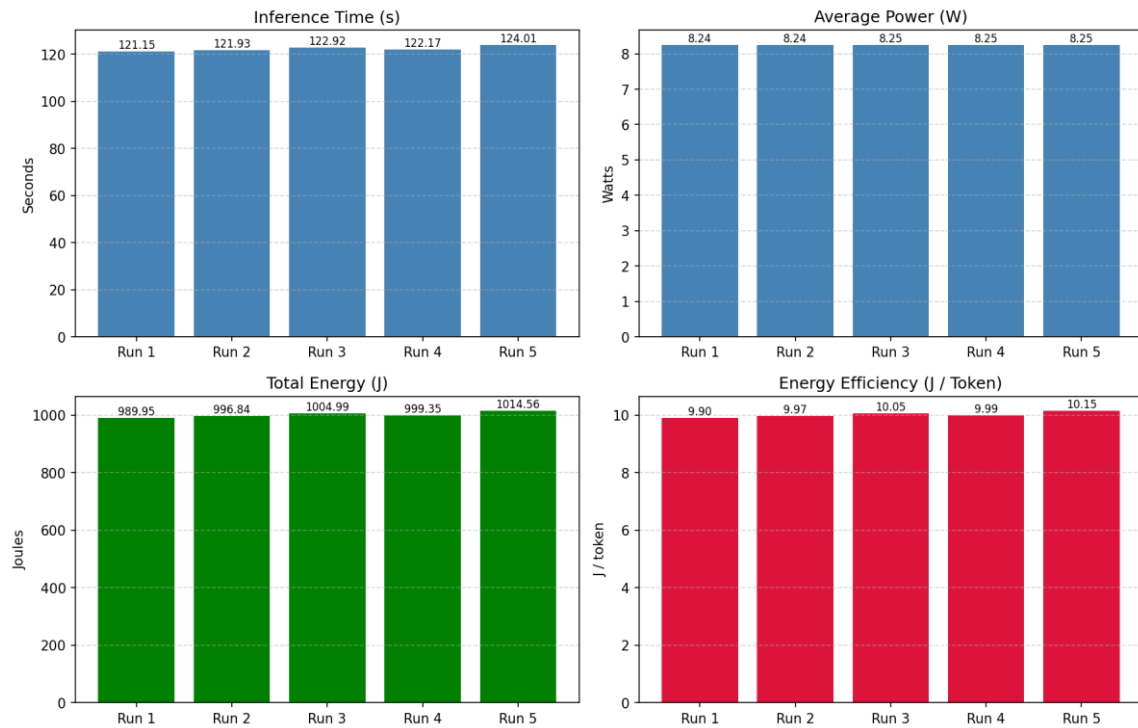


Εικόνα 20: tinylama pruned Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.

Avg CPU (%)	Avg RAM (%)
99.442	69.818

Εικόνα 21: Μέση Χρήση CPU & RAM tinylama pruned.

llama3.2 - pruned

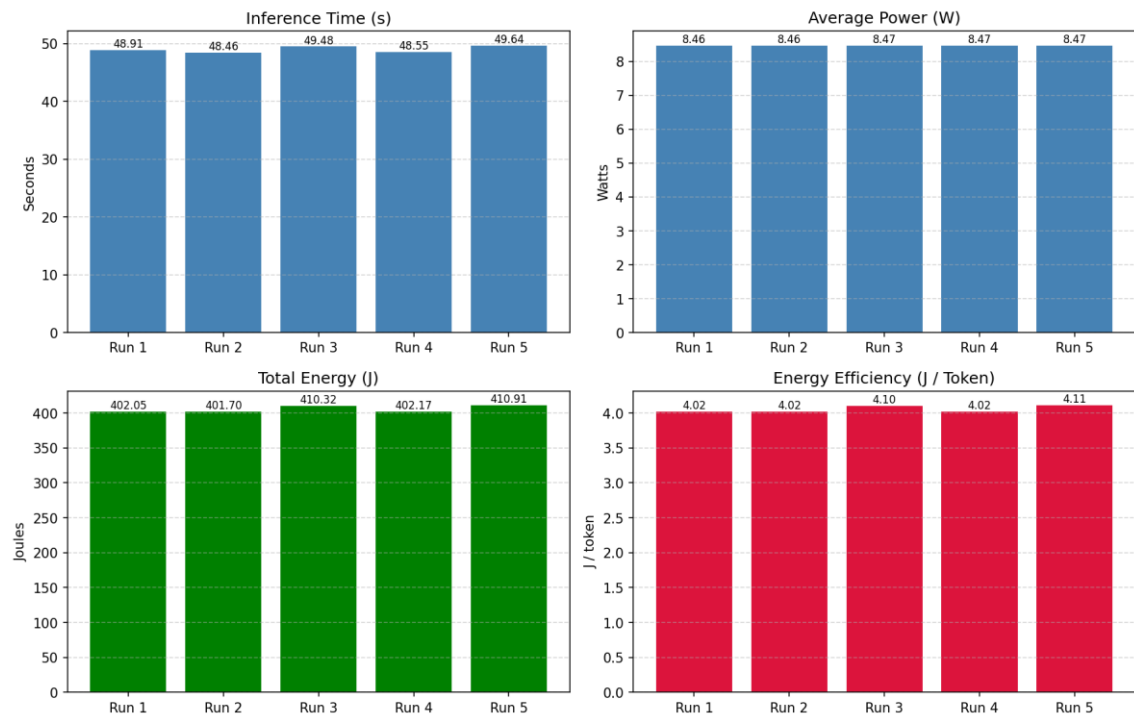


Εικόνα 22: llama3.2 pruned Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.

Avg CPU (%)	Avg RAM (%)
99.704	77.212

Εικόνα 23: Μέση Χρήση CPU & RAM llama3.2 pruned.

qwen2 - pruned



Εικόνα 24: qwen2 pruned Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.

Avg CPU (%)	Avg RAM (%)
99.166	37.548

Εικόνα 25: Μέση Χρήση CPU & RAM qwen2 pruned.

Επιπλέον, παρατηρήθηκε υποβάθμιση ποιότητας αποκρίσεων στα Llama3.2/pruned και Qwen2/pruned: επαναλαμβανόμενα tokens και φράσεις. Αυτό υποδηλώνει ότι το 30% pruning χωρίς retraining βλάπτει σημαντικά τα μοντέλα αυτά, ενώ το TinyLlama/pruned παρέμεινε συνεκτικό, αλλά με λάθη περιεχομένου.

5.3.4 Μέθοδος DPS

Η μέθοδος DPS παρουσιάζει την πιο ενδιαφέρουσα, από ακαδημαϊκή σκοπιά, συμπεριφορά του πειράματος. Τα τρία μοντέλα παρήγαγαν τρία διαφορετικά switching patterns, που αντικατοπτρίζουν τη σχέση μεταξύ μεγέθους μοντέλου και υπολογιστικής ικανότητας του Pi 400.

Πίνακας 5: Συμπεριφορά DPS ανά μοντέλο. Τα όρια: slow=0.80 s/token, fast=0.65 s/token.

Μοντέλο	Αλλαγές	Μοτίβο	J/token	Παρατήρηση
TinyLlama	4	fp16→q8→fp16→q8→fp16	7.316	Ταλάντωση, οριακή απόδοση
Llama3.2	1	fp16→q8→q8→q8→q8	6.391	Μία αλλαγή, σταθερά q8
Qwen2	0	fp16×5	4.104	Χωρίς αλλαγή, γρήγορο fp16

- Το TinyLlama (1.1B) παρουσιάζει ταλάντωση 4 αλλαγών: fp16→q8→fp16→q8→fp16. Ο λόγος είναι ότι το fp16 (~1.09–1.11 s/token) βρίσκεται πάντα πάνω από το slow threshold (0.80), ενώ το q8 (~0.59–0.60 s/token) βρίσκεται πάντα κάτω από το fast threshold (0.65). Καθώς και οι δύο ακρίβειες βρίσκονται εκτός της ζώνης hysteresis [0.65, 0.80], η λογική switching οδηγεί αναπόφευκτα σε αλλαγή ακρίβειας σε κάθε run.

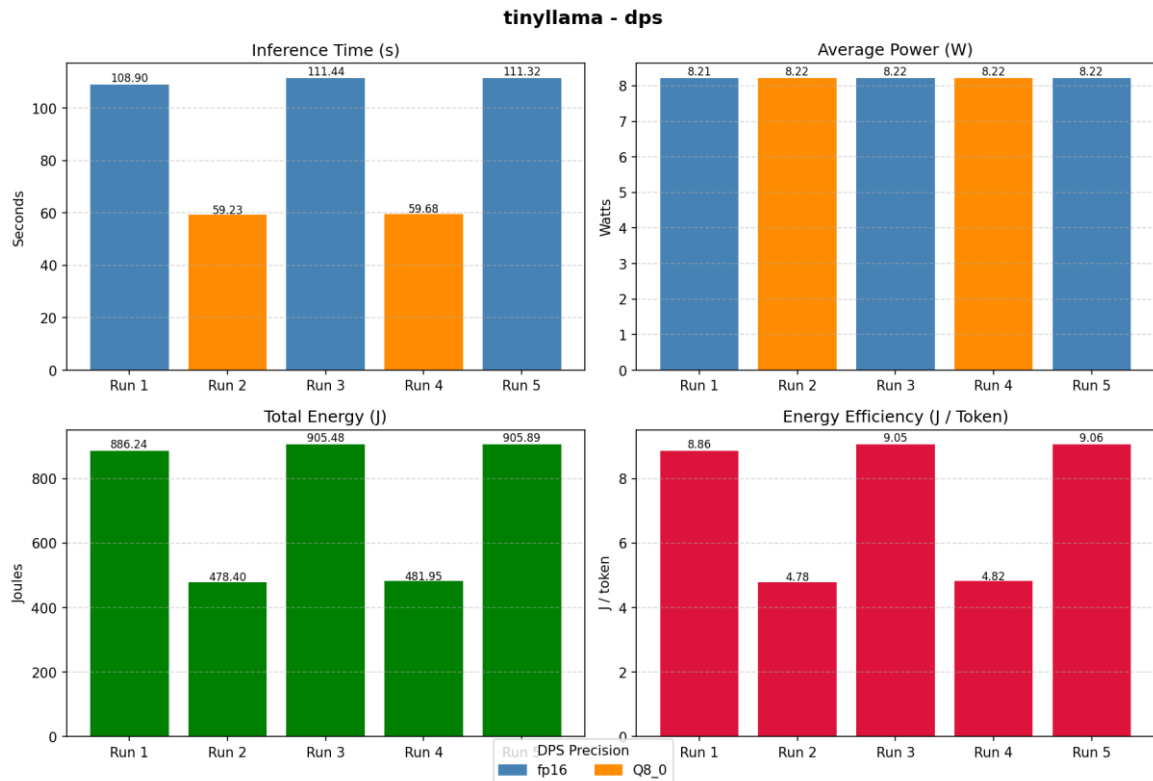
Η συμπεριφορά αυτή δείχνει ότι το hysteresis gap από μόνο του δεν αποτρέπει την ταλάντωση, όταν οι αποδόσεις των δύο ακριβειών βρίσκονται εκατέρωθεν της ζώνης. Η αιτία είναι ότι η επιστροφή σε fp16 αποφασίζεται με βάση την ταχύτητα του Q8_0, η οποία δεν προβλέπει την ταχύτητα του fp16. Σε πραγματική εφαρμογή θα απαιτούνταν moving average του s/token, διατήρηση ξεχωριστού ιστορικού s/token ανά ακρίβεια ή ελάχιστος αριθμός runs παραμονής σε κάθε ακρίβεια.

- Το Llama3.2 (1.2B) εκτελείται σε ~1.21 s/token, τιμή πάνω από το slow threshold. Αλλάζει σε q8 στο run 2 και παραμένει εκεί για τα υπόλοιπα 4 runs (q8: ~0.652–0.665 s/token, μόλις πάνω από το fast threshold 0.65).

Αποτέλεσμα: 1 αλλαγή, 35.7% μείωση ενέργειας ανά token έναντι fp16.

- Το Qwen2 (0.5B παράμετροι) εκτελείται σε ~0.50 s/token, τιμή κάτω από το slow threshold (0.80), οπότε η μετάβαση σε Q8_0 δεν ενεργοποιείται ποτέ.

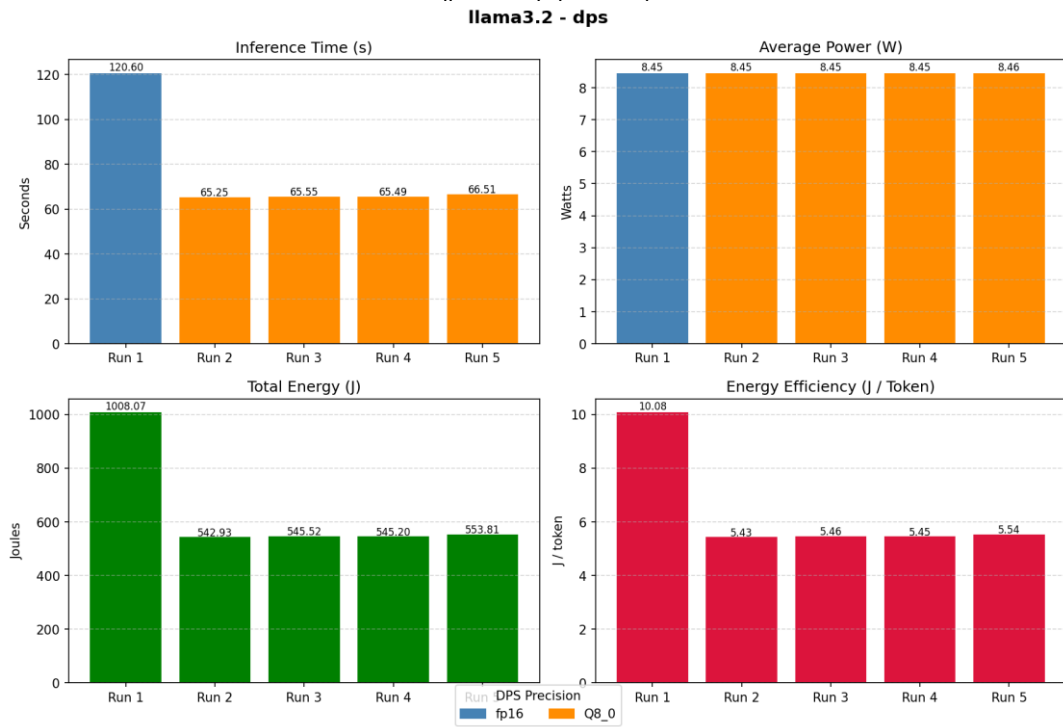
Αποτέλεσμα: καμία αλλαγή, πλήρης ακρίβεια fp16 σε όλα τα runs και ενέργεια ανά token ίση με το fp16 (+0.2%, εντός της διακύμανσης των μετρήσεων). Αυτό δείχνει ότι, όταν το fp16 είναι αρκετά γρήγορο, η DPS διατηρεί πλήρη ακρίβεια χωρίς πρόσθετο ενεργειακό κόστος.



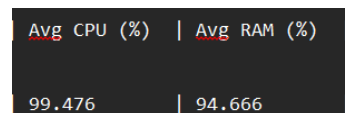
Εικόνα 26: tinylama dps Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.

Avg CPU (%)	Avg RAM (%)
99.778	92.238

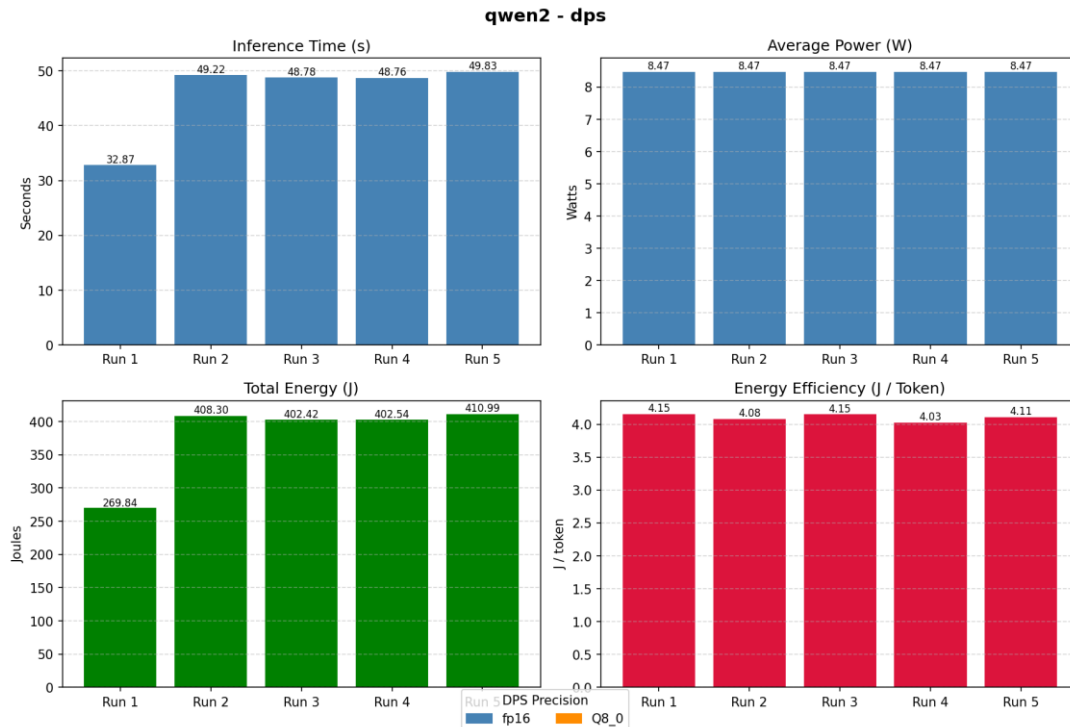
Εικόνα 27: Μέση Χρήση CPU & RAM tinylama dps.



Εικόνα 28: llama3.2 dps Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.



Εικόνα 29: Μέση Χρήση CPU & RAM llama3.2 dps.

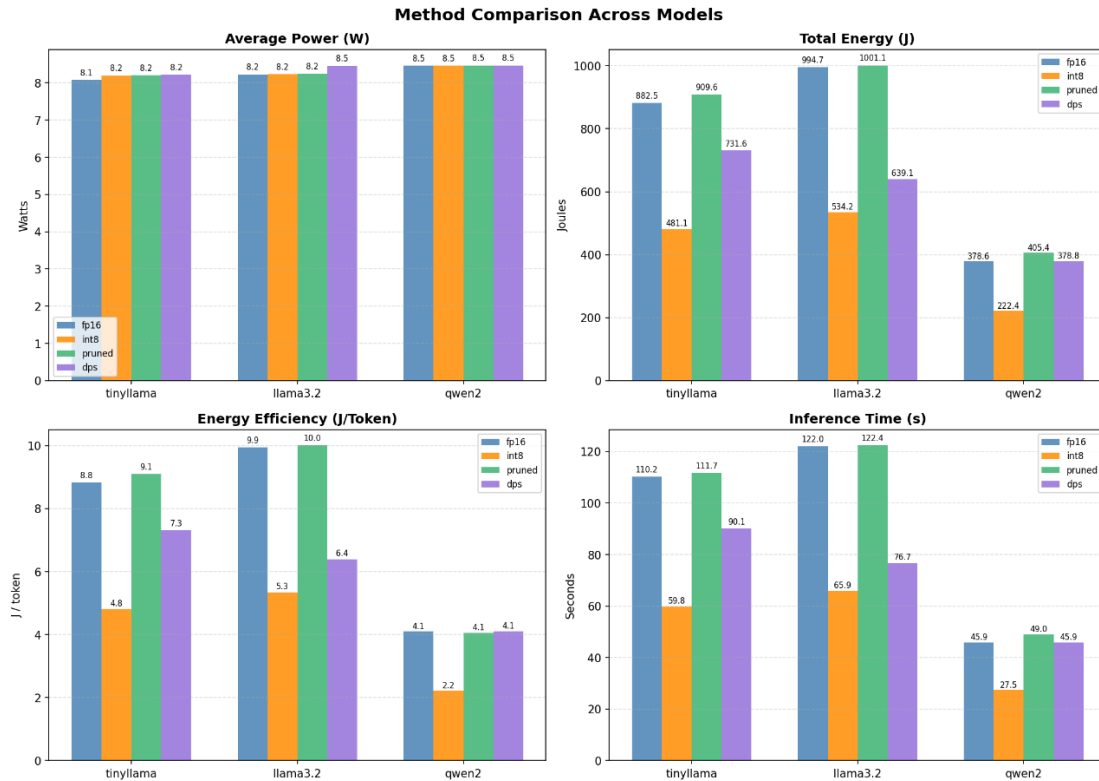


Εικόνα 30: qwen2 dps Χρόνος Εκτέλεσης, Μέση Ισχύς, Συνολική Ενέργεια & Ενεργειακή Απόδοση.

Avg CPU (%)	Avg RAM (%)
99.144	37.512

Εικόνα 31: Μέση Χρήση CPU & RAM qwen2 dps.

5.4 Συγκριτική Ανάλυση Μεθόδων



Εικόνα 32: Σύγκριση Μεθόδων Σε Όλα Τα Μοντέλα

Ο Πίνακας 6 συνοψίζει τη σχετική μεταβολή ενέργειας (J/token) κάθε μεθόδου σε σχέση με το fp16 baseline. Αρνητική τιμή σημαίνει βελτίωση (λιγότερη ενέργεια), θετική επιδείνωση.

Πίνακας 6: Σχετική μεταβολή J/token έναντι fp16 baseline.

Μοντέλο	int8 vs fp16	pruned vs fp16	dps vs fp16	dps vs int8	Καλύτερη
TinyLlama	-45.5%	+3.1%	-17.1%	+52.1%	int8
Llama3.2	-46.3%	+0.7%	-35.7%	+19.6%	int8
Qwen2	-45.7%	-1.0%	+0.2%	+84.5%	int8

Από τη συγκριτική ανάλυση προκύπτουν τα εξής:

- INT8 Κβαντισμός: Παρέχει την πιο σταθερή και μεγαλύτερη μείωση ενέργειας (~45%) σε όλα τα μοντέλα, χωρίς εμφανή υποβάθμιση ποιότητας στα TinyLlama και Llama 3.2. Αποτελεί την προτεινόμενη μέθοδο για CPU inference σε περιορισμένο hardware.

- **Unstructured Pruning:** Δεν αποφέρει ενεργειακό όφελος στο συγκεκριμένο setup. Για να αξιοποιηθεί η αραίωση ενεργειακά, απαιτείται structured pruning (αφαίρεση κεφαλών attention ή layers) ή hardware με native sparse execution support.
- **DPS:** Αποτελεί runtime-adaptive στρατηγική με αρκετό ενδιαφέρον. Στο Pi 400 δεν ξεπερνά το int8 ενεργειακά (επειδή εκτελεί μέρος των runs σε fp16, ενώ ο χρόνος φόρτωσης του μοντέλου δεν περιλαμβάνεται στις μετρήσεις), αλλά η αξία του βρίσκεται στην προσαρμοστικότητα: σε ταχύτερο hardware θα διατηρούσε fp16 για μικρά μοντέλα και θα μεταπηδούσε σε int8 μόνο για μεγαλύτερα.

5.5 Επίδραση Μεγέθους Μοντέλου

Σημαντικό εύρημα του πειράματος είναι ότι το μέγεθος του μοντέλου επηρεάζει περισσότερο την ενεργειακή κατανάλωση από την επιλογή μεθόδου. Το Qwen2 fp16 (4.096 J/token) είναι πιο αποδοτικό από το TinyLlama int8 (4.811 J/token) και το Llama3.2 int8 (5.342 J/token), δηλαδή ένα μοντέλο σε πλήρη ακρίβεια χωρίς καμία βελτιστοποίηση νικά ένα άλλο κβαντισμένο μοντέλο. Αυτό δείχνει ότι η επιλογή του σωστού μοντέλου εξαρχής μπορεί να είναι πιο σημαντική από οποιαδήποτε μέθοδο βελτιστοποίησης.

Αυτό υποδηλώνει ότι για εφαρμογές σε embedded περιβάλλον όπου ο κύριος στόχος είναι η ενεργειακή αποδοτικότητα, η επιλογή μικρότερου μοντέλου είναι πιο αποδοτική από οποιαδήποτε μέθοδο βελτιστοποίησης. Η βέλτιστη επιλογή παραμένει ο συνδυασμός: μικρό μοντέλο + int8 κβαντισμός.

5.6 Τελικά Συμπεράσματα

Τα πειραματικά αποτελέσματα οδηγούν στα εξής συμπεράσματα:

1. Η ισχύς του Raspberry Pi 400 παραμένει σχεδόν σταθερή (8.1 έως 8.5 W) ανεξαρτήτως μεθόδου inference. Ο κύριος παράγοντας που καθορίζει την κατανάλωση ενέργειας είναι ο χρόνος εκτέλεσης.
2. Ο INT8 κβαντισμός αποδίδει σταθερά ~45% εξοικονόμηση ενέργειας σε όλα τα μοντέλα, χωρίς εμφανές κόστος ποιότητας στα δύο μεγαλύτερα μοντέλα. Αποτελεί τη συνιστώμενη μέθοδο για ενεργειακά αποδοτικό inference σε CPU.
3. Το unstructured pruning δεν ωφελεί ενεργειακά σε CPU inference με llama.cpp. Απαιτείται structured pruning ή εξειδικευμένο sparse hardware για να αξιοποιηθεί η αραίωση.
4. Το DPS αποδεικνύει τη λειτουργικότητα του adaptive μηχανισμού: διαφορετικά μοντέλα εκδηλώνουν διαφορετική συμπεριφορά (0, 1 ή 4 αλλαγές) ανάλογα με τη σχέση απόδοσής τους με τα καθορισμένα όρια.
5. Το μέγεθος μοντέλου έχει μεγαλύτερη επίδραση στην ενέργεια από οποιαδήποτε μέθοδο βελτιστοποίησης. Ο βέλτιστος συνδυασμός για ενσωματωμένα συστήματα που ενδιαφέρονται μόνο για ενεργειακή απόδοση είναι: επιλογή μικρότερου μοντέλου + INT8.

6ο Κεφάλαιο: Συμπεράσματα και Μελλοντικές Προτάσεις

Στο παρόν κεφάλαιο συνοψίζονται τα συμπεράσματα της πτυχιακής εργασίας, αναλύονται τα προβλήματα που αντιμετωπίστηκαν κατά την υλοποίηση και οι τρόποι επίλυσής τους, αξιολογείται κριτικά η συνολική προσέγγιση, και παρουσιάζονται προτάσεις για μελλοντική έρευνα και επέκταση της εργασίας.

6.1 Συνολικά Συμπεράσματα της Πτυχιακής

Η παρούσα πτυχιακή εργασία είχε ως στόχο τη μέτρηση και σύγκριση της ενεργειακής κατανάλωσης τεσσάρων μεθόδων inference για γλωσσικά μοντέλα (LLMs) σε περιβάλλον περιορισμένων πόρων, συγκεκριμένα στο Raspberry Pi 400. Οι μέθοδοι που εξετάστηκαν ήταν η εκτέλεση πλήρους ακρίβειας (fp16), ο κβαντισμός INT8 (Q8_0/INT8), η αραιώση βαρών (unstructured pruning 30%), και ο δυναμικός έλεγχος ακρίβειας (Dynamic Precision Scaling/DPS).

Το κεντρικό εύρημα της εργασίας είναι ότι η κατανάλωση ισχύος παραμένει σχεδόν σταθερή (8.1 έως 8.5 W) ανεξαρτήτως μεθόδου, με αποτέλεσμα η ενεργειακή αποδοτικότητα να εξαρτάται κυρίως από τον χρόνο inference. Ο INT8 κβαντισμός επέτυχε μείωση ~45% στην κατανάλωση ενέργειας ανά token σε όλα τα μοντέλα, χωρίς αισθητή υποβάθμιση ποιότητας αποκρίσεων στα TinyLlama και Llama 3.2. Το unstructured pruning δεν απέδωσε κανένα ενεργειακό όφελος λόγω αρχιτεκτονικών περιορισμών της CPU inference. Η DPS επέδειξε τη λειτουργία ενός adaptive μηχανισμού που αποκρίνεται δυναμικά στις συνθήκες του hardware.

Εξίσου σημαντικό εύρημα αποτελεί το γεγονός ότι η επιλογή μοντέλου επηρεάζει την ενεργειακή κατανάλωση περισσότερο από οποιαδήποτε μέθοδο βελτιστοποίησης: το Qwen2 fp16 (4.096 J/token) αποδείχθηκε πιο αποδοτικό από το TinyLlama int8 (4.811 J/token), αναδεικνύοντας ότι ο συνδυασμός μικρού μοντέλου με INT8 αποτελεί τη βέλτιστη στρατηγική για ενεργειακά αποδοτικά embedded περιβάλλοντα.

6.2 Προβλήματα που Αντιμετωπίστηκαν

Κατά την υλοποίηση της πτυχιακής εργασίας αντιμετωπίστηκαν αρκετά τεχνικά προβλήματα που απαίτησαν επανασχεδιασμό τμημάτων της αρχιτεκτονικής. Τα σημαντικότερα παρουσιάζονται παρακάτω.

6.2.1 Ανακατασκευή Λογικής DPS

Η αρχική υλοποίηση της μεθόδου DPS βασιζόταν σε κατώφλια ισχύος (power thresholds): αν η μετρούμενη ισχύς ξεπερνούσε τα 6.0 W, το σύστημα μετέβαινε από fp16 σε Q8_0, ενώ κάτω από τα 5.0 W επέστρεφε σε fp16 (με δευτερεύοντα κριτήρια τη χρήση CPU και RAM). Το πρόβλημα εντοπίστηκε μόλις ξεκίνησαν οι μετρήσεις: το Raspberry Pi 400 καταναλώνει σταθερά ~7.5 W σε αδράνεια, οπότε το threshold ξεπερνιόταν πάντα από το πρώτο run, δηλαδή η DPS δεν ήταν ουσιαστικά διαφορετική από απλή INT8.

Η λύση ήταν να επανασχεδιαστεί η λογική switching βάσει χρόνου inference (seconds-per-token). Αφού η ισχύς είναι σταθερή, το s/token αποτελεί άμεσο proxy της ενεργειακής αποδοτικότητας.

Ορίστηκαν δύο κατώφλια με hysteresis (slow: 0.80 s/token, fast: 0.65 s/token) βαθμονομημένα από τις πραγματικές μετρήσεις του Pi 400. Αυτή η αλλαγή κατέστησε τη λογική της DPS τεκμηριωμένη και μετρήσιμη.

6.2.2 Αποκωδικοποίηση Πρωτοκόλλου UD18

Ο αισθητήρας Atorch UD18 δεν διαθέτει επίσημη τεκμηρίωση του δυαδικού πρωτοκόλλου επικοινωνίας. Τα δεδομένα αποστέλλονται ως raw bytes μέσω Bluetooth RFCOMM και η αποκωδικοποίηση απαιτεί γνώση των ακριβών byte offsets για κάθε μέτρηση.

Το πρόβλημα επιλύθηκε με συνδυασμό reverse engineering και δοκιμαστικών μετρήσεων: συγκρίθηκαν οι τιμές που εμφάνιζε η οθόνη του UD18 με τις τιμές που παρήγαγε ο κώδικας parsing για διάφορα γνωστά φορτία. Με αυτόν τον τρόπο επαληθεύτηκαν τα offsets τάσης (bytes 4–6, uint24/100), ρεύματος (bytes 7–9, uint24/100) και ισχύος (bytes 22–25, uint32/1000). Το byte checksum δεν επαληθεύεται στην τελική υλοποίηση. Η διαδικασία αυτή, αν και χρονοβόρα, οδήγησε σε αξιόπιστη και επαληθεύσιμη υλοποίηση.

6.2.3 Διαχείριση Μνήμης στο Raspberry Pi

Το Raspberry Pi 400 διαθέτει 4 GB RAM, ποσότητα οριακή για τη φόρτωση και εκτέλεση LLM μοντέλων σε fp16. Κατά τις πρώτες δοκιμές, η εκτέλεση διαδοχικών μεθόδων στην ίδια διεργασία Python οδηγούσε σε σταδιακή εξάντληση της μνήμης, με αποτέλεσμα αυξανόμενους χρόνους inference λόγω swap και μη αναπαραγωγίμα αποτελέσματα.

Η λύση ήταν διττή: πρώτον, κάθε συνδυασμός μοντέλου/μεθόδου εκτελείται ως ανεξάρτητη διεργασία subprocess, εξασφαλίζοντας πλήρη αποδέσμευση μνήμης μεταξύ μεθόδων. Δεύτερον, ο `experiment_runner.py` εγγράφει '3' στο `/proc/sys/vm/drop_caches` πριν και μετά από κάθε subprocess, καθαρίζοντας το page cache του λειτουργικού συστήματος. Επίσης, το swap αυξήθηκε σε 2 GB για να αντιμετωπιστούν περιπτώσεις αιχμής. Η προσέγγιση αυτή βελτίωσε σημαντικά την αναπαραγωγιμότητα των αποτελεσμάτων μεταξύ runs.

6.3 Κριτική Αξιολόγηση

Πέρα από τα πειραματικά αποτελέσματα, είναι σημαντικό να αξιολογηθεί κριτικά τόσο τα δυνατά σημεία της υλοποίησης όσο και οι περιορισμοί που επηρεάζουν τη γενίκευση των ευρημάτων.

6.3.1 Δυνατά Σημεία της Υλοποίησης

- **Αναπαραγωγιμότητα.** Ο συνδυασμός subprocess isolation, drop_caches και σταθερής temperature (0.1) εξασφαλίζει υψηλή αναπαραγωγιμότητα. Ενδεικτικά, η μέθοδος DPS στο Qwen2, που δεν άλλαξε ποτέ ακρίβεια, αποτέλεσε στην πράξη ανεξάρτητη επανάληψη του fp16 και έδωσε σχεδόν ίδια ενέργεια (4.104 έναντι 4.096 J/token, απόκλιση 0.2%) και ίδιες αποκρίσεις.
- **Modular αρχιτεκτονική.** Ο διαχωρισμός σε Inference, Monitoring, Evaluation και Experiment modules επιτρέπει εύκολη επέκταση: νέα μοντέλα μπορούν να προστεθούν κυρίως μέσω του config.yaml.
- **Hardware-level μέτρηση.** Η χρήση εξωτερικού αισθητήρα (Atorch UD18) αντί για software-based εκτίμηση παρέχει πραγματικές μετρήσεις ισχύος ανεξάρτητες από το λογισμικό που μετράται, περιορίζοντας έτσι το φαινόμενο του παρατηρητή (observer effect).

- Τεκμηριωμένη DPS λογική. Η time-based DPS με hysteresis είναι θεωρητικά τεκμηριωμένη και ακαδημαϊκά υποστηρίξιμη: η απόφαση switching βασίζεται σε μετρημένα δεδομένα, όχι σε εκτιμήσεις.

6.3.2 Περιορισμοί και Αδυναμίες

- Ένα μόνο περιβάλλον υλικού. Όλα τα πειράματα εκτελέστηκαν αποκλειστικά στο Raspberry Pi 400. Τα αποτελέσματα, ιδιαίτερα για τη DPS, δεν είναι άμεσα γενικεύσιμα σε άλλα hardware. Σε ταχύτερους επεξεργαστές τα thresholds θα έπρεπε να επαναβαθμονομηθούν.
- Μικρός αριθμός μοντέλων. Εξετάστηκαν μόνο τρία μοντέλα στην κατηγορία 0.5 έως 1.2B παραμέτρων. Η επέκταση σε μεγαλύτερα μοντέλα (3B, 7B) ή διαφορετικές αρχιτεκτονικές (Mistral, Gemma) θα ενίσχυε τη γενικευσιμότητα των συμπερασμάτων.
- Unstructured pruning χωρίς retraining. Το 30% L1 unstructured pruning εφαρμόστηκε χωρίς επανεκπαίδευση (fine-tuning). Στη βιβλιογραφία, το pruning χωρίς retraining σπάνια αποδίδει ενεργειακό όφελος σε CPU inference, επομένως το αρνητικό αποτέλεσμα ήταν αναμενόμενο και επιβεβαιώνει τη θεωρία, αλλά δεν αντιπροσωπεύει την καλύτερη δυνατή υλοποίηση pruning.
- Σταθερό workload. Κάθε run παράγει έως 100 tokens (`max_new_tokens = 100`). Τα περισσότερα runs φτάνουν το όριο, με εξαίρεση ορισμένα runs του Qwen2 (fp16 και dps, μέσος όρος 92.4 tokens) που τερμάτισαν νωρίτερα. Σε πραγματικές εφαρμογές το μήκος αποκρίσεων ποικίλλει σημαντικά, γεγονός που επηρεάζει τη συμπεριφορά της DPS και τις μετρήσεις J/token.
- Περιορισμοί στατιστικής και αξιολόγησης. Παρουσιάζονται μέσοι όροι 5 runs χωρίς τυπική απόκλιση, η σειρά εκτέλεσης δεν τυχαιοποιήθηκε (η μέση ισχύς αυξάνεται σταδιακά από 8.08 σε 8.47 W με τη σειρά εκτέλεσης) και η ποιότητα των αποκρίσεων αξιολογήθηκε μόνο ποιοτικά. Η αναφορά τυπικής απόκλισης, η τυχαιοποίηση της σειράς εκτέλεσης και η χρήση μετρικών όπως η perplexity θα ενίσχυαν τα συμπεράσματα.

6.4 Μελλοντικές Προτάσεις και Επεκτάσεις

Τα ευρήματα της παρούσας εργασίας ανοίγουν πολλαπλές κατευθύνσεις για μελλοντική έρευνα. Παρακάτω παρουσιάζονται οι πιο σημαντικές προτάσεις επέκτασης.

6.4.1 Εφαρμογή σε Μεγαλύτερα LLM

Η παρούσα εργασία εστιάστηκε σε μοντέλα της κατηγορίας 0.5B έως 1.2B παραμέτρων, που αποτελούν την κατηγορία που μπορεί να εκτελεστεί στο Raspberry Pi 400. Μελλοντική εργασία θα μπορούσε να επεκτείνει τη μεθοδολογία σε μοντέλα 3B–7B παραμέτρων, χρησιμοποιώντας ισχυρότερο hardware (Raspberry Pi 5, NVIDIA Jetson Nano, Apple Silicon)[5].

Ιδιαίτερο ενδιαφέρον παρουσιάζει η εξέταση του κατά πόσον η σταθερή ισχύς που παρατηρήθηκε στο Pi 400 αποτελεί χαρακτηριστικό αυτής της κατηγορίας hardware ή αν σε άλλες πλατφόρμες η ισχύς μεταβάλλεται σημαντικά μεταξύ μεθόδων, γεγονός που θα άλλαζε ριζικά τα συμπεράσματα για τη βέλτιστη μέθοδο.

6.4.2 DPS σε Ταχύτερο Hardware

Η DPS επέδειξε τη λειτουργία του adaptive μηχανισμού στο Pi 400, αλλά το hardware αυτό είναι οριακά αργό για fp16 inference σε μοντέλα 1B+. Αποτέλεσμα: για τα μοντέλα του ~1B η DPS μεταβαίνει σε Q8_0 ή ταλαντώνεται, περιορίζοντας την ακαδημαϊκή αξία της προσαρμοστικότητας.

Σε ταχύτερο hardware (π.χ. Raspberry Pi 5 με 8 GB RAM ή Apple M-series), το fp16 baseline για μοντέλα όπως τα TinyLlama και Llama 3.2 θα βρισκόταν κάτω από το slow threshold και το DPS θα διατηρούσε πλήρη ακρίβεια για μικρά-μεσαία μοντέλα και θα μετέβαινε σε Q8_0 μόνο για μεγαλύτερα. Αυτή η συμπεριφορά θα αποδείκνυε πλήρως τον ισχυρισμό ότι το DPS λειτουργεί ως στρατηγική που προσαρμόζεται στο hardware χωρίς manual configuration.

6.4.3 Έρευνα για Thermally-Aware DPS AI Inference

Κατά τη διάρκεια του πειράματος παρατηρήθηκε ότι ο χρόνος inference αυξανόταν σταδιακά μετά από πολλά διαδοχικά runs και ότι η μέση ισχύς αυξανόταν με τη σειρά εκτέλεσης (από 8.08 σε 8.47 W). Μια πιθανή εξήγηση είναι η αύξηση της θερμοκρασίας του επεξεργαστή και ενδεχομένως thermal throttling, καθώς το Raspberry Pi 400 δεν διαθέτει ενεργή ψύξη. Η θερμοκρασία όμως δεν καταγράφηκε στο παρόν πείραμα, οπότε η υπόθεση αυτή χρειάζεται επιβεβαίωση.

Μια μελλοντική εργασία θα μπορούσε να εξετάσει την ανάπτυξη Thermally-Aware inference [22]: ένα σύστημα που παρακολουθεί τη θερμοκρασία της CPU σε πραγματικό χρόνο (μέσω /sys/class/thermal) και χρησιμοποιεί αυτήν ως επιπλέον σήμα για τη λήψη αποφάσεων precision switching, συνδυάζοντας δηλαδή τη DPS λογική με thermal management.

Αναφορές

- [1] A. d’Orgeval, S. Sheehan, Q. Avenas, E. Assoumou, and V. Sessa, ‘Generative AI impact assessment through a life cycle analysis of multiple data center typologies’, *Appl. Energy*, vol. 406, p. 127288, Mar. 2026, doi: 10.1016/j.apenergy.2025.127288.
- [2] A. Alaktif, M. Chergui, I. Daoudi, and A. Ammoumou, ‘All You Should Know About Large Language Models (LLMs)’, in *2024 7th International Conference on Advanced Communication Technologies and Networking (CommNet)*, Dec. 2024, pp. 1–10. doi: 10.1109/CommNet63022.2024.10793270.
- [3] X. Zhang *et al.*, ‘Edge Intelligence in the Generative Artificial Intelligence Era’, *IEEE Wirel. Commun.*, vol. 32, no. 6, pp. 60–68, Dec. 2025, doi: 10.1109/MWC.2025.3599652.
- [4] A. Dequino, L. Bompani, L. Benini, and F. Conti, ‘Optimizing BFloat16 Deployment of Tiny Transformers on Ultra-Low Power Extreme Edge SoCs’, *J. Low Power Electron. Appl.*, vol. 15, no. 1, p. 8, Mar. 2025, doi: 10.3390/jlpea15010008.
- [5] M. R. Islam *et al.*, ‘Characterizing and Understanding Energy Footprint and Efficiency of Small Language Model on Edges’, in *2025 IEEE 22nd International Conference on Mobile Ad-Hoc and Smart Systems (MASS)*, Oct. 2025, pp. 363–371. doi: 10.1109/MASS66014.2025.00059.
- [6] H. Martínez, S. Catalán, A. Castelló, and E. S. Quintana-Ortí, ‘Characterization of quantized inference with transformer encoders on low power CPUs’, *Int. J. High Perform. Comput. Appl.*, vol. 39, no. 6, pp. 803–821, Nov. 2025, doi: 10.1177/10943420251355115.
- [7] S. O. Semerikov, T. A. Vakaliuk, O. B. Kanevska, O. A. Ostroushko, and A. O. Kolhatin, ‘Edge intelligence unleashed: a survey on deploying large language models in resource-constrained environments’, *J. Edge Comput.*, vol. 4, no. 2, pp. 179–233, Nov. 2025, doi: 10.55056/jec.1000.
- [8] S. S. Girija, L. Arora, S. Kapoor, D. Pradhan, A. Raj, and A. Shetgaonkar, ‘Optimizing LLMs for Resource-Constrained Environments: A Survey of Model Compression Techniques’, in *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, Jul. 2025, pp. 1657–1664. doi: 10.1109/COMPSAC65507.2025.00224.
- [9] C. Tian *et al.*, ‘A survey on deep learning fundamentals’, *Artif. Intell. Rev.*, vol. 58, no. 12, p. 381, Oct. 2025, doi: 10.1007/s10462-025-11368-7.
- [10] A. Vaswani *et al.*, ‘Attention Is All You Need’, Aug. 02, 2023, *arXiv*: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.
- [11] S. Sun *et al.*, ‘Optimizing Attention for Efficient LLM Inference: A Review’, in *2025 8th World Conference on Computing and Communication Technologies (WCCCT)*, Apr. 2025, pp. 482–491. doi: 10.1109/WCCCT65447.2025.11027973.
- [12] I. Mangkunegara, I. Purwono, A. Ma’arif, D.-N. Basil, H. M. Marhoon, and A.-N. Sharkawy, ‘Transformer Models in Deep Learning: Foundations, Advances, Challenges and Future Directions’, *Bul. Ilm. Sarj. Tek. Elektro*, vol. 7, pp. 231–241, Jun. 2025, doi: 10.12928/biste.v7i2.13053.
- [13] P. V. Dantas, L. C. Cordeiro, and W. S. S. Junior, ‘A review of state-of-the-art techniques for large language model compression’, *Complex Intell. Syst.*, vol. 11, no. 9, p. 407, Aug. 2025, doi: 10.1007/s40747-025-02019-z.
- [14] J. Lang, Z. Guo, and S. Huang, ‘A Comprehensive Study on Quantization Techniques for Large Language Models’, in *2024 4th International Conference on Artificial Intelligence*,

Robotics, and Communication (ICAIRC), Dec. 2024, pp. 224–231. doi: 10.1109/ICAIRC64177.2024.10899941.

[15] E. J. Husom *et al.*, ‘Sustainable LLM Inference for Edge AI: Evaluating Quantized LLMs for Energy Efficiency, Output Accuracy, and Inference Latency’, *ACM Trans Internet Things*, vol. 6, no. 4, p. 28:1-28:35, Nov. 2025, doi: 10.1145/3767742.

[16] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, ‘Understanding the impact of precision quantization on the accuracy and energy of neural networks’, in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, Mar. 2017, pp. 1474–1479. doi: 10.23919/DATE.2017.7927224.

[17] 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.

[18] C. Tian, X. Qin, and L. Li, ‘GreenLLM: Towards Efficient Large Language Model via Energy-aware Pruning’, in *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*, Jun. 2024, pp. 1–2. doi: 10.1109/IWQoS61813.2024.10682928.

[19] L. Liu *et al.*, ‘Drift: Leveraging Distribution-based Dynamic Precision Quantization for Efficient Deep Neural Network Acceleration’, in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, in DAC ’24. New York, NY, USA: Association for Computing Machinery, Nov. 2024, pp. 1–6. doi: 10.1145/3649329.3655986.

[20] M. R. P C, A. M R, V. P. Gopi, G. Lakshminarayanan, G. R. Gangadharan, and J. U. Kidav, ‘Dynamic Precision Scaling in MAC Units for Energy-Efficient Computations in Deep Neural Network Accelerators’, in *2024 28th International Symposium on VLSI Design and Test (VDATE)*, Sep. 2024, pp. 1–7. doi: 10.1109/VDATE63601.2024.10705697.

[21] J. Stojkovic, C. Zhang, Í. Goiri, J. Torrellas, and E. Choukse, ‘DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency’, in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Mar. 2025. doi: 10.1109/HPCA61900.2025.00102.

[22] J. Stojkovic *et al.*, ‘TAPAS: Thermal- and Power-Aware Scheduling for LLM Inference in Cloud Platforms’, in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’25)*, Mar. 2025. doi: 10.1145/3676641.3716025.

[23] Z. Huang *et al.*, ‘MNN-AECS: Energy Optimization for LLM Decoding on Mobile Devices via Adaptive Core Selection’, Jun. 2025, *arXiv*: arXiv:2506.19884. doi: 10.48550/arXiv.2506.19884.

[24] Q. Liu, D. Huang, M. Zapater, and D. Atienza, ‘GreenLLM: SLO-Aware Dynamic Frequency Scaling for Energy-Efficient LLM Serving’, Aug. 2025, *arXiv*: arXiv:2508.16449. doi: 10.48550/arXiv.2508.16449.

Παράρτημα 1. Κώδικας Που Αναπτύχθηκε

Ο κώδικας που αναπτύχθηκε κατά την εκπόνηση αυτής της πτυχιακής εργασίας μπορεί να βρεθεί με πλήρη επεξήγηση στο παρακάτω repository στο GitHub:

https://github.com/TsipiDev/Thesis_Code_Dimitris_Vatousis_220007