Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

An Application Analysis Tool for Cross-Device Energy Consumption Estimation

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΝΙΚΟΛΑΟΥ ΜΑΪΔΩΝΗ

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων
Αθήνα, Φεβρουάριος 2022

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

An Application Analysis Tool for Cross-Device Energy Consumption Estimation

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΝΙΚΟΛΑΟΥ ΜΑΪΔΩΝΗ

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 18η Φεβρουαρίου 2022.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δημήτριος Σούντρης	Παναγιώτης Τσανάκας	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π

Αθήνα, Φεβρουάριος 2022

(Υπογραφή)

.....
ΜΑΪΔΩΝΗΣ ΝΙΚΟΛΑΟΣ
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

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

Περίληψη

Η συνεχώς αυξανόμενη κατανάλωση ενεργειακών πόρων οδηγούν στην έρευνα και ανάπτυξη τεχνολογιών που είναι ενεργειακά βιώσιμες. Οι συσκευές χαμηλής ενεργειακής κατανάλωσης βρίσκονται στο επίκεντρο ενός μεγάλου τεχνολογικού φάσματος. Η ανάπτυξη του Διαδικτύου των Πραγμάτων (Internet of Things), των Ενσωματωμένων Συστημάτων (Embedded Systems), καθώς και της Μηχανικής Μάθησης (Machine Learning) μπορούν να βοηθήσουν στην σχεδίαση λογισμικού με έναν πιο αποδοτικό ενεργειακά τρόπο. Προς αυτή την κατεύθυνση, το πεδίο της εκτίμησης ενέργειας συνεχώς απασχολεί περισσότερους ερευνητές.

Σε αυτή την διπλωματική εργασία, στοχεύουμε στην πράσινη σχεδίαση λογισμικού και πιο ιδιαίτερα στην ενεργειακά βιώσιμη σχεδίαση κώδικα. Προτείνουμε έναν τρόπο χρήσης της Μηχανικής Μάθησης για να προβλέψουμε δυναμικά την κατανάλωση ενέργειας C προγραμμάτων μικρής και μεσαίας ενεργειακής τάξης. Για τη διαδικασία της εκπαίδευσης του μοντέλου δημιουργήθηκαν 700 κώδικες με το χαρακτηριστικό της πολυπλοκότητας βρόχου. Αυτοί οι 700 κωδικοί εκπαίδευσης περνούν από τέσσερις profilers , οι οποίοι συλλέγουν πληροφορίες σχετικά με τους κώδικες, με την πλειοψηφία αυτών να αποτελούν πληροφορίες διαχείρισης μνήμης. Μετά τη δημιουργία, λοιπόν, ενός training dataset που αποτελείται από 700 κώδικες και 141 χαρακτηριστικά για τον καθένα, η κατανάλωση ενέργειας αυτών των 700 προγραμμάτων μετράται στην πλατφόρμα ανάπτυξης του εργαστηρίου χρησιμοποιώντας έναν αισθητήρα ισχύος. Στη συνέχεια, διαχωρίζοντας αυτά τα δεδομένα σε 80% train και 20% test και πραγματοποιείται σύγκριση μεταξύ μερικών αλγορίθμων. Το αποτέλεσμα αυτής ανέδειξε τον αλγόριθμο Lasso ως τον καταλληλότερο για την εργασία αυτή. Έτσι, ένα βασισμένο στον Lasso μοντέλο regression περνά από μια διαδικασία βελτιστοποίησης παραμέτρων και στη συνέχεια εκπαιδεύεται από αυτά τα δεδομένα. Παράλληλα, υπάρχει μια ανάλυση της σημαντικότητας των χαρακτηριστικών των προγραμμάτων, συγκρίνοντας τους συντελεστές του αλγορίθμου Lasso με τα αποτελέσματα ενός τεστ συσχέτισης μεταξύ των χαρακτηριστικών. Στη συνέχεια, αξιολογούμε την ακρίβεια της πρόβλεψης δοκιμάζοντας 56 προγράμματα από το PolyBench Suite. Αυτά τα προγράμματα περνούν από τους profilers και ύστερα μετράται η ενέργεια που καταναλώνουν. Επομένως η αξιολόγηση βασίζεται στη σύγκριση μεταξύ των πραγματικών και των προβλεπόμενων ενεργειακών τιμών. Τα αποτελέσματα είναι πολύ ενθαρρυντικά, καθώς επιτυγχάνουμε R^2 score ακρίβειας 0,960387, η οποία είναι συγκρίσιμη με παρόμοιες προσεγγίσεις εκτίμησης ενέργειας.

Λέξεις Κλειδιά

Κατανάλωση ενέργειας, Βιωσιμότητα, Εκτίμηση Ενέργειας, Χρήση Μνήμης, Ενσωματωμένα Συστήματα, Μηχανική Μάθηση, Εργαλεία Profiling, Μοντέλο Regression, Πολυπλοκότητα Βρόχου, Ανάλυση Χαρακτηριστικών, R^2, Εργαλείο

Abstract

The ever-increasing consumption of energy resources has led to the search for technologies that are energy efficient. Low energy consumption devices have become state-of-the-art for most of the technological aspects. As it follows, Internet of Things (IoT) devices Embedded Systems and Machine Learning (ML) can help in designing software in a more energy efficient way. In this direction, the field of energy estimation constantly employs more researchers.

In this thesis, we target the Green Software Engineering and especially the energy sustainable program development. We propose a way to use ML in order to dynamically predict the energy consumption of low and medium energy, C language programs. For the training process 700 codes with the characteristic of loop complexity were generated. Those 700 training codes are being profiled by four profilers that collect information about the code, with the majority being memory management information. After creating a training dataset of 700 codes and 141 features, the energy consumption of these 700 codes is being measured on the lab's development platform using a power sensor. Then, by splitting this data to a 80% train and 20% test, a comparison between a few algorithms indicated that the Lasso algorithm was suitable for this project. So, a Lasso-based regression model is going through a parameter optimization process, and consequently gets trained by these data. Alongside, there is a feature importance analysis by comparing the Lasso coefficients with the results of a feature correlation test. Then, we evaluate the prediction accuracy by testing 56 benchmarks from PolyBench Suite. These benchmarks are being profiled and energy measured, so the evaluation relies on the comparison between the real and the predicted energy values. The results are very encouraging, as we achieve a R^2 score of 0.960387, that is comparable with similar energy estimation approaches.

The project is designed to be a dynamic tool. The tool gets a low or medium energy, C language code by the user as an input. The user has to put two derivatives around the "heaviest" loop and then by running two python scripts the energy prediction comes as an output.

Keywords

Energy Consumption, Sustainability, Energy Estimation, Memory Usage, Embedded Systems, Machine Learning, Profiling Tools, Regression Model, Loop Complexity, Feature Analysis, R^2, Tool

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κύριο Δημήτριο Σούντρη που μου έδωσε τη δυνατότητα και την ευκαιρία να εκπονήσω ένα θέμα διπλωματικής άκρως ενδιαφέρον για εμένα.

Επίσης θέλω να ευχαριστήσω θερμά τον υποψήφιο διδάκτορα Χαράλαμπο Μαράντο που εξ αρχής μου έδωσε κίνητρο να ασχοληθώ με τομείς που δεν είχα ασχοληθεί μέχρι τότε και να εξελιχθώ μέσα από αυτή την εργασία. Όποτε αντιμετώπιζα κάποιο πρόβλημα ήταν εκεί και με βοηθούσε άμεσα και αποτελεσματικά, παρακάμπτοντας τις δυσκολίες επικοινωνίας λόγω των συνθηκών.

Φυσικά, θα ήθελα να ευχαριστήσω τον αδερφό μου και τους γονείς μου, που χωρίς τη στήριξη αυτών όλα αυτά τα χρόνια δεν θα βρισκόμουν σε αυτή τη θέση. Τέλος, θα ήθελα να ευχαριστήσω τους φίλους μου για το ενδιαφέρον που μου μετέδωσαν και την βοήθειά που μου χάρησαν όλα αυτά τα χρόνια των προπτυχιακών μου σπουδών.

Contents

1 Introduction **40**
- 1.1 Related work . . . 42
- 1.2 Thesis Outline . . . 44

2 Technical and Theoretical Background **45**
- 2.1 Energy measurements in embedded systems . . . 45
- 2.2 Profiling Tools . . . 47
- 2.3 Machine Learning . . . 51
  - 2.3.1 Supervised Learning . . . 51
  - 2.3.2 Regression . . . 52
  - 2.3.3 Classification . . . 53

3 Proposed Tool **56**
- 3.1 Overall Methodology . . . 56
- 3.2 C Code Generator . . . 57
- 3.3 Profiling . . . 58
- 3.4 Building Datasets . . . 59
- 3.5 Training model . . . 60
  - 3.5.1 Preparing the model . . . 60
  - 3.5.2 Lasso Regression Model . . . 61
  - 3.5.3 Grid Search and Cross-Validation . . . 62

4 Experimental Results **63**
- 4.1 Feature Analysis . . . 65
  - 4.1.1 Feature Importance . . . 65
  - 4.1.2 Feature Correlation . . . 66
- 4.2 Model Implementation . . . 68
- 4.3 Evaluation . . . 68
  - 4.3.1 Evaluation through Dataset . . . 70
  - 4.3.2 Evaluation through Polybench Suite . . . 72
- 4.4 Test the method on alternative device . . . 75
- 4.5 Comparison with alternative approaches . . . 76

5 Discussion Conclusions and Future Work **77**

A How to use the tool **78**

Εκτεταμένη Περίληψη

Εισαγωγή

Τις τελευταίες δεκαετίες έχει συντελεστεί εκθετική αύξηση στη χρήση των ηλεκτρονικών συστημάτων. Το όραμα ενός κόσμου όπου η τελευταία λέξη της τεχνολογίας είναι στα χέρια όλων έχει σχεδόν γίνει πραγματικότητα και η ευκολία χρήσης έχει φέρει ταχύτητα και άνεση στην εμπειρία του χρήστη. Πολλές συσκευές όπως υπολογιστές, κινητά τηλέφωνα, αυτοκίνητα, τηλεοράσεις, κ.λπ., είναι συνδεδεμένα στο διαδίκτυο, ώστε να μπορούν να μοιράζονται δεδομένα με άλλες συσκευές που ανήκουν στο Διαδικτύο των Πραγμάτων. Αυτές οι συνδεδεμένες στο διαδίκτυο συσκευές συλλέγουν δεδομένα που τους επιτρέπουν να εκπαιδεύονται, να μαθαίνουν, να προβλέπουν και να προσαρμόζονται στις ανάγκες του χρήστη.

Η Μηχανική Μάθηση είναι ένα υποσύνολο της Τεχνητής Νοημοσύνης που παρέχει στο σύστημα τη δυνατότητα της αυτόματης μάθησης και βελτίωσης από την εμπειρία και όχι από τον ρητό προγραμματισμό. Αυτό θεωρείται μια σημαντική τεχνολογική καινοτομία που μπορεί να διαχειριστεί έναν τεράστιο όγκο δεδομένων, να ερμηνεύσει μοτίβα και δομές που επιτρέπουν τη μάθηση, και να οδηγήσει τελικά στην λήψη αποφάσεων χωρίς την επέμβαση του ανθρώπου.

Την ίδια στιγμή, οι ενεργειακές απαιτήσεις παγκοσμίως έχουν φτάσει στο υψηλότερο σημείο και η ενεργειακή βιωσιμότητα έχει γίνει ένας από τους κύριους τομείς της επιστημονικής έρευνας. Με την άνθηση της Μηχανικής Μάθησης και του Διαδικτύου των Πραγμάτων, εφαρμογή τους στην ενεργειακή αλυσίδα δεν αποτελεί εξαίρεση, αντιθέτως πρέπει να αποτελεί κινητήρια δύναμη αλλαγών παγκόσμιας κλίμακας.

Όπως προκύπτει, είναι αναγκαίο για τις βιομηχανίες και τους προγραμματιστές να επικεντρωθούν στον Πράσινο Προγραμματισμό. Σε επίπεδο υλικού, υπάρχουν πολλές λύσεις για την κατασκευή και χρήση υλικού με ενεργειακά αποδοτικό τρόπο, κατασκευάζοντας Ενσωματωμένα Συστήματα που προσαρμόζονται στις ανάγκες της εκάστοτε εφαρμογής. Αλλά σε επίπεδο λογισμικού, η συζήτηση για την ενεργειακά βέλτιστη σχεδίασή του μόλις αρχίζει.

Ο σκοπός αυτής της εργασίας είναι η χρήση της Μηχανικής Μάθησης για την δυναμική πρόβλεψη της κατανάλωσης ενέργειας αρχείων κώδικα. Πιο συγκεκριμένα στην εργασία αυτή επικεντρωνόμαστε σε χαμηλών και μεσαίων ενεργειακών απαιτήσεων προγράμματα, γλώσσας C. Ο ορισμός των προγραμμάτων ως χαμηλών και μεσαίων ενέργειακών απαιτήσεων δίνεται παρακάτω:

•Χαμηλών Ενεργειακών Απαιτήσεων: Περίπου 128KB μνήμης. Το πρόβλημα δεν πρέπει να χρησιμοποιήσει μόνο την L1 Cache, αλλά πρέπει να χρησιμοποιήσει και την L2 Cache.

•Μεσαίων Ενεργειακών Απαιτήσεων: Περίπου 1MB μνήμης. Το πρόβλημα δεν πρέπει να χρησιμοποιήσει μόνο την L2 Cache, αλλά πρέπει να χρησιμοποιήσει και την L3 Cache.

Κατά την εκτέλεση ενός κώδικα σε ένα συγκεκριμένο μηχάνημα, είναι πολύ σημαντικό για τον προγραμματιστή να γνωρίζει το επερχόμενο ενεργειακό κόστος, ώστε αν χρειάζεται να μπορεί να σχεδιάσει τον κώδικα με έναν περισσότερο αποδοτικό τρόπο ή να το εκτελέσει σε διαφορετικό μηχάνημα. Σε μεγαλύτερη κλίμακα, η γνώση της κατανάλωσης ενέργειας των εφαρμογών μπορεί να βοηθήσει την βιομηχανία στην αποφυγή της σπατάλης ενεργειακών πόρων και κατ΄ επέκταση στην σχεδίαση των εφαρμογών με τεχνικές προς την κατεύθυνση της Ενεργειακής Βιωσιμότητας.

Τεχνικό και Θεωρητικό Υπόβαθρο

Κατά την διαδικασία των ενεργειακών μετρήσεων, χρησιμοποιήθηκαν πλακέτες και αισθητήρες ισχύος. Αυτά τα ενσωματωμένα συστήματα βρίσκονται στο Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών ΕΜΠ και παρατίθενται παρακάτω.

•Η πλακέτα **NVIDIA Jetson TX1** είναι ένα ενσωματωμένο system-on-module (SoM) με ενσωματωμένη GPU μικροαρχιτεκτονικής NVIDIA Maxwell, τετραπύρηνη CPU ARM Cortex-A57 και μνήμη 4 GB LPDDR4. Χρήσιμο για την ανάπτυξη της Όρασης Υπολογιστών και της Βαθιάς Μάθησης, η πλακέτα NVIDIA Jetson TX1 παρέχει υπολογιστική απόδοση 1TFLOPS FP16 σε ισχύ 10 Watt.

•Η πλακέτα **NVIDIA Jetson Xavier NX** είναι ένα ενσωματωμένο system-on-module (SoM) με ενσωματωμένη GPU που βασίζεται στην αρχιτεκτονική Nvidia Volta 384 CUDA και εξαπύρηνη CPU 64-bit NVIDIA Carmel ARM v8.2. Διαθέτει cache 2 επιπέδων (6 MB L2 + 4 MB L3) και 8 GB 128-bit LPDDR4x RAM. Χρησιμοποιείται συνήθως σε συστήματα Τεχνητής Νοημοσύνης υψηλής απόδοσης και επιταχύνει τη στοίβα λογισμικού NVIDIA σε ισχύ 10 watt.

•Ο αισθητήρας **INA3221** είναι μια οθόνη τριών καναλιών ρεύματος και τάσης διαύλου με διεπαφή συμβατή με I2C και SMBUS. Ο INA3221 παρακολουθεί τόσο τις πτώσεις τάσης διακλάδωσης όσο και τις τάσεις τροφοδοσίας διαύλου, εκτός από το ότι έχει προγραμματιζόμενους χρόνους μετατροπής και λειτουργίες μέσου όρου για αυτά τα σήματα, που επιτρέπουν τη μέτρηση ισχύος διαφόρων εφαρμογών.

Όπως αναφέρθηκε προηγουμένως, ο σκοπός αυτού του έργου είναι να προβλέψει την ενεργειακή κατανάλωση προγραμμάτων. Προκειμένου να επιτευχθεί αυτό μέσω Μηχανικής Μάθησης, επιλέξαμε να χρησιμοποιήσουμε ορισμένα εργαλεία για την εξαγωγή χρήσιμων πληροφοριών σχετικά με τους κώδικες, που αφορούν κυρίως την πρόσβαση στη μνήμη. Τα εργαλεία που έχουν χρησιμοποιηθεί είναι μερικά υποεργαλεία των **Valgrind** και **Intel Pin**.

Το Valgrind είναι ένα framework για την δημιουργία εργαλείων δυναμικής ανάλυσης. Υπάρχουν εργαλεία Valgrind που μπορούν να εντοπίσουν αυτόματα πολλά σφάλματα διαχείρισης μνήμης, καθώς και να χαρακτηρίσουν ένα πρόγραμμα

λεπτομερώς. Περιλαμβάνει εργαλεία ποιότητας παραγωγής και συλλέγει πληροφορίες σφαλμάτων μνήμης, σφάλματων νήματος, κρυφής μνήμης, πρόβλεψης διακλαδώσεων, κ.λπ.

Το Pin είναι ένα δυναμικό binary framework που επιτρέπει τη δημιουργία εργαλείων δυναμικής ανάλυσης προγραμμάτων. Ως δυναμικά εργαλεία, η ανάλυση γίνεται κατά το χρόνο εκτέλεσης στα μεταγλωττισμένα δυαδικά αρχεία. Το Pin παρέχει ένα πλούσιο API που επιτρέπει την συλλογή πληροφοριών, όπως τα περιεχόμενα των καταχωρητών να μεταβιβάζονται στον εισαγόμενο κώδικα ως παράμετροι. Αυτόματα αποθηκεύει και επαναφέρει τους καταχωρητές που έχουν αντικατασταθεί από τον κώδικα που έχει εισαχθεί, ώστε η εφαρμογή να συνεχίζει να λειτουργεί. Περιορισμένη πρόσβαση σε πληροφορίες συμβόλων και εντοπισμού σφαλμάτων είναι επίσης διαθέσιμη.

Παρακάτω παρουσιάζονται τα συγκεκριμένα υποεργαλεία των Valgrind και IntelPin που έχουν χρησιμοποιήθηκαν.

Cachegrind

Το Cachegrind προσομοιώνει τον τρόπο με τον οποίο το πρόγραμμα αλληλεπιδρά με την ιεραρχία της κρυφής μνήμης και (προαιρετικά) την πρόβλεψη διακλάδωσης μιας μηχανής. Προσομοιώνει μια μηχανή με ανεξάρτητες εντολές πρώτου επιπέδου και κρυφές μνήμες δεδομένων (I1 και D1), που υποστηρίζονται από μια ενοποιημένη κρυφή μνήμη δεύτερου επιπέδου (L2). Αυτό ταιριάζει ακριβώς με τη διαμόρφωση πολλών σύγχρονων μηχανών.

Ωστόσο, ορισμένα σύγχρονα μηχανήματα έχουν τρία ή τέσσερα επίπεδα κρυφής μνήμης. Για αυτά τα μηχανήματα, το Cachegrind προσομοιώνει τις κρυφές μνήμες πρώτου και τελευταίου επιπέδου. Ο λόγος για αυτήν την επιλογή είναι ότι η κρυφή μνήμη τελευταίου επιπέδου έχει τη μεγαλύτερη επιρροή στο χρόνο εκτέλεσης, καθώς καλύπτει τις προσβάσεις στην κύρια μνήμη. Επομένως, το Cachegrind αναφέρεται πάντα στις κρυφές μνήμες I1, D1 και LL (τελευταίο επίπεδο).

Η λεπτομερής δημιουργία προφίλ προσωρινής μνήμης και διακλάδωσης μπορεί να είναι πολύ χρήσιμη για την κατανόηση του τρόπου με τον οποίο το πρόγραμμά αλληλεπιδρά με το μηχάνημα. Επίσης, καθώς εκτελείται μια εντολή ανάγνωσης της προσωρινής μνήμης ανά εντολή που εκτελείται, ανακαλύπτει πόσες εντολές εκτελούνται ανά γραμμή, κάτι που μπορεί να είναι χρήσιμο για την παραδοσιακή ανάλυση κώδικα.

Massif

Το Massif είναι ένα εργαλείο ανάλυσης της μνήμης σωρού (heap memory), καθώς μετρά πόση μνήμη σωρού χρησιμοποιεί το πρόγραμμα. Αυτό περιλαμβάνει τόσο τον χρήσιμο χώρο όσο και τα επιπλέον byte που δεσμεύνονται. Μπορεί επίσης να μετρήσει το μέγεθος της στοίβας του προγράμματος. Η δημιουργία προφίλ σωρού μπορεί να βοηθήσει στη μείωση της ποσότητας της μνήμης που χρησιμοποιεί το πρόγραμμα.

Επίσης, υπάρχουν ορισμένες διαρροές χώρου που δεν εντοπίζονται από τους παραδοσιακούς ελεγκτές διαρροών. Αυτό συμβαίνει επειδή η μνήμη δεν χάνεται πο-

τέ (απομένει ένας δείκτης σε αυτήν) αλλά δεν χρησιμοποιείται. Προγράμματα που έχουν διαρροές όπως αυτή μπορεί να αυξήσουν άσκοπα την ποσότητα της μνήμης που χρησιμοποιούν με την πάροδο του χρόνου. Το Massif μπορεί να βοηθήσει στον εντοπισμό αυτών των διαρροών. Είναι σημαντικό ότι το Massif λέει όχι μόνο πόση μνήμη σωρού χρησιμοποιεί το πρόγραμμα, αλλά παρέχει επίσης πολύ λεπτομερείς πληροφορίες που υποδεικνύουν ποια μέρη του προγράμματος είναι υπεύθυνα για την εκχώρηση της μνήμης σωρού.

MICA

Το MICA (Microarchitecture-Independent Characterization of Applications) είναι ένα εργαλείο Pin, το οποίο επιτρέπει στο χρήστη να συλλέξει μια σειρά από χαρακτηριστικά προγράμματος για να ποσοτικοποιήσει τη συμπεριφορά του προγράμματος κατά τον χρόνο εκτέλεσης. Τα χαρακτηριστικά αυτά αφορούν την κατηγοριοποίηση των εντολών χαμηλού επιπέδου, το αποτύπωμα μνήμης, την πρόβλεψη διακλαδώσεων, κ.λπ. Αυτά τα χαρακτηριστικά του προγράμματος είναι εντελώς ανεξάρτητα από τη μικροαρχιτεκτονική στην οποία γίνονται οι μετρήσεις, σε αντίθεση με άλλες τεχνικές χαρακτηρισμού του προγράμματος που χρησιμοποιούν προσομοίωση ή μετρητές απόδοσης υλικού.

PinTool

Το PinTool είναι ένα εργαλείο Pin, το οποίο επιτρέπει στον χρήστη να συλλέξει πολλά χαρακτηριστικά προγράμματος επίσης κατά τον χρόνο εκτέλεσης. Το εργαλείο αυτό συλλέγει πληροφορίες που αφορούν τις αριθμιτικές πράξεις, τον έλεγχο ροής, τον υπολογισμό των επαναλήψεων αν ένας κλάδος χωριστεί σε ομάδες επαναλήψεων, κ.λπ.

Μηχανική Μάθηση (***Machine Learning***)

Η Μηχανική Μάθηση είναι ένας κλάδος της Τεχνητής Νοημοσύνης και της Επιστήμης των Υπολογιστών που εστιάζει στη χρήση δεδομένων και αλγορίθμων για τη μίμηση του τρόπου μάθησης των ανθρώπων, βελτιώνοντας σταδιακά την ακρίβειά της. Είναι ένα σημαντικό πεδίο του αναπτυσσόμενου τομέα της Επιστήμης των Δεδομένων. Οι αλγόριθμοι μηχανικής μάθησης δημιουργούν ένα μοντέλο που βασίζεται σε δείγματα δεδομένων, γνωστά ως δεδομένα εκπαίδευσης, προκειμένου να κάνουν προβλέψεις ή να παίρνουν αποφάσεις χωρίς να είναι ρητά προγραμματισμένοι να το κάνουν. Αυτοί οι αλγόριθμοι χρησιμοποιούνται σε μια μεγάλη ποικιλία εφαρμογών, όπως στην ιατρική, την αναγνώριση ομιλίας και την όραση υπολογιστών, όπου είναι δύσκολο ή ανέφικτο να αναπτυχθούν συμβατικοί αλγόριθμοι για την εκτέλεση των απαραίτητων εργασιών.

Επιτηρούμενη Μάθηση (***Supervised Learning***)

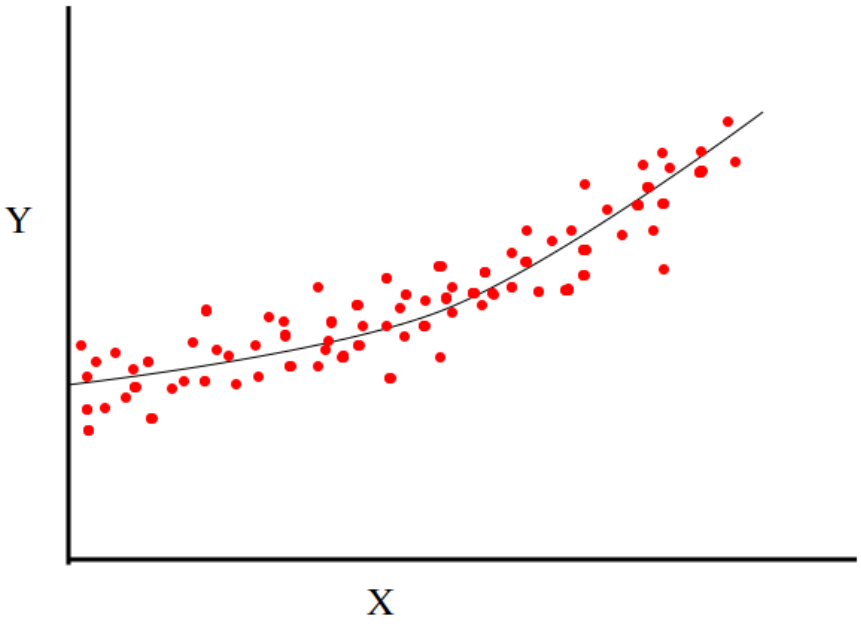
Η Επιτηρούμενη Μάθηση είναι η διαδικασία Μηχανικής Μάθησης για την εκ-

μάθηση μιας συνάρτησης που αντιστοιχίζει τις μεταβλητές εισόδου (x) με μια μεταβλητή εξόδου (Y) μέσω ενός αλγορίθμου με μαθηματική σχέση Y = f(x). Οι τεχνικές των αλγορίθμων εποπτευόμενης μηχανικής μάθησης περιλαμβάνουν γραμμική και λογιστική παλινδρόμηση, ταξινόμηση πολλαπλών κλάσεων, δέντρα αποφάσεων και μηχανές υποστήριξης διανυσμάτων. Η εποπτευόμενη μάθηση απαιτεί τα δεδομένα που χρησιμοποιούνται για την εκπαίδευση του αλγόριθμου να έχουν ήδη επισημανθεί με σωστές απαντήσεις. Για παράδειγμα, ένας αλγόριθμος ταξινόμησης θα μάθει να αναγνωρίζει ζώα αφού εκπαιδευτεί σε ένα σύνολο δεδομένων εικόνων που είναι κατάλληλα επισημασμένες με το είδος του ζώου και ορισμένα χαρακτηριστικά αναγνώρισης.

Τα επιτηρούμενα μαθησιακά προβλήματα μπορούν περαιτέρω να ομαδοποιηθούν σε προβλήματα Regression (Παλινδρόμησης) και Classification (Ταξινόμησης). Και τα δύο προβλήματα έχουν ως στόχο την κατασκευή ενός συνοπτικού μοντέλου που μπορεί να προβλέψει την τιμή της εξαρτημένης μεταβλητής από τις τιμές των ανεξάρτητων μεταβλητών. Η διαφορά μεταξύ των δύο εργασιών είναι το γεγονός ότι το εξαρτημένο χαρακτηριστικό είναι αριθμητικό για παλινδρόμηση και κατηγορικό για ταξινόμηση.

Παλινδρόμηση (***Regression***)

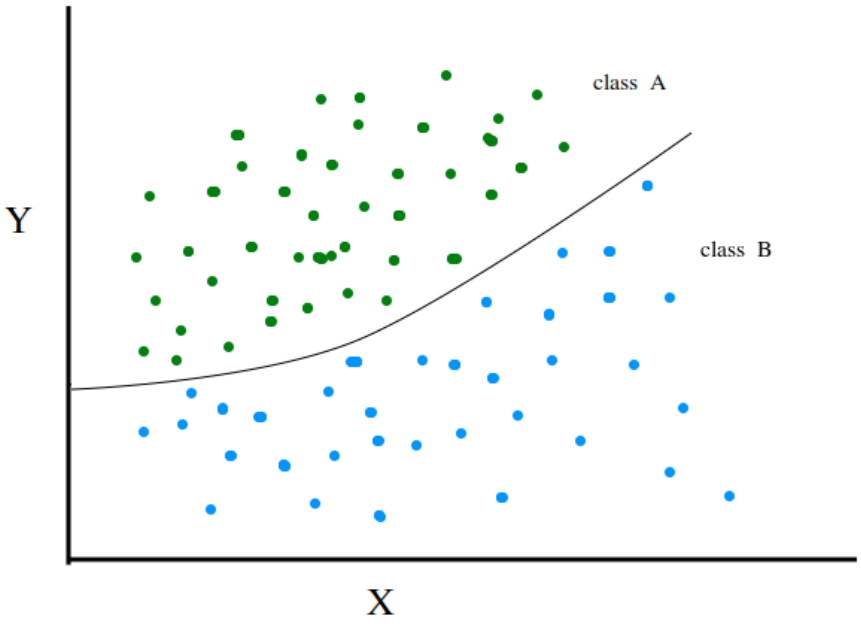
Η Παλινδρόμηση είναι μια επιτηρούμενη τεχνική μηχανικής μάθησης που χρησιμοποιείται για την πρόβλεψη συνεχών τιμών. Ο απώτερος στόχος ενός αλγορίθμου παλινδρόμησης είναι να σχεδιάσει μια γραμμή ή μια καμπύλη που αποτυπώνει καλύτερα την σχέση μευαξύ των δεδομένων, όπως δείχνει το παρακάτω σχήμα. Βοηθά στη δημιουργία μιας σχέσης μεταξύ των μεταβλητών εκτιμώντας πώς η μία μεταβλητή επηρεάζει την άλλη.



Σχήμα 1: Παλινδρόμηση

Ταξινόμηση (***Classification***)

Η ταξινόμηση είναι ένα μοντέλο πρόβλεψης που προσεγγίζει μια συνάρτηση αντιστοίχισης από μεταβλητές εισόδου για να προσδιορίσει διακριτές μεταβλητές εξόδου, οι οποίες μπορεί να είναι ετικέτες ή κατηγορίες. Η συνάρτηση χαρτογράφησης των αλγορίθμων ταξινόμησης είναι υπεύθυνη για την πρόβλεψη της ετικέτας ή της κατηγορίας των δεδομένων μεταβλητών εισόδου, όπως φαίνεται στο παρακάτω σχήμα. Ένας αλγόριθμος ταξινόμησης μπορεί να έχει τόσο διακριτές όσο και με πραγματική αξία μεταβλητές, αλλά απαιτεί τα παραδείγματα να ταξινομηθούν σε μία από δύο ή περισσότερες κλάσεις.



Σχήμα 2: Ταξινόμηση

Προτεινόμενο Εργαλείο

Μεθοδολογία

Αυτή η εργασία ξεκινά με έναν Code Generator που παράγει 700 αρχεία C σύμφωνα με ορισμένες προδιαγραφές. Στη συνέχεια, αυτά τα προγράμματα εφαρμόζονται στα εργαλεία Profiling τα οποία με την σειρά τους εξάγουν πολύτιμες πληροφορίες ενώ τις συγκεντρώνουν σε ένα αρχείο csv που ονομάζεται dataset και περιέχει τις X μεταβλητές. Παράλληλα, για τα 700 προγράμματα πραγματοποιείται μέτρηση της καταναλισκόμενης ενέργειας στην ενσωματωμένη πλατφόρμα του εργαστηρίου, και τα αποτελέσματα συγκεντρώνονται σε ένα δεύτερο αρχείο με το όνομα dataset_energy που αποτελεί την μεταβλητή Y. Μέσω αυτών των datasets, το μοντέλο θα εκπαιδευτεί. Στη συνέχεια, κατασκευάζεται ένα μοντέλο Παλινδρόμησης βασισμένο στον αλγόριθμο Lasso και οι παράμετροί του αποτελούν αντικείμενο βελτιστοποίησης. Ακολούθως, 56 benchmarks γλώσσας C από το PolyBench Suite εφαρμόζονται στα εργαλεία Profiling ακριβώς όπως τα 700 αρχεία κώδικα και το αρχείο test_dataset δημιουργείται και περιέχει τις μεταβλητές X_test. Μετά από όλα αυτά τα βήματα, είμαστε έτοιμοι να χρησιμοποιήσουμε το μοντέλο για να προβλέψουμε την κατανάλωση ενέργειας αυτών των 56 benchmarks.

Τέλος, οι ενέργειες που καταναλώνουν αυτά τα benchmarks (Y_test) μετράται επίσης προκειμένου να συγκριθούν με τις προβλεπόμενες ενέργειες (Y_pred) και να αξιολογηθεί η ακρίβεια της πρόβλεψης του εργαλείου.

Τα επιμέρους στάδια θα αναλυθούν περαιτέρω στη συνέχεια.

C Code Generator

Προκειμένου να δημιουργήσουμε ένα αρκετά μεγάλο σύνολο δεδομένων εκπαίδευσης, δημιουργήσαμε ένα script που δημιουργεί κώδικες γλώσσας C με το χαρακτηριστικό της ύπαρξης εμφωλευμένων βρόχων. Αυτός ο τύπος προγραμμάτων θεωρείται ιδιαίτερα ενδιαφέρον, καθώς η κατανάλωση ενέργειας κατά τη διάρκεια της εκτέλεσης των εμφωλευμένων βρόχων γίνεται εξαιρετικά υψηλή. Αυτό συμβαίνει, λόγω του μεγάλου αριθμού αλμάτων του μετρητή προγράμματος (PC), καθώς και της μη βέλτιστης συμπλήρωσης της κρυφής μνήμης.

Οι παραγόμενοι κώδικες ξεκινούν με την δημιουργία έως και 4 πινάκων ακέραιων αριθμών και το πολύ 4 πινάκων αριθμών κινητής υποδιαστολής, που έχουν το πολύ 6 διαστάσεις και κάθε διάσταση περιέχει το πολύ 1000 στοιχεία. Αυτά τα χαρακτηριστικά εντάσσουν τους κώδικες αυτούς μέσα στο πλαίσιο των προγραμμάτων χαμηλών και μεσαίων ενεργειακών απαιτήσεων. Αυτός ο τύπος προγραμμάτων είναι αυτός που αυτό το εργαλείο θα μάθει να εκτιμά ενεργειακά.

Καθώς ολοκληρώνεται η προετοιμασία, οι παραπάνω πίνακες λαμβάνουν τυχαιοποιημένες τιμές, σύμφωνα με τον τύπο μεταβλητών κάθε πίνακα και εκχωρείται ο κατάλληλος όγκος μνήμης. Το στοιχείο κάθε πίνακα έχει μια τιμή και το πιο ¨επιβαρυμένο' τμήμα του κώδικα είναι έτοιμο να ξεκινήσει.

Στη συνέχεια, το τμήμα των εμφωλευμένων βρόχων αναλαμβάνει. Αρχικά, η οδηγία #pragma scop" τοποθετείται πριν από τον πρώτο βρόχο, προκειμένου να ενημερώσει το script ανάλυσης ότι ξεκινά η ενότητα. Μέσα στους ένθετους βρόχους, περιλαμβάνεται ένα τμήμα υπολογισμού, όπου οι τύποι υπολογισμών (πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση) καθορίζονται τυχαία. Μετά το κλείσιμο όλων των βρόχων, μια δεύτερη οδηγία "#pragma endscop" τοποθετείται κάτω από την τελευταία αγκύλη. Αυτό ενημερώνει το script ανάλυσης ότι η ενότητα των εμφωλευμένων βρόχων έχει ολοκληρωθεί. Ο παραγόμενος κώδικας τελειώνει με ένα τελευταίο τμήμα απελευθέρωσης του χρησιμοποιούμενου χώρου μνήμης και στη συνέχεια επιστρέφοντας το 0, ενημερώνοντας ότι η εκτέλεση ολοκληρώνεται σωστά.

Δημιουργία των Datasets

Ένα python script με όνομα "total_analysis.py" είναι υπεύθυνο για τη δημιουργία προφίλ των 700 προγραμμάτων, καθώς εφαρμόζει κάθε πρόγραμμα στα τέσσερα υποεργαλεία Profiling που περιγράφηκαν παραπάνω. Στη συνέχεια, αναζητά τα εξαγώμενα των εργαλείων αρχεία που παράγονται για να συγκεντρώσει τις πληροφορίες που περιέχουν. Αναφορικά με το .cachegrind out-file, το script συλλέγει μετρήσεις τόσο για ολόκληρο το πρόγραμμα (σύνοψη) όσο και για τον εμφωλευμένο βρόχο ξεχωριστά. Αυτή είναι η φάση που η οδηγία #pragma βοηθά πραγματικά. Δευτερευόντως, το αρχείο .massif out-file δίνει τις εξαγόμενες

πληροφορίες για ένα στιγμιότυπο. Σε αυτό το στάδιο, το script καταφέρνει να εξάγει τις πληροφορίες στο στιγμιότυπο peak, το οποίο λαμβάνεται κατά τη διάρκεια των εμφωλευμένων βρόχων. Στη συνέχεια, τα αρχεία εξόδου των MICA και PinTool παρέχουν πληροφορίες που προσομοιώνουν τη συμπεριφορά ολόκληρου του προγράμματος με διαφορετικά μεγέθη παραθύρων ή διαφορετικές πλατφόρμες. Έτσι, το script total_analysis.py εξάγει όλες αυτές τις πληροφορίες. Τέλος, αυτές οι χρήσιμες πληροφορίες (συνολικά 141) συλλέγονται για καθένα από τους 700 κώδικες που δημιουργήθηκαν και το script τις εγγράφει σε ένα αρχείο csv που ονομάζεται dataset.csv. Αυτές οι μετρήσεις είναι πλέον τα features του dataset εκπαίδευσης X και απεικονίζονται ως στήλες, ενώ οι 700 κωδικοί απεικονίζονται ως γραμμές.

Παράλληλα, πρέπει να γίνει η μέτρηση της καταναλισκόμενης ενέργειας αυτών των 700 προγραμμάτων προκειμένου να δημιουργηθεί το dataset y και να εκπαιδευτεί το μοντέλο. Για αυτόν τον λόγο, ένα άλλο script με το όνομα collect_train_codes_energy.py έχει αναπτυχθεί στο target μηχάνημα του εργαστηρίου. Η μέτρηση πραγματοποιείται εκτελώντας το εκάστοτε πρόγραμμα 20 φορές και ύστερα διαιρώντας την τιμή που μετρήθηκε με 20, ώστε να επιτευχθεί μεγαλύτερη ακρίβεια στην μέτρηση. Αμέσως μετά τη μέτρηση, το script καταγράφει κάθε τιμή σε ένα αρχείο csv μιας στήλης που ονομάζεται dataset_energy.csv, το οποίο τελικά περιέχει την κατανάλωση ενέργειας των 700 αρχείων κώδικα. Οι κώδικες απεικονίζονται ως σειρές όπως και πριν, και όλες αυτές οι πραγματικές ενέργειες αποτελούν πλέον την μεταβλητή y.

Εκπαίδευση του Μοντέλου

Προοετοιμασία

Είναι σημαντικό να αναφέρουμε ότι οι περισσότερες από τις συναρτήσεις και τους αλγορίθμους που ακολουθούν αποτελούν μέρος του Scikit-learn, επίσης γνωστού ως sklearn, το οποίο είναι μια βιβλιοθήκη μηχανικής μάθησης ελεύθερου λογισμικού για τη γλώσσα προγραμματισμού Python.

Σε αυτό το σημείο, που τα σύνολα δεδομένων εκπαίδευσης X και y είναι έτοιμα, είναι η ώρα να επιλεχθεί ο παλινδρομητής που ταιριάζει στις ανάγκες του προβλήματος και να ευρεθούν οι καλύτερες παράμετροι για τη βελτιστοποίηση των προβλέψεων. Για τον σκοπό αυτό, ένα νέο script με όνομα set_up_Regressor.py δημιουργήθηκε για την υλοποίηση αυτών των εργασιών και τα τμήματά του περιγράφονται παρακάτω.

Ως πρώτο βήμα, τα σύνολα δεδομένων που διευκρινίστηκαν παραπάνω εισάγονται και ορίζονται ως dataset X και dataset y. Το X αναφέρεται σε όλα τα χαρακτηριστικά του κάθε κώδικα και το y αναφέρεται στην ενέργεια του κάθε κώδικα. Η λίστα X έχει σχήμα (700,141), ενώ η λίστα y έχει σχήμα (1,141).

Στη συνέχεια, πρέπει να γίνει ένας διαχωρισμός προκειμένου να αφήσουμε να οριστεί το μέγεθος δοκιμής σε σχέση με το μέγεθος εκπαίδευσης. Η αναλογία που επιλέχθηκε είναι 1/5, που σημαίνει ότι για 560 γραμμές εκπαίδευσης, οι υπόλοιπες 140 γραμμές αντιστοιχούν σε δοκιμή. Τώρα, δημιουργήσαμε τα X_train, y_train, X_test και y_test, με σκοπό την δοκιμή πολλών συνδυασμών.

Καθώς πρόκειται να γίνουν τυχαίες παλινδρομήσεις με τη χρήση των διαχωρισμών που περιγράφονται παραπάνω, είναι χρήσιμο να δημιουργηθεί ένα pipeline που αποτελείται από δύο μέρη: το μέρος της προεπεξεργασίας (scaling) και το μέρος του παλινδρομητή (regressor).

Σε αυτό το σημείο, είναι πολύ χρήσιμο να κάνουμε κάποια κανονικοποίηση στις τιμές της μεταβλητής X προκειμένου να βοηθήσουμε τον παλινδρομητή. Είναι πιο δύσκολο για αυτόν να διαχειριστεί τιμές με μεγάλο εύρος, γι΄ αυτό επιλέξαμε να χρησιμοποιήσουμε την συνάρτηση StandardScaler() για να μειώσουμε το πλάτος των τιμών των χαρακτηριστικών. Η StandardScaler() κανονικοποιεί ένα χαρακτηριστικό αφαιρώντας τη μέση τιμή και στη συνέχεια κλιμακώνει σε μοναδιαία διακύμανση. Μοναδιαία διακύμανση σημαίνει ότι διαιρούμε όλες τις τιμές με την τυπική απόκλιση. Η συνάρτηση StandardScaler() κάνει τη μέση τιμή της κατανομής 0 και τελικά οι περισσότερες τιμές θα βρίσκονται μεταξύ -1 και 1.

Μοντέλο Lasso

Κάθε πρόβλημα πρόβλεψης έχει έναν κατάλληλο αλγόριθμο που ταιριάζει καλύτερα στο πρόβλημα. Έτσι, μια θεωρητική έρευνα μπορεί να κατευθύνει τον προγραμματιστή στον κατάλληλο αλγόριθμο, αλλά πρέπει να γίνει δοκιμή προκειμένου να επιλεγεί σωστά ο κατάλληλος. Το κριτήριο σύμφωνα με το οποίο συγκρίναμε τους αλγορίθμους παλινδρόμησης ήταν το μέσο τετραγωνικό σφάλμα (MSE) της πρόβλεψης που έγινε με τον διαχωρισμό του συνόλου δεδομένων όπως περιγράφηκε παραπάνω. Δοκιμάστηκαν πολλοί αλγόριθμοι και τα αποτελέσματα παρουσιάζονται παρακάτω.

Regression Model	Mean Squared Error
Ridge	2.3933
Lasso	0.5310
Random Forest	37.223
Stohastic Gradient Descent	∞
Bayesian Ridge	1.3658
K-Nearest Neighbors	40.239
Gradient Boosting	39.317
Decision Tree	39.743

Σχήμα 3: Σύγκριση Μοντέλων

Το προφανές αποτέλεσμα αυτών των δοκιμών είναι ότι ο Lasso Regressor ταιριάζει βέλτιστα στο πρόβλημά μας. Αυτό ήταν πρωτίστως αναμενόμενο, καθώς το σύνολο δεδομένων που χρησιμοποιούμε είναι αρκετά ογκώδες και τα features συσχετίζονται αρκετά.

Το Lasso σημαίνει Least Absolute Selection Shrinkage Operator, όπου η συρρίκνωση (shrinkage) ορίζεται ως περιορισμός στις παραμέτρους. Ο στόχος της παλινδρόμησης Lasso είναι να ληφθεί υπόψιν μόνο το υποσύνολο των προβλεπτών που ελαχιστοποιεί το σφάλμα πρόβλεψης για μια ποσοτική μεταβλητή απόκρισης. Ο αλγόριθμος λειτουργεί επιβάλλοντας έναν περιορισμό στις παραμέτρους του μοντέλου, που προκαλεί συρρίκνωση των συντελεστών παλινδρόμησης για ορισμένες μεταβλητές προς το μηδέν.

Οι μεταβλητές με συντελεστή παλινδρόμησης ίσο με μηδέν μετά τη διαδικασία συρρίκνωσης αποκλείονται από το μοντέλο. Οι μεταβλητές με μη μηδενικούς συντελεστές παλινδρόμησης μεταβλητών συνδέονται πιο ισχυρά με τη μεταβλητή απόκρισης. Οι επεξηγηματικές μεταβλητές μπορεί να είναι είτε ποσοτικές, είτε κατηγορικές είτε και τα δύο. Αυτή η ανάλυση παλινδρόμησης lasso είναι ουσιαστικά μια μέθοδος συρρίκνωσης και επιλογής μεταβλητών και βοηθά τους αναλυτές να καθορίσουν ποιοι από τους προγνωστικούς παράγοντες είναι πιο σημαντικοί.

Ο αλγόριθμος παλινδρόμησης Lasso προτιμάται ιδιαίτερα όταν υπάρχει υψηλή πολυσυγγραμμικότητα στο σύνολο δεδομένων. Η πολυσυγγραμμικότητα στο σύνολο δεδομένων σημαίνει ότι οι ανεξάρτητες μεταβλητές σχετίζονται σε μεγάλο βαθμό μεταξύ τους και μια μικρή αλλαγή στα δεδομένα μπορεί να προκαλέσει μεγάλη αλλαγή στους συντελεστές παλινδρόμησης. Το δικό μας σύνολο δεδομένων έχει υψηλή πολυσυγγραμμικότητα, καθώς έχει ισχυρή συσχέτιση μεταξύ των ανεξάρτητων μεταβλητών, και αυτός είναι ο κύριος λόγος που ο συγκεκριμένος αλγόριθμος δίνει τα καλύτερα αποτελέσματα για τη συγκεκριμένη εφαρμογή.

Η διαδικασία της συρρίκνωσης πραγματοποιείται μέσω της L1-κανονικοποίησης, κατά την οποία τιμωρούνται ορισμένα φεατυρες σε ένα μοντέλο, προκειμένου να διατηρηθούν μόνο τα πιο σημαντικά features, χρησιμοποιώντας μια παράμετρο, το alpha. Όταν το alpha είναι 0, η παλινδρόμηση Lasso παράγει τους ίδιους συντελεστές με μια γραμμική παλινδρόμηση. Όταν το alpha είναι πολύ μεγάλο, όλοι οι συντελεστές είναι μηδενικοί.

Grid Search και Cross-Validation

Grid Search είναι μια τεχνική για την επιλογή του καλύτερου μοντέλου μηχανικής μάθησης, παραμετροποιημένο από ένα πλέγμα υπερπαραμέτρων. Δοκιμάζει όλους τους συνδυασμούς πλέγματος παραμέτρων για ένα μοντέλο και επιστρέφει με το καλύτερο σύνολο παραμέτρων που έχει την καλύτερη βαθμολογία απόδοσης. Το Cross-Validation είναι μια μέθοδος επαναδειγματοληψίας που χρησιμοποιεί διαφορετικά τμήματα των δεδομένων για τη δοκιμή και την εκπαίδευση ενός μοντέλου σε διαφορετικές επαναλήψεις. Ειδικότερα, το Cross-Validation k-folds είναι ένας τύπος Cross-Validation, όπου τα δεδομένα εκπαίδευσης χωρίζονται σε k-υποσύνολα. Από αυτά, (k-1) υποσύνολα χρησιμοποιούνται για την εκπαίδευση και το k-οστό υποσύνολο χρησιμοποιείται για την δοκιμή του μοντέλου.

Η διαδικασία αυτή ξεκινά με τον ορισμό του τυχαίου πλέγματος των παραμέτρων. Εκτός από το alpha, που είναι η πιο σημαντική παράμετρος όπως αναφέρθηκε προηγουμένως, εμπλουτίσαμε το τυχαίο πλέγμα με δύο ακόμη παραμέτρους. Ο μέγιστος αριθμός επαναλήψεων καθορίζεται από την παράμετρο max_iter, ενώ η ανοχή για τη βελτιστοποίηση καθορίζεται από την παράμετρο tol.

Μετά τη δημιουργία του τυχαίου πλέγματος, επιλέγεται η τεχνική Repeated K Fold για να γίνει πιο έγκυρο το Cross-Validation. Η συνάρτηση RepeatedKFold() έκανε 10 διαχωρισμούς και 20 επαναλήψεις στο σύνολο δεδομένων, που σημαίνει 200 διαφορετικές περιπτώσεις. Αυτές οι περιπτώσεις αξιολογήθηκαν μέσω της συνάρτησης βαθμολόγησης Negative Mean Squared Error (NMSE). Μετά την προσαρμογή των δεδομένων Ξ και ψ στο πλέγμα, έχουμε τα ακόλουθα αποτελέσματα:

- Negative mean squared error of the best_estimator: -1.5002
- Parameter settings that gave the best results: 'regressor__alpha': 0.3, 'regressor__max_iter': 200, 'regressor__tol': 0.01

Αυτές οι παράμετροι θα εφαρμοστούν στον παλινδρομητή Lasso και θα σχηματίσουν αυτό το μοντέλο μηχανικής μάθησης.

Πειραματικά Αποτελέσματα

Μετά την εύρεση των καλύτερων παραμέτρων για το μοντέλο με βάση τον αλγόριθμο Lasso, δημιουργήθηκε ένα νέο script με όνομα Predict_energy.py. Σε αυτό το script περνάμε από ορισμένα test ανάλυσης των features, προκειμένου να εξαχθούν συμπεράσματα σχετικά με τη σημασία των features που χρησιμοποιήθηκαν για την εκπαίδευση του μοντέλου. Επίσης, για την αξιολόγησή του χρησιμοποιήθηκαν πραγματικά benchmarks που λήφθηκαν από το Polybench, μια συλλογή benchmarks που θεωρείται αντιπροσωπευτική για ενσωματωμένες εφαρμογές. Το εν λόγω benchmark suite επιλέχθηκε επειδή παρέχει στον προγραμματιστή τη δυνατότητα μεταγλώττισης σε σύνολα δεδομένων διαφορετικού μεγέθους (διαφορετικά μεγέθη πινάκων). Για το παρόν έργο επιλέξαμε να χρησιμοποιήσουμε τα μικρά και μεσαία σύνολα δεδομένων για τα 28 ακόλουθα benchmarks . Αυτή η επιλογή έγινε για να ορίσει τα benchmarks ως προγράμματα χαμηλών και μεσαίων ενεργειακών απαιτήσεων. Αυτό σημαίνει ότι το σύνολο των προγραμμάτων δοκιμής αποτελείται από 28x2=56 κώδικες.

Benchmark	Description
gemm	Matrix-multiply C=alpha.A.B+beta.C
jacobi-2D	2-D Jacobi stencil computation
heat-3d	3-D Heat stencil computation
trisolv	Triangular solver
symm	Symmetric matrix-multiply
trmm	Triangular matrix-multiply
atax	Matrix Transpose and Vector Multiplication
durbin	Toeplitz system solver
covariance	Covariance Computation
gemver	Vector Multiplication and Matrix Addition
gesummv	Scalar, Vector and Matrix Multiplication
syr2k	Symmetric rank-2k operations
syrk	Symmetric rank-k operations
2mm	2 Matrix Multiplications (D=A.B; E=C.D)
3mm	3 Matrix Multiplications (E=A.B; F=C.D; G=E.F)
bicg	BiCG Sub Kernel of BiCGStab Linear Solver
doitgen	Multiresolution analysis kernel (MADNESS)
mvt	Matrix Vector Product and Transpose
cholesky	Cholesky Decomposition
gramschmidt	Gram-Schmidt decomposition
correlation	Correlation Computation
lu	LU decomposition
ludcmp	LU decomposition
floyd-warshall	Floyd–Warshall algorithm
adi	Alternating Direction Implicit solver
fdtd-2d	2-D Finite Different Time Domain Kernel
jacobi-1D	1-D Jacobi stencil computation
seidel	2-D Seidel stencil computation

Σχήμα 4: Λίστα με τα προγράμματα αξιολόγησης

Αυτά τα 56 benchmarks περνούν από το script "real_test_analysis.py". Αυτό το script είναι πανομοιότυπο με το "total_analysis.py", που παρουσιάστηκε προηγουμένως. Με αυτή τη διαδικασία, τα εργαλεία σκιαγράφησης προφίλ συλλέγουν όλα τα δεδομένα και δημιουργούν το X test dataset. Παράλληλα, πρέπει να γίνει η μέτρηση της κατανάλωσης ενέργειας αυτών των 56 benchmarks, προκειμένου να συγκριθούν οι πραγματικές ενέργειές τους με τις προβλεπόμενες. Για το λόγο αυτό, εκτελείται στο target-platform το script με όνομα collect_energy.py. Αυτό το script είναι παρόμοιο με το collect_train_codes_energy.py, που παρουσιάστηκε προηγουμένως και δημιουργεί το y test dataset.

Το script Predict_energy.py ξεκινά με την εισαγωγή των train και test datasets. Το train dataset περιέχει τους 700 παραγόμενους κώδικες και το test dataset περιέχει τα 56 benchmarks που περιγράφηκαν παραπάνω.

Σημαντικότητα των Features

Σε αυτό το σημείο επικεντρωνόμαστε στη σημασία που δίνει ο παλινδρομητής Lasso στα features. Ο λόγος για τον οποίο επιλέξαμε να βάλουμε πολλά features στο σύνολο δεδομένων δεν ήταν μόνο για να δώσουμε αρκετές πληροφορίες στο μοντέλο, αφού ο Lasso Regressor κάνει μια επιλογή features και δεν τα διαχειρίζεται όλα, αλλά και για να προχωρήσουμε σε κάποια ανάλυση σχετικά με τη σημασία που δίνει ο παλινδρομητής στις μετρικές όσον αφορά με την κατανάλωση ενέργειας.

Όπως αναφέρθηκε προηγουμένως, ο Lasso Regressor εφαρμόζει έναν συντελεστή βάρους στα χαρακτηριστικά. Τα περισσότερα από τα χαρακτηριστικά συρρικνώνονται στο μηδέν και τα υπόλοιπα χαρακτηριστικά έχουν ένα βάρος που ονομάζεται σημαντικότητα. Η ανάλυση των χαρακτηριστικών αρχίζει με την εύρεση των χαρακτηριστικών που έχουν μη μηδενική σημαντικότητα και κάνουν τον παλινδρομητή να επιτύχει τα βέλτιστα αποτελέσματα. Ο κατάλογος που αποφάσισε να κρατήσει ο Lasso Regressor παρουσιάζεται παρακάτω.

```
---------find the important Lasso coefficients-------------------------
Feature: D1mr_total has Lasso non zero importance: 0.05392
Feature: DLmr_total has Lasso non zero importance: 0.11928
Feature: Dw_total has Lasso non zero importance: 0.68814
Feature: Bc_total has Lasso non zero importance: 0.18763
Feature: Bcm_total has Lasso non zero importance: 0.18580
Feature: mem_heap_B has Lasso non zero importance: 0.37906
Feature: ilp has Lasso non zero importance: 2.48196
Feature: ilp.1 has Lasso non zero importance: 0.00165
Feature: ilp.2 has Lasso non zero importance: 0.03192
Feature: ilp.3 has Lasso non zero importance: 0.04219
Feature: ilp.4 has Lasso non zero importance: 0.02634
Feature: itypes has Lasso non zero importance: 0.01440
Feature: itypes.3 has Lasso non zero importance: 0.32430
Feature: itypes.5 has Lasso non zero importance: 0.09939
Feature: memfootprint has Lasso non zero importance: 0.43879
Feature: memfootprint.1 has Lasso non zero importance: 0.33088
Feature: stride.1 has Lasso non zero importance: 0.24014
```

Σχήμα 5: Σημαντικότητα των Χαρακτηριστικών

Όπως φαίνεται, τα πιο σημαντικά χαρακτηριστικά είναι τα εξής:

• Τα D1mr_total, DLmr_total και Dw_total αναφέρονται στις αναγνώσεις και εγγραφές ολόκληρου του προγράμματος και αυτές οι λειτουργίες μνήμης είναι πολύ βαριές από ενεργειακή άποψη. Ειδικά οι εγγραφές δεδομένων φαίνεται να έχουν μεγάλη επίδραση στην ενεργειακή κατανάλωση του κώδικα.
• Τα στοιχεία Bc_total και Bcm_total αντιστοιχούν στις εκτελούμενες υπό όρους διακλαδώσεις και στις λανθασμένα προβλεπόμενες υπό όρους διακλαδώσεις ολόκληρου του προγράμματος. Ο μετρητής προγράμματος αποθηκεύει τη διεύθυνση

μνήμης της επόμενης εντολής που πρόκειται να εκτελεστεί και όταν γίνεται μια διακλάδωση, ο μετρητής προγράμματος του επεξεργαστή τίθεται στην θέση της εντολής άλματος, πράγμα που είναι εξαιρετικά δαπανηρό.
• mem_heap_B που αναφέρεται στη μνήμη σωρού, η οποία είναι μια μνήμη που μοιράζονται όλα τα νήματα που εκτελούνται στην εφαρμογή. Αυτή η μνήμη επηρεάζει σημαντικά την ισχύ και τη χρήση ενέργειας του συνολικού συστήματος διακομιστή.
• ilp, ilp.1, ilp.2, ilp.3 και ilp.4 τα οποία αντιστοιχούν στην παράλληλη ή ταυτόχρονη εκτέλεση μιας ακολουθίας εντολών σε πέντε διαφορετικά παράθυρα εντολών (το παράθυρο μεγέθους 16 φαίνεται να είναι μακράν το χαρακτηριστικό με τη μεγαλύτερη επιρροή). Ο Παράλληλος Προγραμματισμός οδηγεί σε βελτίωση της απόδοσης, αλλά η ανταλλαγή δεδομένων κατά την εκτέλεση μέσω περιοχών κοινής μνήμης οδηγεί σε υψηλότερη κατανάλωση ενέργειας.
• itypes, itypes.3 και itypes.5 αντιστοιχούν στον αριθμό των αναγνώσεων μνήμης, των αριθμητικών υπολογισμών και της χρήσης στοίβας. Οι αριθμητικοί υπολογισμοί είναι μια βαριά εργασία που διαχειρίζεται η αριθμητική/λογική μονάδα (ALU) της CPU και συχνά απαιτεί πολλή ενέργεια, ιδίως στην περίπτωση της διαίρεσης. Η χρήση της στοίβας φαίνεται να είναι επίσης σημαντική για τον παλινδρομητή, καθώς η σωστή κατανομή μνήμης οδηγεί σε χαμηλότερη κατανάλωση ενέργειας.
• Τα στοιχεία memfootprint και memfootprint.1 αναφέρονται στον αριθμό των μπλοκ (64-byte) και των σελίδων (4KB) που αγγίζουν οι διευθύνσεις δεδομένων. Το μέγεθος της κύριας μνήμης που χρησιμοποιεί ή αναφέρει ένα πρόγραμμα κατά την εκτέλεσή του αποτελεί βασικό παράγοντα για τον Lasso Regressor.
• Το stride.1 αντιστοιχεί στις αποστάσεις επαναχρησιμοποίησης μνήμης και αναφέρεται εν συντομία στις απαραίτητες αλλαγές κρυφής μνήμης που είναι πολύ σημαντικές, καθώς όσο μεγαλύτερες είναι αυτές οι αποστάσεις, τόσο πιο βαρύ καταλήγει να είναι το πρόγραμμα.

Συσχέτιση των Features

Αυτή η επιλογή χαρακτηριστικών που πραγματοποιεί ο παλινδρομητής Lasso θεωρείται λογική. Όμως η ανάλυση των χαρακτηριστικών περιέχει ένα άλλο κρίσιμο μέρος, τον έλεγχο συσχέτισης. Για το σκοπό αυτό εξετάζεται η συσχέτιση Spearman, καθώς αξιολογεί τη μονοτονική σχέση μεταξύ δύο μεταβλητών. Η διαδικασία αυτή υπολογίζει έναν συντελεστή για κάθε χαρακτηριστικό, ο οποίος κυμαίνεται μεταξύ -1 και +1, με το 0 να σημαίνει ότι δεν υπάρχει συσχέτιση. Οι συσχετίσεις -1 ή +1 υποδηλώνουν ακριβή μονοτονική σχέση. Οι θετικές συσχετίσεις υποδηλώνουν ότι καθώς αυξάνεται το x, αυξάνεται και το y. Οι αρνητικές συσχετίσεις υποδηλώνουν ότι καθώς αυξάνεται το x, μειώνεται το y. Τα αποτελέσματα αυτού του test παρουσιάζεται παρακάτω.

```
SPEARMAN STRONG X-y CORRELATIONS
Feature: D1mr has Strong Spearman correlation: 0.975110
Feature: Ir_total has Strong Spearman correlation: 0.979055
Feature: Dr_total has Strong Spearman correlation: 0.978974
Feature: Dw_total has Strong Spearman correlation: 0.977122
Feature: D1mw_total has Strong Spearman correlation: 0.975812
Feature: DLmw_total has Strong Spearman correlation: 0.977342
Feature: #INS has Strong Spearman correlation: 0.979055
----------------------------------------------------------------------
SPEARMAN WEAK X-y CORRELATIONS
Feature: memstackdist.9 has Weak Spearman correlation: -0.017775
Feature: memstackdist.13 has Weak Spearman correlation: 0.074254
Feature: SP has Weak Spearman correlation: -0.022092
Feature: FDIV has Weak Spearman correlation: -0.014320
```

Σχήμα 6: Ισχυρές και Ασθενείς Συσχετίσεις

Τα ισχυρώς συσχετιζόμενα με την ενέργεια χαρακτηριστικά είναι τα εξής:

• D1mr, το οποίο αναφέρεται στις αστοχίες ανάγνωσης της κρυφής μνήμης D1 του τμήματος βρόχου. Αυτό σημαίνει ότι οι αστοχίες ανάγνωσης που συμβαίνουν στο εσωτερικό των εμφωλευμένων βρόχων είναι πολύ επιδραστικές ενεργειακά.

• Ir_total και #INS, τα οποία μετρούν τον αριθμό των εντολών που εκτελέστηκαν και είναι τα πιο συσχετιζόμενα χαρακτηριστικά με την κατανάλωση ενέργειας, καθώς περιγράφουν πρακτικά το μέγεθος του εκτελούμενου προγράμματος.

• Dr_total, το οποίο αντιστοιχεί στον αριθμό των αναγνώσεων μνήμης ολόκληρου του προγράμματος. Μια λειτουργία ανάγνωσης μνήμης μεταφέρει την επιθυμητή λέξη στις γραμμές διεύθυνσης και ενεργοποιεί τη γραμμή ελέγχου ανάγνωσης. Καθώς ο αριθμός αυτός αυξάνεται, αυξάνονται και οι απαιτήσεις ενέργειας.

• Dw_total, το οποίο αντιστοιχεί στον αριθμό των εγγραφών μνήμης ολόκληρου του προγράμματος. Μια λειτουργία εγγραφής μνήμης μεταφέρει τη διεύθυνση της επιθυμητής λέξης στις γραμμές διεύθυνσης, μεταφέρει τα bit δεδομένων που πρόκειται να αποθηκευτούν στη μνήμη στις γραμμές εισόδου δεδομένων. Αυτή η διαδικασία έχει μεγάλη επίδραση στην κατανάλωση ενέργειας.

• D1mw_total και DLmw_total, τα οποία αντιστοιχούν στις αστοχίες εγγραφής της κρυφής μνήμης D1 και της κρυφής μνήμης LL. Όταν συμβαίνει μια αστοχία εγγραφής, απαιτείται από την εφαρμογή να κάνει μια δεύτερη προσπάθεια εντοπισμού των δεδομένων, αυτή τη φορά έναντι της πιο αργής κύριας βάσης δεδομένων και αυτή η διαδικασία έχει υψηλό ενεργειακό κόστος.

Τα αποτελέσματα αυτά αναδεικνύουν τη σημασία των λειτουργιών της κρυφής μνήμης όσον αφορά την κατανάλωση ενέργειας. Ως εκ τούτου, είναι εξαιρετικά σημαντικό να χρησιμοποιούνται ενσωματωμένα συστήματα με προδιαγραφές κρυφής μνήμης που προσαρμόζονται στην εφαρμογή (π.χ. πολλαπλά επίπεδα κρυφής μνήμης) και να σχεδιάζεται ο κώδικας με τρόπο που να μειώνει τις αστοχίες της κρυφής μνήμης, προκειμένου να επιτευχθεί καλή ενεργειακή απόδοση.

Από την άλλη πλευρά, τα ασθενώς συσχετιζόμενα χαρακτηριστικά είναι:

• memstackdist, το οποίο αναφέρεται στις αποστάσεις επαναχρησιμοποίησης

της μνήμης. Για κάθε ανάγνωση μνήμης, προσδιορίζεται το αντίστοιχο μπλοκ κρυφής μνήμης 64 byte και για κάθε μπλοκ κρυφής μνήμης στο οποίο γίνεται πρόσβαση, προσδιορίζεται ο αριθμός των μοναδικών μπλοκ κρυφής μνήμης στα οποία έγινε πρόσβαση από την τελευταία φορά που έγινε αναφορά σε αυτό, χρησιμοποιώντας μια στοίβα LRU. Πρόκειται για μια πολύ λεπτομερή μετρική που μπορεί να βοηθήσει στην περαιτέρω ανάλυση, αλλά όχι τόσο πολύ όσον αφορά την κατανάλωση ενέργειας.

• SP, το οποίο είναι ο αριθμός των πράξεων κινητής υποδιαστολής μονής ακρίβειας. Αυτή η μετρική είναι λογικά ασθενώς συσχετισμένη, καθώς μετρά έναν πολύ συγκεκριμένο τύπο αριθμητικών πράξεων.

• FDIV, το οποίο αναφέρεται στις πράξεις διαίρεσης (κινητής υποδιαστολής) και δεν συσχετίζεται τόσο πολύ με την κατανάλωση ενέργειας, για παρόμοιους λόγους με το SP.

Εφαρμογή του Μοντέλου

Το σύνολο δεδομένων εκπαίδευσης X περνάει από τη διαδικασία της προεπεξεργασίας. Για την εργασία αυτή, χρησιμοποιείται η συνάρτηση StandardScaler() για τη μείωση του πλάτους των τιμών των χαρακτηριστικών. Η StandardScaler() τυποποιεί ένα χαρακτηριστικό αφαιρώντας τη μέση τιμή και στη συνέχεια κανονικοποιεί σε μοναδιαία διακύμανση. Μοναδιαία διακύμανση σημαίνει διαίρεση όλων των τιμών με την τυπική απόκλιση. Η StandardScaler() κάνει τη μέση τιμή της κατανομής 0 και τελικά οι περισσότερες τιμές θα βρίσκονται μεταξύ -1 και 1.

Στη συνέχεια, εισάγουμε τον LassoRegressor() και τον ορίζουμε με τις παραμέτρους που προέκυψαν από το script set_up_Regressor.py που παρουσιάστηκε στο παραπάνω. Οι παράμετροι που προέκυψαν υπενθυμίζονται επίσης σε αυτό το σημείο:

• Parameter settings: 'regressor__alpha': 0.3, 'regressor__max_iter': 200, 'regressor__tol': 0.01

Τα σύνολα δεδομένων X train scaled και y train προσαρμόζονται στον παλινδρομητή και το μοντέλο μας είναι πλέον έτοιμο να αξιολογηθεί μέσω των πραγματικών benchmarks. Αρχικά, το σύνολο δεδομένων X test μετασχηματίζεται από τον scaler, καθώς το μοντέλο εκπαιδεύτηκε μέσω κανονικοποιημένων δεδομένων. Στη συνέχεια, ο παλινδρομητής χρησιμοποιεί αυτό το κλιμακωμένο σύνολο δεδομένων X test scaled για να προβλέψει τις ενέργειες των benchmarks. Δημιουργείται το σύνολο δεδομένων y pred, που περιέχει τις προβλεπόμενες ενέργειες και το μοντέλο μας είναι έτοιμο για αξιολόγηση.

Αξιολόγηση του Μοντέλου

Η ακρίβεια σε ένα μοντέλο παλινδρόμησης είναι ελαφρώς δύσκολο να βαθμολογηθεί. Είναι αδύνατο για ένα μοντέλο να προβλέψει την ακριβή τιμή, αλλά είναι απαραίτητο να εξεταστεί πόσο κοντά είναι η πρόβλεψη σε σχέση με την πραγματική τιμή. Για τη βαθμολόγηση της απόδοσης ενός μοντέλου παλινδρόμησης,

υπάρχουν διάφορες μετρικές που αξιολογούν την ακρίβεια των προβλέψεων. Οι μετρικές αξιολόγησης που επιλέχθηκαν για το παρόν έργο παρουσιάζονται παρακάτω:

• **Spearman Συσχέτιση**. Η συσχέτιση Spearman είναι ένα μη παραμετρικό τεστ που χρησιμοποιείται για τη μέτρηση του βαθμού συσχέτισης μεταξύ των πραγματικών τιμών ενέργειας που μετρήθηκαν στο target-machine και των τιμών ενέργειας που προέβλεψε το μοντέλο. Μια τέλεια συσχέτιση Spearman $+1$ ή -1 προκύπτει όταν κάθε μια από τις μεταβλητές είναι μια τέλεια μονότονη συνάρτηση της άλλης, ενώ μια συσχέτιση 0 υποδηλώνει ότι δεν υπάρχει τάση για τις μεταβλητές να είναι μια μονότονη συνάρτηση η μία της άλλης.
• **Mean Squared Error (MSE)**. Το MSE είναι ένα απόλυτο μέτρο της καλής προσαρμογής, καθώς υπολογίζεται από το άθροισμα των τετραγώνων των σφαλμάτων πρόβλεψης και στη συνέχεια διαιρείται με τον αριθμό των προβλέψεων. Δίνει έναν απόλυτο αριθμό σχετικά με το πόσο τα προβλεπόμενα αποτελέσματα αποκλίνουν από τον πραγματικό αριθμό. Είναι πάντα μια θετική τιμή που μειώνεται καθώς το σφάλμα πλησιάζει το 0.
• **Mean Absolute Error (MAE)**. Το MAE είναι μια απλή μετρική που υπολογίζει την απόλυτη διαφορά μεταξύ πραγματικών και προβλεπόμενων τιμών. Είναι ένας αριθμητικός μέσος όρος των απόλυτων σφαλμάτων για κάθε πραγματική και προβλεπόμενη τιμή και θεωρείται ανθεκτική στις ακραίες τιμές. Ο στόχος είναι να επιτευχθεί ένα ελάχιστο MAE, το οποίο ιδανικά να πλησιάζει το 0.
• **R squared (R^2)**. Το R^2 είναι μια μετρική που δείχνει την απόδοση ενός μοντέλου παλινδρόμησης και είναι επίσης γνωστή ως Coefficient of Determination (Συντελεστής Καθορισμού). Αντίθετα με τα MAE και MSE που εξαρτώνται από τις τιμές των δεδομένων, η βαθμολογία R^2 είναι ανεξάρτητη από αυτές. Έτσι, με το R^2 έχουμε ένα baseline για να συγκρίνουμε άλλα μοντέλα, δυνατότητα την οποία δεν παρέχει καμία από τις άλλες μετρικές. Η τιμή του R^2 κυμαίνεται μεταξύ 0 και 1 και μια μεγαλύτερη τιμή υποδηλώνει καλύτερη προσαρμογή μεταξύ πρόβλεψης και πραγματικής τιμής.

Επιπλέον, η ακρίβεια της πρόβλεψης γίνεται πλήρως κατανοητή μέσω γραφημάτων. Τα διαγράμματα σύκρισης των πραγματικήών έναντι των προβλεπόμενων τιμών αποτελούν μια από τις πλουσιότερες μορφές οπτικοποίησης δεδομένων. Οι πραγματικές τιμές ενέργειας, τοποθετούνται στον άξονα x, ενώ οι προβλεπόμενες από τη μηχανική μάθηση τιμές ενέργειας τοποθετούνται στον άξονα y. Οι τιμές στους άξονες x και y έχουν μονάδα μέτρησης 1 Joule. Τα σημεία στη γραφική παράσταση θα σχηματίζουν ιδανικά την ευθεία $y = x$, η οποία φαίνεται εξασθενημένη με μπλε χρώμα. Τα σημεία που προκύπτουν επισημαίνονται με κόκκινο χρώμα.

Τέλος, καθώς παρακάτω αξιολογούμε αυτό το μοντέλο μέσω 56 benchmarks, υπάρχει ένα διάγραμμα που συγκρίνει τις πραγματικές και τις προβλεπόμενες ενέργειες για κάθε benchmark ξεχωριστά, αναφέροντας το όνομα κάθε προγράμματος. Αυτά τοποθετούνται στον άξονα x, ενώ οι προβλεπόμενες τιμές ενέργειας υψώνονται κατά τον άξονα y. Δύο μπάρες που αναφέρονται στο ίδιο benchmark θα φτάσουν ιδανικά στο ίδιο ύψος.

Αξιολόγηση μέσω του Dataset

Σε αυτό το σημείο, το σύνολο δεδομένων που περιλαμβάνει τους 700 παραγόμενους κωδικούς χωρίζεται σε κλίμακα 1/5. Αυτό σημαίνει ότι οι κώδικες που χρησιμοποιούνται για την εκπαίδευση του μοντέλου είναι 560 και οι κώδικες που χρησιμοποιούνται για τη δοκιμή είναι 140.

Οι τέσσερις μετρικές που δείχνουν την ποιότητα των προβλέψεων του μοντέλου μας παρουσιάζονται παρακάτω:

```
Spearman coefficient between true and predicted is: 0.98250
Mean Squared Error (MSE): 0.417378
Mean Absolute Error (MAE): 0.192683
R squared (R^2): 0.988932
```

Σχήμα 7: Μετρικές Αξιολόγησης Μοντέλου

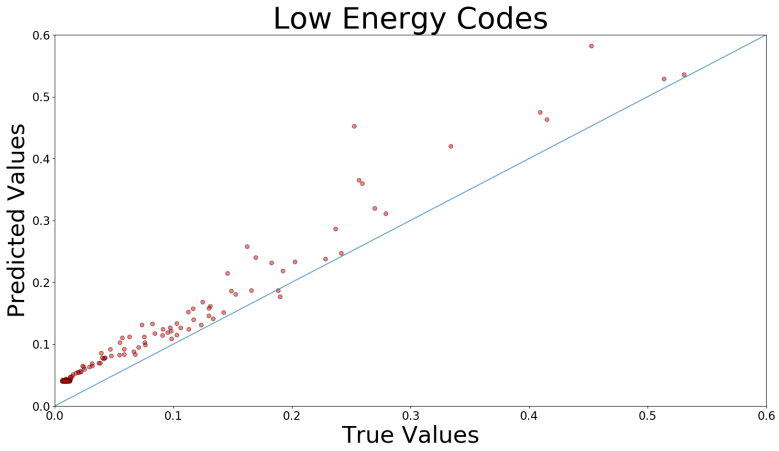
Τα αποτελέσματα δείχνουν πολύ καλή ακρίβεια στις προβλέψεις, ιδίως όσον αφορά το R^2. Αυτό ήταν αναμενόμενο, καθώς το μοντέλο προβλέπει τις ενεργειακές τιμές προγραμμάτων που έχουν συγκεκριμένα χαρακτηριστικά, παρόμοια με τα προγράμματα που το εκπαίδευσαν. Αναλυτικότερα:

• Η υψηλή συσχέτιση Spearman μεταξύ των πραγματικών τιμών ενέργειας και των προβλεπόμενων προσεγγίζει το 1, πράγμα που σημαίνει ότι οι πραγματικές ενέργειες είναι παρόμοιες με τις προβλεπόμενες.
• Το MSE δεν είναι πολύ χαμηλό. Αυτό πιθανώς συμβαίνει λόγω του μεγάλου test dataset που χρησιμοποιείται.
• Το MAE βρίσκεται σε καλό επίπεδο, υποδεικνύοντας ότι το άθροισμα των απόλυτων διαφορών μεταξύ των πραγματικών και των προβλεπόμενων τιμών δεν είναι μεγάλο.
• Η βαθμολογία R^2 είναι μεγάλη και δείχνει ότι οι πραγματικές και οι προβλεπόμενες τιμές ενέργειας είναι σχεδόν ταυτόσημες. Δείχνει πως το 98,89% όλων των μεταβολών της ενεργειακής κατανάλωσης των test codes καταγράφεται από το προτεινόμενο μοντέλο εκτίμησης ενέργειας.

Η σύγκριση των πραγματικών και των προβλεπόμενων τιμών ενέργειας απεικονίζεται στο παρακάτω σχήμα. Οι τιμές ενέργειας στον άξονα x και στον άξονα y κυμαίνονται μεταξύ [0, 8] Joules.

Σχήμα 8: Πραγματικές Τιμές έναντι Προβλεπόμενων Τιμών

Όπως βλέπουμε υπάρχουν πολλά σημεία συγκεντρωμένα στην αρχή των αξόνων. Για το λόγο αυτό, στο επόμενο σχήμα οι άξονες x και y κυμαίνονται μεταξύ [0, 0.6] Joules, εστιάζοντας στους κώδικες με χαμηλή κατανάλωση ενέργειας.



Σχήμα 9: Πραγματικές Τιμές έναντι Προβλεπόμενων Τιμών σε Κώδικες Χαμηλής Ενέργειας

Είναι προφανές ότι το R^2 0,9889 επαληθεύεται μέσω αυτών των γραφημάτων. Τα περισσότερα σημεία συγκεντρώνονται πολύ κοντά στην ιδανική διαγώνια γραμμή, επιβεβαιώνοντας το υψηλό επίπεδο ακρίβειας πρόβλεψης.

Αξιολόγηση μέσω του Polybench Suite

Το πιο σημαντικό στάδιο της αξιολόγησης του μοντέλου είναι η εφαρμογή 56 βενσημαρκς της σουίτας Πολψβενση στο μοντέλο μας, προκειμένου να εξεταστεί η ποιότητα των προβλέψεων σε μη παραγόμενους κώδικες.

Οι τέσσερις μετρικές εξετάστηκαν στο μοντέλο μας και τα αποτελέσματα παρουσιάζονται παρακάτω:

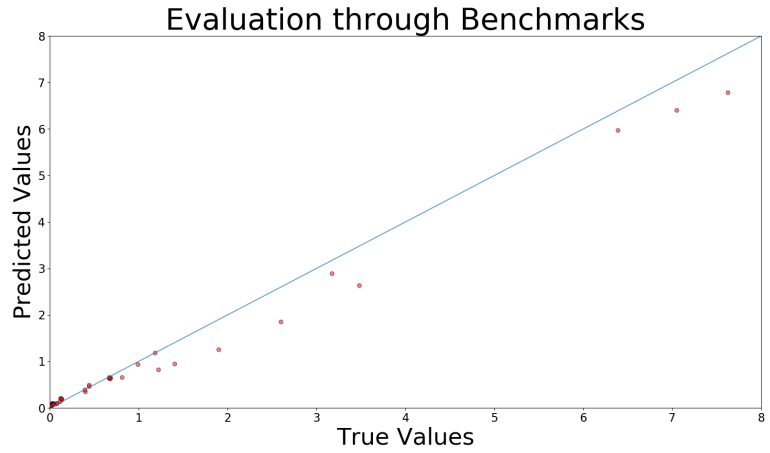
```
Spearman coefficient between true and predicted is: 0.98551
Mean Squared Error (MSE): 0.152317
Mean Absolute Error (MAE): 0.170158
R squared (R^2): 0.960387
```

Σχήμα 10: Μετρικές Αξιολόγησης Μοντέλου

Όπως γίνεται εύκολα αντιληπτό, τα αποτελέσματα ανταποκρίνονται στο σκοπό του παρόντος έργου. Πιο αναλυτικά:

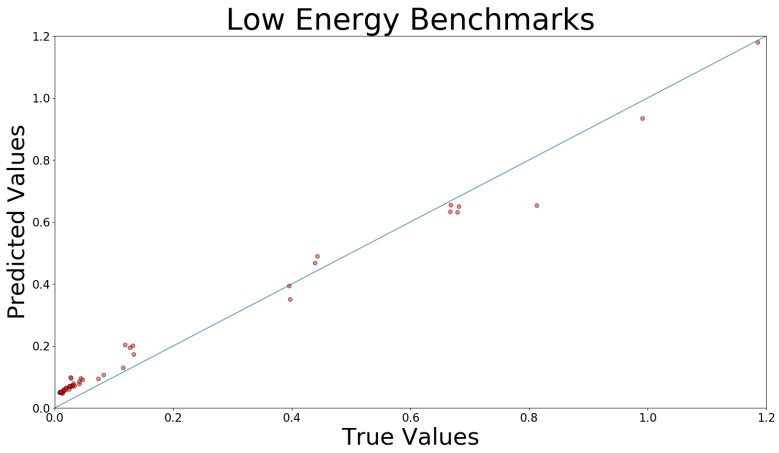
• Η υψηλή συσχέτιση Spearman μεταξύ των πραγματικών και των προβλεπόμενων τιμών ενέργειας πλησιάζει το 1, πράγμα που σημαίνει ότι οι πραγματικές τιμές ενέργειας των 56 benchmarks που μετρήθηκαν είναι παρόμοιες με τις αντίστοιχες τιμές ενέργειας που προέβλεψε το μοντέλο.
• Το MSE που πλησιάζει το μηδέν δηλώνει ότι δεν υπάρχουν σοβαρά μεγάλα σφάλματα μεταξύ των πραγματικών και των προβλεπόμενων τιμών.
• Το MAE βρίσκεται πολύ κοντά στην ίδια τάξη, υποδεικνύοντας ότι το άθροισμα των απόλυτων διαφορών μεταξύ των πραγματικών και των προβλεπόμενων τιμών είναι σε χαμηλά επίπεδα.
• Η βαθμολογία R^2 είναι μεγάλη και δείχνει ότι οι πραγματικές και οι προβλεπόμενες τιμές ενέργειας είναι σχεδόν ταυτόσημες. Αυτό σημαίνει ότι το 96,03% όλων των μεταβολών της κατανάλωσης ενέργειας των δοκιμαστικών εφαρμογών καταγράφεται από το προτεινόμενο μοντέλο εκτίμησης ενέργειας. Στην πραγματικότητα, το επίπεδο 0,960387 μπορεί να αποτελέσει την baseline τιμή αυτού του μοντέλου μηχανικής μάθησης, δηλαδή ένα σκορ που δημιουργεί κίνητρο για να ξεπεραστεί στο μέλλον.

Η σύγκριση των πραγματικών και των προβλεπόμενων τιμών ενέργειας απεικονίζεται στο παρακάτω σχήμα. Οι τιμές ενέργειας στον άξονα x και στον άξονα y κυμαίνονται μεταξύ [0, 8] Joules.



Σχήμα 11: Πραγματικές Τιμές έναντι Προβλεπόμενων Τιμών

Όπως βλέπουμε υπάρχουν πολλά σημεία συγκεντρωμένα στην αρχή των αξόνων. Αυτό συμβαίνει επειδή έχουν γίνει πολλές δοκιμές κωδικών με χαμηλή κατανάλωση ενέργειας. Για το λόγο αυτό, στο επόμενο σχήμα οι άξονες x και y κυμαίνονται μεταξύ $[0,\ 1.2]$ Joules, εστιάζοντας στους κώδικες με χαμηλή κατανάλωση ενέργειας.

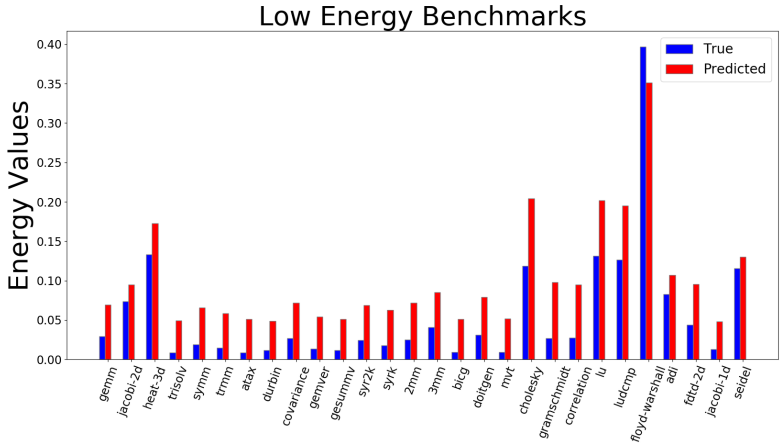


Σχήμα 12: Πραγματικές Τιμές έναντι Προβλεπόμενων Τιμών σε Κώδικες Χαμηλής Ενέργειας

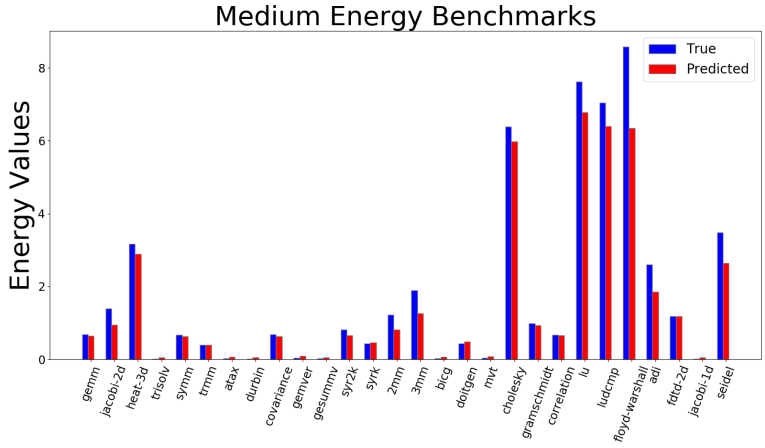
Είναι προφανές ότι το R^2 επαληθεύεται μέσω αυτών των γραφημάτων. Τα περισσότερα σημεία συγκεντρώνονται πολύ κοντά στην ιδανική διαγώνια γραμμή,

επιβεβαιώνοντας το υψηλό επίπεδο ακρίβειας πρόβλεψης. Προβληματίζει το γεγονός ότι στο γράφημα, η πρόβλεψη υπερεκτιμά ως επί το πλείστον το πραγματικό αποτέλεσμα $(y > x)$ειδικά στις χαμηλές ενέργειες, αλλά όχι σε επίπεδο που να μειώνει την επιτυχία των προβλέψεων.

Αυτά τα αποτελέσματα των πρόβλεψεων μπορούν να γίνουν απολύτως κατανοητά συγκρίνοντας την πραγματική και την προβλεπόμενη ενέργεια που καταναλώνεται από κάθε μεμονωμένο benchmark. Στα ακόλουθα σχήματα παρουσιάζονται οι πραγματικές και οι προβλεπόμενες τιμές ενέργειας των benchmarks χαμηλής και μεσαίας ενέργειας ξεχωριστά. Αναφέρεται επίσης το όνομα του κάθε προγράμματος.



Σχήμα 13: Πραγματικές Τιμές έναντι Προβλεπόμενων Τιμών σε Κώδικες Χαμηλών Ενεργειακών Απαιτήσεων



Σχήμα 14: Πραγματικές Τιμές έναντι Προβλεπόμενων Τιμών σε Κώδικες Μεσαίων Ενεργειακών Απαιτήσεων

Όπως φαίνεται καθαρά, στις περισσότερες περιπτώσεις οι προβλεπόμενες τιμές ενέργειας είναι πολύ κοντά στις πραγματικές. Το ποσοστό σφάλματος στην πρόβλεψη ένεργειας των benchmarks χαμηλών ενεργειακών απαιτήσεων που αναφέραμε προηγουμένως, μπορεί να διαπιστωθεί στο Σχήμα 13. Από την άλλη πλευρά, η πρόβλεψη ενέργειας σε κώδικες που έχουν ενεργειακές απαιτήσεις στο εύρος [0.4, 1.2] είναι άκρως επιτυχής.

Συμπερασματικά, η ακρίβεια της πρόβλεψης βρίσκεται σε πολύ καλό επίπεδο και η βαθμολογία R^2 0.960387 είναι ένα απαιτητικό baseline για να ξεπεραστεί στο μέλλον.

Δοκιμή της μεθόδου σε εναλλακτική συσκευή

Ένα από τα πλεονεκτήματα του εργαλείου είναι η δυνατότητα επαναστόχευσης. Το μοντέλο εκπαιδεύτηκε μέσω των ενεργειών που μετρήθηκαν στην πλακέτα Nvidia Jetson TX1, χρησιμοποιώντας τον αισθητήρα ισχύος INA3221. Συνεπώς, οι προβλέψεις που πραγματοποιεί το μοντέλο αναφέρονται σε τιμές ενέργειας που θα καταναλώθούν αν ο χρήστης εκτελέσει την εφαρμογή του στην συγκεκριμένη αυτή πλακέτα. Παράλληλα, όμως, το εργαλείο αυτό μπορεί να προσαρμοστεί σε οποιαδήποτε άλλη συσκευή. Ένα παράδειγμα αυτής της εφαρμογής έγινε με τη χρήση της πλακέτας NVIDIA Jetson Xavier NX, η οποία χρησιμοποιήθηκε για τη μέτρηση των ενεργειών τόσο του training dataset όσο και του test dataset. Το μοντέλο δεν τροποποιήθηκε και τα αποτελέσματα παρουσιάζονται παρακάτω:

```
Spearman coefficient between true and predicted is: 0.98018
Mean Squared Error (MSE): 0.030702
Mean Absolute Error (MAE): 0.102754
R squared (R^2): 0.769088
```

Σχήμα 15: Αξιολόγηση της μεθόδου σε άλλη συσκευή

Τα αποτελέσματα αυτά δείχνουν μια πολύ υψηλή συσχέτιση μεταξύ των προβλεπόμενων και των πραγματικών τιμών, η οποία προσεγγίζει σχεδόν το 1. Εξάλλου, το αποτέλεσμα MSE 0,03 θεωρείται εξαιρετικά χαμηλό και δείχνει ότι δεν υπάρχουν σχεδόν καθόλου σφάλματα μεταξύ των πραγματικών και των προβλεπόμενων τιμών. Αυτό επιβεβαιώνεται επίσης λαμβάνοντας υπόψη την πολύ χαμηλή βαθμολογία MAE. Από την άλλη πλευρά, η βαθμολογία R^2 δεν είναι πολύ υψηλή σε σύγκριση με το 0,96 που επιτεύχθηκε με το ενεργειακό σύνολο δεδομένων της πλακέτας Nvidia Jetson TX1, αλλά το επίπεδο 0,77 θεωρείται ικανοποιητικό.

Συμπερασματικά, η δοκιμή αυτή αποδεικνύει ότι το μοντέλο αυτό μπορεί να χρησιμοποιηθεί για την ακριβή πρόβλεψη της κατανάλωσης ενέργειας σε διάφορες πλακέτες. Σε μια μελλοντική εφαρμογή, ο χρήστης θα μπορεί να ¨προσθέσει' μια μηχανή-στόχο στον κατάλογο συσκευών-στόχων και να προβλέψει το ενεργειακό κόστος του κώδικα όταν εκτελείται σε αυτή ακριβώς τη συσκευή-στόχο.

Συμπεράσματα

Ο στόχος αυτού του έργου ήταν να δημιουργηθεί ένα εύχρηστο εργαλείο που θα μπορεί να βοηθήσει έναν προγραμματιστή στην σχεδίαση κώδικα με καλύτερη ενεργειακή απόδοση. Η γνώση της ενεργειακής κατανάλωσης του προγράμματος, χωρίς τη χρήση οποιουδήποτε αισθητήρα μέτρησης ενέργειας, μπορεί να οδηγήσει σε ενεργειακά βιώσιμους τρόπους προγραμματισμού. Μόλις ο προγραμματιστής τελειώσει τη σύνταξη του κώδικα, είναι σε θέση να ελέγξει την προβλεπόμενη κατανάλωση ενέργειας και να επανασχεδιάσει τον κώδικα με πιο αποδοτικό τρόπο εφόσον χρειάζεται. Ο προγραμματιστής μπορεί επίσης να μάθει σημαντικές πληροφορίες σχετικά με τη συμπεριφορά του κώδικα μέσω του συνόλου δεδομένων που συλλέγεται και ίσως να αλλάξει το χρησιμοποιούμενο υλικό.

Παρουσιάστηκε ένα ακριβές μοντέλο εκτίμησης της ενέργειας για προγράμματα χαμηλών και μεσαίων ενεργειακών απαιτήσεων. Με τη σκιαγράφηση κυρίως της συμπεριφοράς της μνήμης του προγράμματος, το μοντέλο που παρουσιάστηκε οδηγεί σε εξαιρετικά αποτελέσματα. Είναι δομημένο με τέτοιο τρόπο, ώστε να μπορεί να χρησιμοποιηθεί ως ένα απλό εργαλείο που μπορεί να επιταχύνει τη διαδικασία εκτίμησης της ενέργειας και συνεπώς την ανάπτυξη των ενσωματωμένων συστημάτων. Το μοντέλο αναπτύχθηκε με βάση μια πλατφόρμα υλικού και η ακρίβεια της εκτίμησης για μια σειρά εφαρμογών από τη σουίτα εφαρμογών PolyBench ήταν 96,03%.

Η δυνατότητα επαναστόχευσης είναι ένα ακόμα πλεονέκτημα του προτεινόμενου μοντέλου μας. Μετρώντας την ενεργειακή κατανάλωση των 700 παραγόμενων προγραμμάτων (train dataset) σε μια διαφορετική πλατφόρμα-στόχο, είναι εύκολο να χρησιμοποιηθεί το μοντέλο -δίχως καμία τροποποίηση- για την πρόβλεψη της ενεργειακής κατανάλωσης κατά την εκτέλεση ενός προγράμματος σε αυτή τη νέα πλατφόρμα. Κατά συνέπεια, αυτό το εργαλείο θα ήταν πολύ βοηθητικό στην επιλογή του κατάλληλου hardware που ταιριάζει σε μία συγκεκριμένη εφαρμογή.

Λαμβάνοντας υπόψη ότι ένα εργαλείο εκτίμησης ενέργειας με τη δυνατότητα να προτείνει μετασχηματισμούς σχεδίασης θα μπορούσε να είναι πολύ χρήσιμο, θεωρούμε ότι ένας Δείκτης Μετασχηματισμών Βρόχου (Loop Transformations Indicator) είναι ένας πιθανός στόχος για τις μελλοντικές ερευνητικές εργασίες μας. Προς αυτή την κατεύθυνση, οι οδηγίες "#pragma scop" και "#pragma endscop" θα μπορούσαν να βοηθήσουν στην ένδειξη του 'βαρύτερου βρόχου' και στη συνέχεια στο μετασχηματισμό αυτού με έναν πιο βιώσιμο ενεργειακά τρόπο.

Τρόπος Χρήσης του Εργαλείου

Πριν ο χρήστης χρησιμοποιήσει το εργαλείο, υπάρχουν ορισμένες απαιτήσεις που πρέπει να εξασφαλίσει. Αρχικά, πρέπει να εγκαταστήσει το Valgrind και το IntelPin στον υπολογιστή του. Επιπλέον, τα paths των scripts real_test_analysis.py και predict_energy.py πρέπει να αλλάξουν προκειμένου να προσαρμοστούν στις βιβλιοθήκες του υπολογιστή του. Αφού διευθετήσει τα παραπάνω, ο χρήστης είναι έτοιμος να χρησιμοποιήσει το εργαλείο.

Η χρήση αυτού του εργαλείου είναι πολύ απλή και αποτελείται από τρία βήματα. Πρώτα, ο χρήστης πρέπει να ορίσει τον ¨επιβαρυμένο' βρόχο βάζοντας τα "#pragma scop" και "#pragma endscop" έξω από αυτόν, προκειμένου να βοηθήσει την ανάλυση. Στη συνέχεια, πρέπει να εκτελέσει το script real_test_analysis.py για το αρχείο mycode.c προκειμένου να δημιουργήσει το αρχείο dataset.csv, όπως το παρακάτω παράδειγμα:

$python real_test_analysis.py mycode

Τέλος, ο χρήστης πρέπει να εκτελέσει το script predict_energy.py για το αρχείο mycode.c για να δει την προβλεπόμενη κατανάλωση ενέργειας της εφαρμογής του, όπως στο παρακάτω παράδειγμα:

$python predict_energy.py mycode

Chapter 1

Introduction

In the past decade there has been a large increase in use of electronic systems. The vision of a world where the latest technology is in the hands of everyone has almost become a reality and the ease of use has brought speed and comfort to the user's experience. Plenty of devices such as computers, cellphones, cars, televisions, etc., are connected to the internet, so they can share data with other Internet of Things (IoT) applications, connected devices, industrial machines and more. These internet-connected devices use built-in sensors to collect data and, in some cases, act on it, so they can adapt to the user's needs. This large amount of available data lets machines train, learn and predict. Machine learning (ML) is a subset of Artificial Intelligence (AI) that provides the system with the ability to automatically learn and improve from experience rather than explicit programming. This is considered a major technological revolution that can analyze a massive amount of data, and interpret patterns and structures to enable learning, reasoning, and decision making outside of human interaction. The Internet of Things and Machine Learning promise many positive changes for health and safety, business operations, industrial performance, and global environmental issues.

At the same time, the global energy demand has reached the highest point and has become one of the main aspects of scientific research. The environmental concerns, such as global warming and local air pollution, scarcity of water resources for thermal power generation, and the limitation of depleting fossil energy resources, raise an urgent need for more efficient use of energy and the use of renewable energy sources (RESs) [1].

With the rise of IoT and ML, the implementation of them on energy chain and electrical grids is not an exemption, even more it is the driving force of world-scale changes. IoT can be employed for improving energy efficiency, increasing the share of renewable energy, and reducing environmental impacts of the energy use. Besides, ML can can improve energy generation consumption predictions, which thereby improves the stability of an energy system.

As follows, it is a necessity for industries and developers to focus on Green Software Engineering. The two big fields of sustainability and Information and

Communication Technology (ICT) are Green IT (how can we make ICT itself more sustainable) and Green by IT (how can we encourage sustainability by ICT). Taking a deeper look, software links these two areas: Regarding Green IT, there are a lot of solutions to build and use hardware in a more energy efficient way, manufacturing embedded systems that adapt to the needs of the exact application. But the debate how energy-intensive software might be is just beginning [2]. The fields of Green Software Engineering are being shown in Figure 1.1.

The purpose of this project is to use ML in order to dynamically predict the energy consumption of low and medium energy, C language programs. The definition of low and medium energy codes given below:

- *SMALL*: Around 128KB of memory. The problem should not fit within the L1 cache, but may fit L2.
- *MEDIUM*: Around 1MB of memory. The problem should not fit within the L2 cache, but may fit L3.

When executing a code in a specific machine, it is very important for the developer to know the upcoming energy cost, and thereafter design it in a more efficient way or execute it in a different machine. In a bigger scale, the knowledge of the energy consumption of codes can help industries to avoid any waste of energy resources and design applications in the direction of Energy Sustainability.

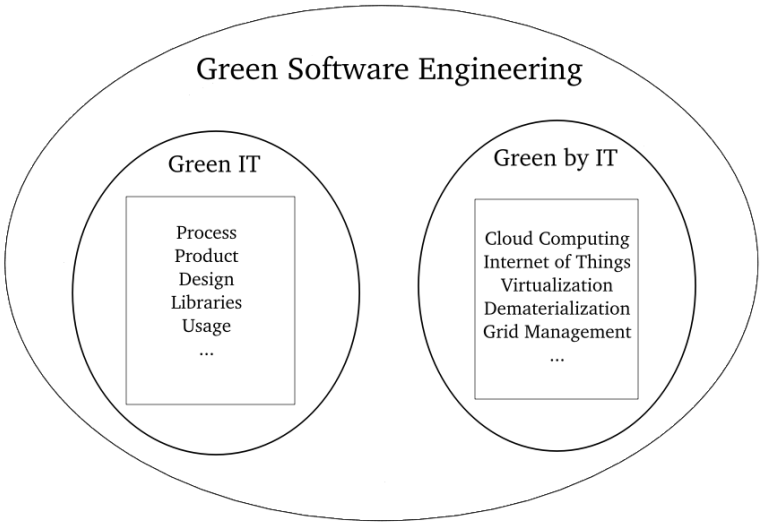


Figure 1.1: Aspects of Green Software Engineering

1.1 Related work

One of the core challenges in system designs is the necessity for fast and accurate prediction of both performance and power consumption of real world applications and benchmarks. Applications often exhibit significant power and performance variations, where estimation of time-varying performance and power traces can provide useful information for optimization of both hardware and software. Early analytical models focused on the evaluation and study of microarchitectural variations on pipeline and instruction-level parallelism. More recently, statistical and regression-based methodologies have emerged and evolved. Some of these works motivated us during the process of this Thesis.

Accurate Phase-Level Cross-Platform Power and Performance Estimation, Xinnian Zheng, Lizy K. John, Andreas Gerstlauer[3]

LACross is a a fine-grained phase-based approach, where the learning algorithm synthesizes analytical proxy models that predict the performance and power of the workload in each program phase from performance statistics obtained through hardware counter measurements on the host. During the training stage, a set of sample programs are executed both on the host machine and a reference target model. The target and the host do not necessarily have to be of similar architectures. For each workload LACross obtains, at phase level, various hardware performance features from the host as well as reference performance and power from the target. LACross formulates the problem of the latent relationship between the host and target, into a statistical learning setting, and derive prediction models for both performance and power on the target. A set of performance features is obtained at phase level and used as inputs to the prediction model in order to produce an estimate of the performance and power on the target. The tool predicts the number of executed cycles, while the prediction of the program's execution speed gives a comprehensive view of the performance. Regarding power, LACross predicts the energy consumption in relation with the runtime of the program. On the other hand, this tool necessitates 5 separate runs of the program on the Intel host to obtain 14 features, and its training dataset consists of only 157 sample programs. Despite these, this tool gave us a lot of ideas for designing the proposed methodology.

An accurate instruction-level energy estimation model and tool for embedded systems, Mostafa Bazzaz, Mohammad Salehi, Alireza Ejlali[4]

An instruction-level energy estimation model and tool for microcontrollers. It adopts a measurement-based method and it can be used to estimate the energy consumption of the processor core with good accuracy. The model is based on parameters such as instructions type, number of executed shift operations, register bank bit flips, weight and Hamming distance of the instruction words. Also, different type of memory accesses and pipeline stalls have been considered. However, it targets only on microcontrollers and it is based on a specific ARM-based Instruction Set. Furthermore, this approach uses a very limited dataset

of 60 programs.

Worst-Case Execution Time Prediction by Static Program Analysis, Reinhold Heckmann, Christian Ferdinand[5]

Abstract interpretation can be used to efficiently compute a safe approximation for all possible cache and pipeline states that can occur at a program point. These results can be combined with ILP (Integer Linear Programming) techniques to safely predict the worst-case execution time and a corresponding worst-case execution path. This approach can help to overcome the challenges listed in the previous sections, however this kind of tools are extremely slow,they are not so accurate and they can support only specific architecture models.

Energy consumption and execution time estimation of embedded system applications, Callou G, Maciel P, Tavares E, Andrade E, Nogueira B, Araujo C, Cunha P.[6]

This work proposes a mechanism for supporting design decisions on energy consumption and performance of embedded system applications. In order to depict the practical usability of the proposed methodology, a real case study as well as customized examples are presented. The estimates obtained through the conceived model are 93% close to the respective measures obtained from the real hardware platform. Although, this tool proposes a stohastic evaluation approach through discreet event simulation and it is performed in a definite microprocessor.

Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks, Charith Mendis, Alex Renda, Saman Amarasinghe, Michael Carbin[7]

Ithemal (Instruction THroughput Estimator using MAchine Learning), is a novel data–driven approach to predicting throughput for a block of instructions, inspired by advances in Deep Neural Networks (DNNs). Ithemal models the throughput estimation problem as a regression task and leverages a DNN to learn to predict throughput by using a large corpus of labeled data, mapping assembly sequences to real valued throughputs. More concretely, Ithemal uses a hierarchical multiscale Recurrent Neural Network (RNN), which generates an independent embedding for each instruction, then sequentially combines the instruction embeddings to predict throughput.

Studying those works helped us shape the idea of making a cross-platform, dynamic analysis tool that profiles an amount of 700 representative generated sample programs, trains a Lasso-built model and predicts the energy consumption of a test program.

1.2 Thesis Outline

In chapter 2 some technical mandatory background will be given in order to better understand the broader use of energy measurements embedded systems, profiling tools, regression and classification models as the two subcategories of machine learning.

In chapter 3, the proposed tool will be shown. More specifically, the way of profiling programs, building training and test datasets, and constructing the regression-based model will be analyzed extensively.

Following, in chapter 4 the experimental results will be presented. In detail, some comparisons between the feature importance, the characteristics of the model and the evaluation of the tool will be displayed.

Finally, Chapter 5 shows some conclusion drawn from the previous results, as well as some suggestions for future work.

Chapter 2

Technical and Theoretical Background

2.1 Energy measurements in embedded systems

In purpose of the energy measurements, boards and power sensors have been used. Those embedded systems are located in the Microprocessors Laboratory and Digital Systems Lab (MicroLab) laboratory of the School of Electrical and Computer Engineering of NTUA and are listed below.

Nvidia Jetson TX1

NVIDIA Jetson TX1 is an embedded system-on-module (SoM) with a GPU integrated 256-core NVIDIA Maxwell and a quad-core ARM Cortex-A57 CPU and memory of 4GB LPDDR4. Useful for deploying computer vision and deep learning, Jetson TX1 runs Linux and provides 1TFLOPS of FP16 compute performance in 10 watts of power. The block diagram of NVIDIA Jetson TX1 is shown below.

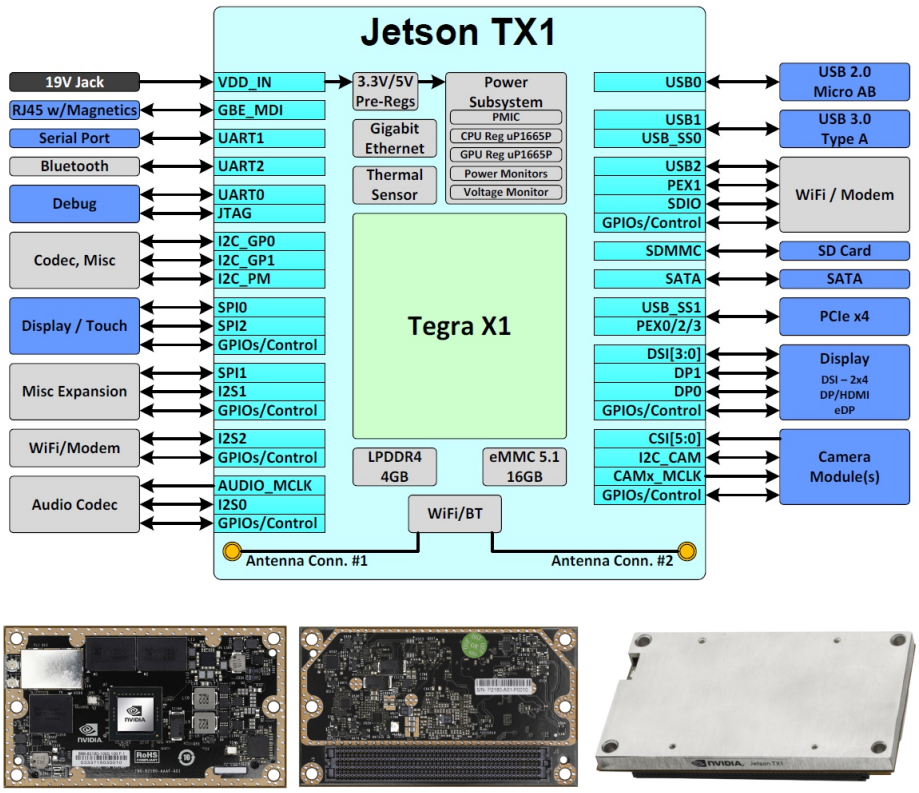


Figure 2.1: NVIDIA Jetson TX1 Block Diagram Module

NVIDIA Jetson Xavier NX

NVIDIA Jetson Xavier NX is an embedded system-on-module (SoM) with a GPU based on architecture Nvidia Volta 384 CUDA and 6-core 64-bit CPU NVIDIA Carmel ARM v8.2 . It has 2-level cache (6 MB L2 + 4 MB L3) and 8 GB 128-bit LPDDR4x RAM. It is perfect for high-performance AI systems, and accelerates the NVIDIA software stack in 10 watts of power. The block diagram of NVIDIA Jetson Xavier NX is shown below.

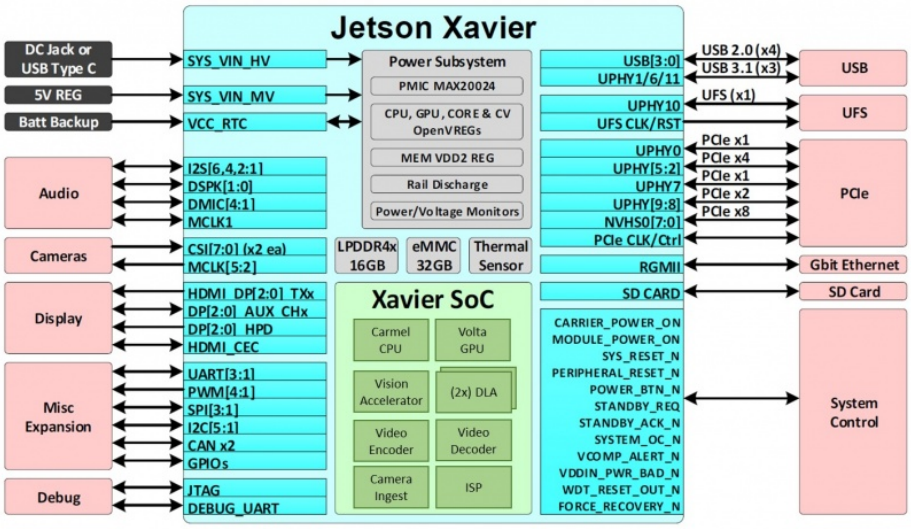


Figure 2.2: NVIDIA Xavier Block Diagram Module

INA3221 Power Sensor

The INA3221 is a three-channel, high-side current and bus voltage monitor with an I2C- and SMBUS- compatible interface. The INA3221 monitors both shunt voltage drops and bus supply voltages, in addition to having programmable conversion times and averaging modes for these signals, that allows power measurement of various applications.

2.2 Profiling Tools

As mentioned before, the purpose of this project is to predict the energy consumption of programming codes. In order to accomplish that through ML, we chose to use some profiling tools to extract useful data about the codes, especially data related to memory access. The tools that have been used are some subtools of Valgrind and Intel Pin.

Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile programs in detail. It was originally designed to be a free memory debugging tool for Linux on x86, but has since evolved to become a generic framework for creating dynamic analysis tools such as checkers and profilers. The Valgrind distribution includes seven production-quality tools: a memory error detector, two thread error detectors, a cache and branch-prediction profiler, a call-graph generating cache and branch-prediction profiler, and two different heap profilers. It also includes an experimental SimPoint basic block vector generator.

Pin is a dynamic binary instrumentation framework that enables the cre-

ation of dynamic program analysis tools. Some tools built with Pin are Intel VTune Amplifier, Intel Inspector, Intel Advisor and Intel Software Development Emulator (Intel SDE). The tools created using Pin, called Pintools, can be used to perform program analysis on user space applications on Linux, Windows and macOS. As a dynamic binary instrumentation tool, instrumentation is performed at run time on the compiled binary files. Thus, it requires no recompiling of source code and can support instrumenting programs that dynamically generate code. Pin provides a rich API that abstracts away the underlying instruction-set idiosyncrasies and allows context information such as register contents to be passed to the injected code as parameters. Pin automatically saves and restores the registers that are overwritten by the injected code so the application continues to work. Limited access to symbol and debug information is available as well. Pin was originally created as a tool for computer architecture analysis, but its flexible API and an active community (called "Pinheads") have created a diverse set of tools for security, emulation and parallel program analysis.

The specific subtools of Valgrind and IntelPin that have been used are shown below:

Cachegrind

Cachegrind simulates how the program interacts with a machine's cache hierarchy and (optionally) branch predictor. It simulates a machine with independent first-level instruction and data caches (I1 and D1), backed by a unified second-level cache (L2). This exactly matches the configuration of many modern machines.

However, some modern machines have three or four levels of cache. For these machines (in the cases where Cachegrind can auto-detect the cache configuration) Cachegrind simulates the first-level and last-level caches. The reason for this choice is that the last-level cache has the most influence on runtime, as it masks accesses to main memory. Furthermore, the L1 caches often have low associativity, so simulating them can detect cases where the code interacts badly with this cache. Therefore, Cachegrind always refers to the I1, D1 and LL (last-level) caches. Cachegrind gathers the following statistics (abbreviations used for each statistic is given in parentheses):

- I cache reads (which equals the number of instructions executed), I1 cache read misses and LL cache instruction read misses.
- D cache reads (which equals the number of memory reads), D1 cache read misses, and LL cache data read misses.
- D cache writes (which equals the number of memory writes), D1 cache write misses, and LL cache data write misses.
- Conditional branches executed and conditional branches mispredicted.
- Indirect branches executed and indirect branches mispredicted.

These statistics are presented for the entire program and for each function in the program. You can also annotate each line of source code in the program with the counts that were caused directly by it. Detailed cache and branch profiling can be very useful for understanding how your program interacts with the machine and thus how to make it faster. Also, since one instruction cache read is performed per instruction executed, it finds out how many instructions are executed per line, which can be useful for traditional profiling.

Massif

Massif is a heap profiler. It measures how much heap memory your program uses. This includes both the useful space, and the extra bytes allocated for bookkeeping and alignment purposes. It can also measure the size of the program's stack(s). Heap profiling can help in reducing the amount of memory the program uses. On modern machines with virtual memory, this provides the following benefits:

- It can speed up a program – a smaller program will interact better with the machine's caches and avoid paging.
- If a program uses lots of memory, it will reduce the chance that it exhausts your machine's swap space.

Also, there are certain space leaks that aren't detected by traditional leak-checkers. That's because the memory is not ever actually lost (a pointer remains to it) but it's not in use. Programs that have leaks like this can unnecessarily increase the amount of memory they are using over time. Massif can help identify these leaks. Importantly, Massif tells you not only how much heap memory the program is using, it also gives very detailed information that indicates which parts of the program are responsible for allocating the heap memory. Massif also provides Execution Trees memory profiling.

MICA

MICA (Microarchitecture-Independent Characterization of Applications) is a Pin tool, developed by Kenneth Hoste and Lieven Eeckhout (Ghent University), which allows the user to collect a number of program characteristics to quantify runtime program behavior. These program characteristics are totally independent of the microarchitecture (cache configuration, branch predictor, etc.) on which the measurements are done, in contrast to other workload characterization techniques using simulation or hardware performance counters. MICA tool contains eight types of analysis:

- Instruction-Level Parallellism, available for five different instruction window sizes (16, 32, 64, 128, 256).
- I-types, which is the instruction mix analysis and is evaluated by categorizing the executed instructions of x86 architecture assembly in nine categories(memory read, memory write, control flow, arithmetic, floating-point, stack, shift, string, sse)

- Instruction and data memory footprint. The size of the instruction and data memory footprint is characterized by counting the number of blocks (64-byte) and pages (4KB) touched. This is done separately for data and instruction addresses.

- Memory reuse distances. For each memory read, the corresponding 64-byte cache block is determined. For each cache block accessed, the number of unique cache blocks accessed since the last time it was referenced is determined, using a LRU stack. The reuse distances for all memory reads are reported in buckets. The first bucket is used for so called 'cold references'. The subsequent buckets capture reuse distances of $(2^n, 2^{(n+1)})$, where n ranges from 0 to 18. The first of these actually captures (0,2), while the last bucket, $(2^{18}, 2^{19})$, captures all reuse distances larger than or equal to 2^{18}. In total, this delivers 20 buckets, and the total number of memory accesses (the first number in the output), thus 21 numbers.

- Branch predictor of the conditional branches in the program, in 4 different configurations (global/local branch history, shared/seperate prediction table(s)), using 3 different history length (4,8,12 bits). Additionally, average taken and transition count are also being measured.

- The register traffic is analysis, in different aspects such as average number of register operands, average degree of use and dependency distances. Dependency distances are chosen in powers of 2 (i.e. 1, 2, 4, 8, 16, 32, 64).

- Data stream strides, that are the distances between subsequent memory accesses in four categories (local load (memory read) strides, global load (memory read) strides, local store (memory write) strides, global store (memory write) strides). Local means per static instruction accesses, global means over all instructions. The strides are characterized by powers of 8 (0, 8, 64, 512, 4096, 32768, 262144).

MICA also allows to measure the characteristics either for the entire execution, or per interval of N dynamic instructions. It also allows custom selection of which of the above types to be analyzed.

PinTool

PinTool is an IntelPin tool developed by Charalampos Marantos (ECE NTUA) that collects plenty of program characteristics. PinTool contains ten types of analysis:

- Single precision floating point operations.

- Double precision floating point operations.

- Integer operations

- Division operations.

- Control-flow operations, like if, then or else statements.
- Memory operations, such as load and store operations.
- The number of instructions executed.
- No Conflict
- Stride 0 (local and global)
- Branch divergence related features. For example, if you split all branches in sets of 16 iterations, it will calculate the number of these sets. In fact, this analysis provides sets of 16, 32, 64, 128, 512 and 1024 iterations.

2.3 Machine Learning

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. It is an important component of the growing field of data science. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so. These algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

2.3.1 Supervised Learning

The majority of practical machine learning uses supervised learning. Supervised learning (SL) is the machine learning task of learning a function that maps input variables (x) and an output variable (Y) through an algorithm with a mathematical relation Y = f(X). Techniques of Supervised Machine Learning algorithms include linear and logistic regression, multi-class classification, Decision Trees and support vector machines. Supervised learning requires that the data used to train the algorithm is already labeled with correct answers. For example, a classification algorithm will learn to identify animals after being trained on a dataset of images that are properly labeled with the species of the animal and some identifying characteristics.

Supervised learning problems can be further grouped into Regression and Classification problems. Both problems have as goal the construction of a succinct model that can predict the value of the dependent attribute from the attribute variables. The difference between the two tasks is the fact that the dependent attribute is numerical for regression and categorical for classification.

2.3.2 Regression

Regression is a supervised machine learning technique which is used to predict continuous values. The ultimate goal of the regression algorithm is to plot a best-fit line or a curve between the data, as Figure 2.3 shows. It helps in establishing a relationship among the variables by estimating how one variable affects the other. The different types of regression algorithms include:

- *Simple linear regression*
  With simple linear regression,the relationship between one independent variable and another dependent variable is estimated using a straight line, given both variables are quantitative.

- *Multiple linear regression*
  An extension of simple linear regression, multiple regression can predict the values of a dependent variable based on the values of two or more independent variables.

- *Polynomial regression*
  The main aim of polynomial regression is to model or find a nonlinear relationship between dependent and independent variables.

- *Logistic Regression*
  Logistic regression is a type of regression technique when the dependent variable is discrete. This means the target variable can have only two values, and a logistic function shows the relation between the target variable and the independent variable.

- *Ridge Regression*
  It is usually used when there is a high correlation between the parameters. This is because as the correlation increases the least square estimates give unbiased values. It is a powerful regression method where the model is less susceptible to overfitting.

- *Lasso Regression*
  Lasso Regression performs regularization along with feature selection. It avoids the absolute size of the regression coefficient. This results in the coefficient value getting nearer to zero, this property is different from what in ridge regression.This helps avoid the overfitting in the model. But if independent variables are highly collinear, then Lasso regression chooses only one variable and makes other variables reduce to zero.

- *Bayesian Linear Regression*
  In Bayesian linear regression, the posterior distribution of the features is determined instead of finding the least-squares. Bayesian Linear Regression is a combination of Linear Regression and Ridge Regression but is more stable than simple Linear Regression.

- *Decision Tree Regression*
  It has mainly attributed that include internal nodes, branches, and a terminal node.Every internal node holds a "test" on an attribute, branches hold the conclusion of the test and every leaf node means the class label.

- *Random Forest Regression*
  Random forest, as its name suggests, comprises an enormous amount of individual decision trees that work as a group or as they say, an ensemble. Every individual decision tree in the random forest lets out a class prediction and the class with the most votes is considered as the model's prediction.

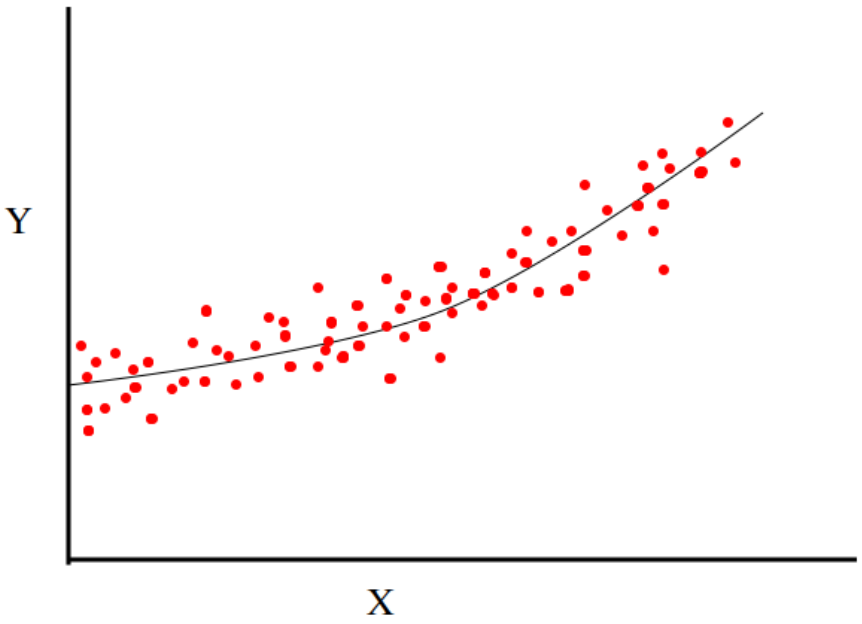


Figure 2.3: Regression Model

2.3.3 Classification

Classification is a predictive model that approximates a mapping function from input variables to identify discrete output variables, which can be labels or categories. The mapping function of classification algorithms is responsible for predicting the label or category of the given input variables. A classification algorithm can have both discrete and real-valued variables, but it requires that

the examples be classified into one of two or more classes. The different types of classification algorithms include:

- *Logistic regression*
  In this algorithm, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.

- *Naive Bayes*
  Based on Bayes' theorem with the assumption of independence between every pair of features. Naive Bayes classifiers work well in many real-world situations such as document classification and spam filtering.

- *Stochastic Gradient Descent*
  A simple and very efficient approach to fit linear models. It is particularly useful when the number of samples is very large. It supports different loss functions and penalties for classification.

- *K-Nearest Neighbours*
  A type of lazy learning as it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the k nearest neighbours of each point.

- *Decision Tree*
  Given a data of attributes together with its classes, a decision tree produces a sequence of rules that can be used to classify the data.

- *Random Forest*
  Random forest classifier is a meta-estimator that fits a number of decision trees on various sub-samples of datasets and uses average to improve the predictive accuracy of the model and controls over-fitting.

- *Support Vector Machine*
  Is a representation of the training data as points in space separated into categories by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

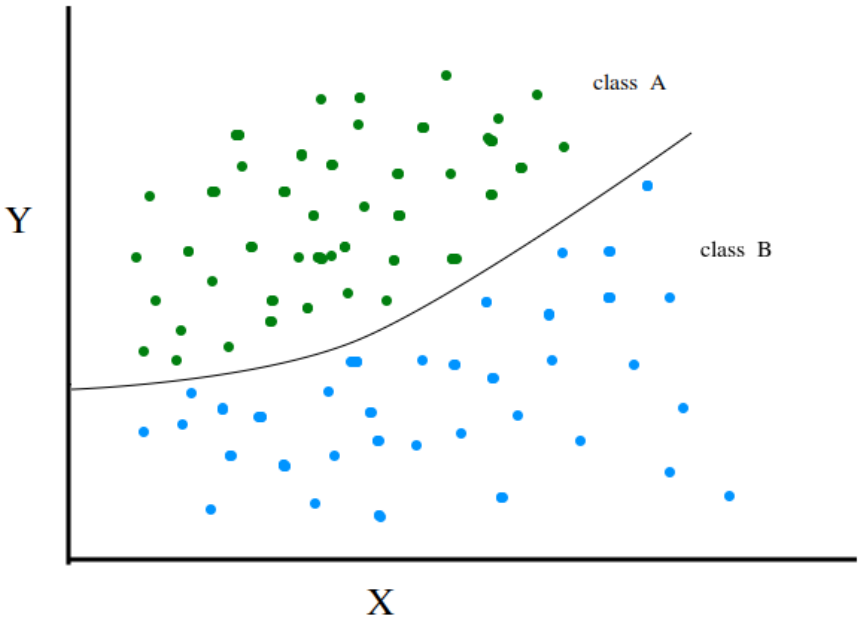


Figure 2.4: Classification Model

Chapter 3

Proposed Tool

3.1 Overall Methodology

In this chapter, every stage of the methodology will be presented. This project begins with a Code Generator that produces 700 individual C-files under certain specifications. Then, a script profiles these C codes and extracts valuable information while it includes them in a csv file called dataset. Alongside, the 700 codes are been measured and collected as in terms of energy consumption on embedded platform, as a second file named dataset_energy is been illustrated. By these datasets, the model will get trained. Next, a Lasso model Regressor is been constructed and its parameters are a subject of optimization. Likewise, 56 C-language benchmarks from PolyBench Suite are been profiled just like the 700 code files and a X test_dataset is been made. After all these steps, we are ready to use the model in order to predict the energy consumption of these 56 codes. Finally, the test codes are been also measured on the board in order to compare them with the predicted energies and evaluate the accuracy of the tool's prediction. The following schematic 3.1 recapitulates the overall methodology. The details of every step will be the focal point in the next sections.

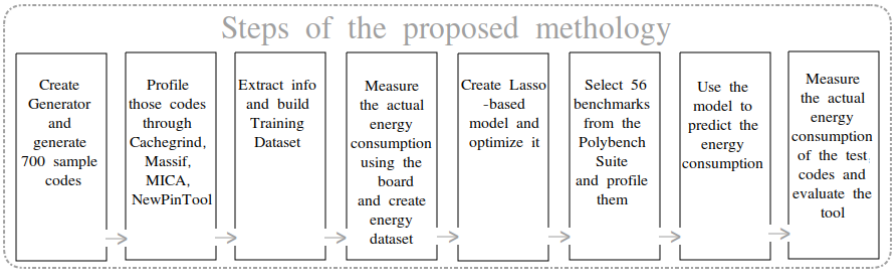


Figure 3.1: Overall Methodology

3.2 C Code Generator

In order to create a large enough training dataset we created a script that generates C-language codes with the characteristic of nested loops existence. This type of codes is concerned to have a particular interest, as the energy consumption during the nested loops gets extremely high. This happens, because of the large number of jumps of the program counter (PC), as well as the not optimal fill of the memory caches. Avoidable memory accesses and cache misses cause low performance and high, unnecessary energy effort. Embedded Systems analysis has focused on impugning this waste of energy resources, by creating specific technical tricks like loop transformations. Ideas of contributing in this aspect, by this tool, are given in the Future Work section.

The codes begin with the initialization of a maximum of 4 integer arrays and a maximum of 4 float point arrays, that have a maximum of 6 dimensions and every dimension contains a maximum of 1000 elements. These characteristics put these C codes inside the frame of Small & Medium weighted codes. This type of codes is the one that this tool will learn to estimate energy-wise.

As the initialization completes, the arrays above receive randomized values, according to each array's variables type and the proper memory volume is been allocated. Every array's element has a value and the code's most burdened section is ready to begin.

Then, the nested loops part takes over. Firstly, the directive "#pragma scop" is been placed before the first loop, in order to inform the analysis script that the section begins. Inside the nested loops, a computation part is included and the types of computations (addition, subtraction, multiplication, division) are randomly determined. After closing all the loops a second directive "#pragma endscop" is being placed under the last bracket. That informs the analysis script that the loop section is finished. An example of the loop typification is given below:

```
#pragma scop
for(i=0; i<18; i++){
    for(j=0; j<150; j++){
        for(k=0; k<87; k++){
            A0[i][j][k] = A1[i][j][k]*A1[i][j][k]/A1[i][j][k];
            B0[i][j][k] = B1[i][j][k]-B1[i][j][k]/B1[i][j][k]*B1[i][j][k];
        }
    }
}
#pragma endscop
```

The generated code ends with a final section of freeing the used memory space and then returning 0, informing that the execution completes properly.

3.3 Profiling

A python script named "total_analysis.py" is responsible for profiling the 700 codes. It begins by running five commands that are shown below:

- **gcc -g code.c -o code**
  By this command, GCC compiler is called and it compiles the source code while producing the executive file. The -g flag is used for debbugging info

- **valgrind –tool=cachegrind –branch-sim=yes –cachegrind-out-file=code.cachegrind ./code**
  This command calls Valgrind's tool Cachegrind. This tool simulates the first-level and last-level caches, and with the special option enabled, it collects of branch instruction and misprediction counts. An output file named code.cachegrind is produced and it contains Ir (I cache reads), I1mr (I1 cache read misses), ILmr (LL cache instruction read misses), Dr (D cache reads), D1mr (D1 cache read misses), DLmr (LL cache data read misses), Dw (D cache writes), D1mw (D1 cache write misses), DLmw (LL cache data write misses), Bc (conditional branches executed), Bcm (conditional branches mispredicted), Bi (indirect branches executed) and Bim (indirect branches mispredicted) for the whole program.

- **valgrind –tool=massif –stacks=yes –massif-out-file=code.massif ./code**
  With this command another Valgrind's tool Massif is called. It measures how much heap memory your program uses. This includes both the useful space, and the extra bytes allocated for book-keeping and alignment purposes. It also measures the size of your program's stack. An output file named code.massif is produced and it contains mem_stacks_B (size of the program's stack), mem_heap_extra_B (extra heap bytes allocated), mem_heap_B (useful heap bytes allocated) and the time it was taken to execute the whole program.

- **../../../pin-3.18-98332-gaebd7b1e6-gcc-linux/pin -t ../../../pin-3.18-98332-gaebd7b1e6-gcc-linux/source/tools/MICA-master/obj-intel64/mica.so – ./code**
  By this command MICA tool is called. Multiple output files (ilp_full_int_pin.out, itypes_full_int_pin.out, memfootprint_full_int_pin.out, memstackdist_full_int_pin.out, ppm_full_int_pin.out, reg_full_int_pin.out, stride_full_int_pin.out) are produced that contain ilp (Instruction-Level Parallellism for four different window sizes), itypes (instruction mix, evaluated by categorizing executed instructions like mr, mw, control flow, arithmetic etc.), memfootprint (instruction and data memory footprint), memstackdist (memory reuse distances), ppm (conditional branch predictability) and stride (distances between subsequent memory accesses).

- **../../../pin-3.18-98332-gaebd7b1e6-gcc-linux/pin -t ../../../pin-3.18-98332-gaebd7b1e6-gcc-linux/source/tools/NewPinTool/obj-intel64/all_in_one.so – ./code**
  This command calls the last tool, PinTool to collect some useful information related to CPU. It produces an output file named Pin_tool_resuls.txt and contains branch divergence rate information.

3.4 Building Datasets

The procedure of total_analysis.py script has not finished yet. Subsequently, it searches for the out-files produced by the tools to collectively gather the information they contain. According .cachegrind out-file, the script collects the metrics mentioned before for both entire program (summary) and the nested loop only. That is the phase when the #pragma directive actually helps. The script finds the position of the nested loops in the source code file, noting the numbers of the lines the nested code exists. Afterwards, it searches the .cachegrind file for that precisely line numbers and drugs the metrics counted inside this section.

Secondary, the .massif out-file gives us exported information for a snapshot. In this stage, the script manages to drag the information at the peak snapshot, which is taken during the nested loops. So, except time which refers to the whole program, the heap memory used refers to the loop nested loop section.

The out-files from MICA and PinTool provide information simulating the whole program behavior with different window sizes or different platforms. So, the total_analysis.py script extracts all those information.

Finally, all those (141 in total) useful metrics are collected for each of the 700 generated codes and the script writes them in a csv file named dataset.csv. These metrics are now the features of the X training dataset file and they are depicted as columns, while the 700 codes are depicted as rows.

Alongside, the measurement of those 700 codes energy consumption has to be done in order to create the y dataset and train the model. For this reason, another script named collect_train_codes_energy.py is been developed in the target platform. The script compiles the source file producing the executive

file just like the total_analysis.py script. Then, it runs the following command:

- **python measure_time_energy.py ./code 20**
  This measure_time_energy.py script is responsible for measuring the energy of the code. The number "20" stands for the repetition of code's execution. The code executes 20 times, and this energy is measured by the sensor. Then, this value is divided by 20 and gives us the energy cost of a single execution. This technique is implemented to achieve a better accuracy in terms of energy measurement.

Immediately after the measurement, the script writes every value to a 1-column csv file called dataset_energy.csv, that finally contains the energy consumption of the 700 codes. The codes are depicted as rows just like before, and all these real energies are now the y training dataset file.

3.5 Training model

It is important to mention that most of the functions and algorithms that follow, are part of Scikit-learn, also known as sklearn, which is a free software machine learning library for the Python programming language.

At this point, when the X and y training datasets are ready, it is time to construct the model that fits to our prediction needs, and select the best parameters to optimize the predictions.

3.5.1 Preparing the model

A new script called set_up_Regressor.py is made for implement these tasks, and its sections are being described below.

In this first step, the datasets clarified above are imported and set as X and y datasets. The X refers to all the code features and y refers to the code energies. The X list has a shape of (700,141) while the y list has a shape of (1,141).

Then, a split has to be done in order to let the set the test size in relationship with the training size. The chosen proportion is 1/5 which means that for 560 train rows, the rest 140 rows correspond to test. Now, we have made X_train, y_train, X_test and y_test. The reason of this split is to make try out many combinations and see the results in the GridSearch() algorithm that we will discuss in the next section.

As it is going to make random regressions using the splits described above, it is useful to create a pipeline that consists of two parts: the preprocessing (scaling) part and the regressor part.

At this point, it is very useful to do some scaling to the X dataset in order to help the regressor. It is more difficult for it to manage values with large variety, so we chose to use the StandardScaler() to reduce the width of the feature values. StandardScaler() standardizes a feature by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by

the standard deviation. StandardScaler() makes the mean of the distribution 0, and finally most of the values will lie be between -1 and 1.

The selection of this specific regression algorithm will be a matter of discussion in the next section. This pipeline will scale every partition of the dataset before it applies to the GridSearch() function.

3.5.2 Lasso Regression Model

During the stonework of this project, many regression algorithms were tested. Every prediction problem has a suitable algorithm that fits to the problem well. So, a theoretical research can direct the developer to the perfect model, but it has to be a matter of testing in order to choose properly the best model. The criterion according to which we compared the models was the Mean Squared Error (MSE) of the prediction made by splitting the dataset as described above. Many algorithms were tested and the results are been shown in 3.2.

Regression Model	Mean Squared Error
Ridge	2.3933
Lasso	0.5310
Random Forest	37.223
Stohastic Gradient Descent	∞
Bayesian Ridge	1.3658
K-Nearest Neighbors	40.239
Gradient Boosting	39.317
Decision Tree	39.743

Figure 3.2: Regression models comparison

The obvious result of these tests is that the Lasso Regressor suites to our project optimally. That was primarily expected, as the dataset we use is large enough in terms of features and these features are strongly related to each other.

Lasso stands for Least Absolute Selection Shrinkage Operator wherein shrinkage is defined as a constraint on parameters. The goal of lasso regression is to obtain the subset of predictors that minimize prediction error for a quantitative response variable. The algorithm operates by imposing a constraint on the model parameters that causes regression coefficients for some variables to shrink toward a zero.

Variables with a regression coefficient equal to zero after the shrinkage process are excluded from the model. Variables with non-zero regression coefficients variables are most strongly associated with the response variable. Explanatory variables can be either quantitative, categorical or both. This lasso regression

analysis is basically a shrinkage and variable selection method and it helps analysts to determine which of the predictors are most important.

Lasso regression algorithm is highly preferred when there is high multicollinearity in the given dataset. Multicollinearity in the dataset means independent variables are highly related to each other, and a small change in the data can cause a large change in the regression coefficients. Our dataset has high multicollinearity, as it has strong correlation between the independent variables, and this is the main reason this algorithms gives the best results for this application.

The process of shrinkage is performed through L1-regularization, wherein it penalizes the number of features in a model in order to only keep the most important features, using a parameter, alpha. When alpha is 0, Lasso regression produces the same coefficients as a linear regression. When alpha is very large, all coefficients are zero.

3.5.3 Grid Search and Cross-Validation

Grid Search is a technique to select the best of the machine learning model, parameterized by a grid of hyperparameters. It tries all combinations of parameters grid for a model and returns with the best set of parameters having the best performance score. Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations. Especially, k-fold cross-validation is a type of cross-validation, where the training data is split into k-folds and (k-1) folds is used for training and k-th fold is used for validation of the model.

This process begins by defining the random grid of the parameters. Except alpha, which is the most important parameter as mentioned before, we enriched the random grid with two more parameters. The maximum number of iterations is specified by parameter max_iter, while the tolerance for the optimization is specified by tol.

After creating the random grid, the Repeated K Fold technique is chosen to make the cross-validation more valid. The RepeatedKFold() function made 10 splits and 20 repeats to the dataset, which means 200 different cases. These cases were evaluated through the Negative Mean Squared Error (NMSE) scoring function. After fitting X and y data to the grid, we get the following results:

- Negative mean squared error of the best_estimator: -1.5002
- Parameter settings that gave the best results: 'regressor__alpha': 0.3, 'regressor__max_iter': 200, 'regressor__tol': 0.01

These parameters will be applied to the Lasso regressor and form this Machine Learning model.

Chapter 4

Experimental Results

After finding the best parameters for the Lasso-based model, a new script called Predict_energy.py was created. In this script we are going through some feature analysis tests, in order to draw conclusions about the importance of the training dataset's features, as well as evaluate the model using real test benchmarks taken from Polybench, a collection of benchmarks that is considered as representative for embedded applications. These benchmark was chosen because it provides the developer the option of compiling in different sizes of datasets (different sizes of arrays). For this project we chose to use the small and medium datasets for the 28 following benchmarks. This choice was made to define the benchmarks as low and medium energy demand programs. This means that the test dataset consists of 28x2=56 codes.

Benchmark	Description
gemm	Matrix-multiply C=alpha.A.B+beta.C
jacobi-2D	2-D Jacobi stencil computation
heat-3d	3-D Heat stencil computation
trisolv	Triangular solver
symm	Symmetric matrix-multiply
trmm	Triangular matrix-multiply
atax	Matrix Transpose and Vector Multiplication
durbin	Toeplitz system solver
covariance	Covariance Computation
gemver	Vector Multiplication and Matrix Addition
gesummv	Scalar, Vector and Matrix Multiplication
syr2k	Symmetric rank-2k operations
syrk	Symmetric rank-k operations
2mm	2 Matrix Multiplications (D=A.B; E=C.D)
3mm	3 Matrix Multiplications (E=A.B; F=C.D; G=E.F)
bicg	BiCG Sub Kernel of BiCGStab Linear Solver
doitgen	Multiresolution analysis kernel (MADNESS)
mvt	Matrix Vector Product and Transpose
cholesky	Cholesky Decomposition
gramschmidt	Gram-Schmidt decomposition
correlation	Correlation Computation
lu	LU decomposition
ludcmp	LU decomposition
floyd-warshall	Floyd–Warshall algorithm
adi	Alternating Direction Implicit solver
fdtd-2d	2-D Finite Different Time Domain Kernel
jacobi-1D	1-D Jacobi stencil computation
seidel	2-D Seidel stencil computation

Figure 4.1: List of Benchmarks

These 56 benchmarks go through the "real_test_analysis.py" script. This script is identical to the "total_analysis.py" script, that previously presented in Section 3.3. By this procedure, the profiling tools collect all the data and create the X test dataset. Alongside, the measurement of those 56 test codes energy consumption has to be done in order to compare the real test energies with the predicted ones. For this reason, the script named collect_energy.py is run in the target platform. This script is similar to the collect_train_codes_energy.py script, that was presented in Section 3.4 and creates the y test dataset.

The Predict_energy.py script starts by importing the train and test datasets. The train dataset contains the 700 generated codes and the test dataset contains the 56 benchmarks described above.

4.1 Feature Analysis

The next section of this script is the Feature Correlation section. In this section we focused on the importance the Lasso Regressor gives to the features, as well as the Spearman's correlation between the features and the energy. The reason why we chose to put many features in the dataset was not only to give enough information to the model -since Lasso Regressor does a feature selection and does not manage all these 141 features-, but also to move on to some analysis on the importance of the metrics according to energy consumption.

4.1.1 Feature Importance

As mentioned before, Lasso Regressor applies a weight coefficient to the features. Most of the features are shrunk to zero, and the rest of the features have a weight that is called importance. The feature analysis begins by finding the features which have non zero importance and make the regressor achieve the optimum results. The list Lasso Regressor decided to keep are shown in Figure 4.2.

```
---------find the important Lasso coefficients------------------------
Feature: D1mr_total has Lasso non zero importance: 0.05392
Feature: DLmr_total has Lasso non zero importance: 0.11928
Feature: Dw_total has Lasso non zero importance: 0.68814
Feature: Bc_total has Lasso non zero importance: 0.18763
Feature: Bcm_total has Lasso non zero importance: 0.18580
Feature: mem_heap_B has Lasso non zero importance: 0.37906
Feature: ilp has Lasso non zero importance: 2.48196
Feature: ilp.1 has Lasso non zero importance: 0.00165
Feature: ilp.2 has Lasso non zero importance: 0.03192
Feature: ilp.3 has Lasso non zero importance: 0.04219
Feature: ilp.4 has Lasso non zero importance: 0.02634
Feature: itypes has Lasso non zero importance: 0.01440
Feature: itypes.3 has Lasso non zero importance: 0.32430
Feature: itypes.5 has Lasso non zero importance: 0.09939
Feature: memfootprint has Lasso non zero importance: 0.43879
Feature: memfootprint.1 has Lasso non zero importance: 0.33088
Feature: stride.1 has Lasso non zero importance: 0.24014
```

Figure 4.2: Lasso feature importance

As it seems, the most weighted features are:

- D1mr_total, DLmr_total and Dw_total refer to the reads and writes of the whole program, and these memory operations are too heavy energy-wise. Especially the Data writes seem to have much effect in the code 's energy consumption.
- Bc_total and Bcm_total correspond to the conditional branches executed and conditional branches mispredicted of the whole program. The program counter stores the memory address of the next instruction to be

executed and when a branch is taken, the CPU's program counter is set to the argument of the jump instruction, which is extremely costly.

- mem_heap_B which refers to heap memory, which is a memory shared by all executing threads in the application. This memory significantly impacts the power and energy utilization of the overall server system.

- ilp, ilp.1, ilp.2, ilp.3 and ilp.4 which correspond to the parallel or simultaneous execution of a sequence of instructions in five different instruction windows (16 size window seems to be by far the most influential feature). Parallel Computing leads to performance improvements, but the exchanging data at runtime through shared memory regions lead to higher power consumption.

- itypes, itypes.3 and itypes.5 correspond to the number of memory reads, arithmetic calculations and stack usage. Arithmetic calculations are a heavy task that is managed by the CPU's arithmetic/logic unit (ALU) and often efforts a lot of energy, especially in case of division. The usage of the stack seems to be also important for the regressor, as a proper memory allocation leads to lower energy consumption.

- memfootprint and memfootprint.1 refer to the number of blocks (64-byte) and pages (4KB) touched by data addresses. The amount of main memory that a program uses or references while running is a key factor for Lasso Regressor.

- stride.1 corresponds to the memory reuse distances and briefly refers to the necessary cache memory changes, which are very important as the longest these distances are, the heaviest the program ends up being.

4.1.2 Feature Correlation

This Lasso Regressor feature selection is theoretically considered reasonable. But the feature analysis contains another critical part, the correlation test. For this purpose the Spearman's correlation is applied, as it evaluates the monotonic relationship between two continuous or ordinal variables. This procedure calculates a coefficient for each feature, that varies between -1 and +1 with 0 implying no correlation. Correlations of -1 or +1 imply an exact monotonic relationship. Positive correlations imply that as x increases, so does y. Negative correlations imply that as x increases, y decreases. The results of this test are been shown below in Figure 4.3.

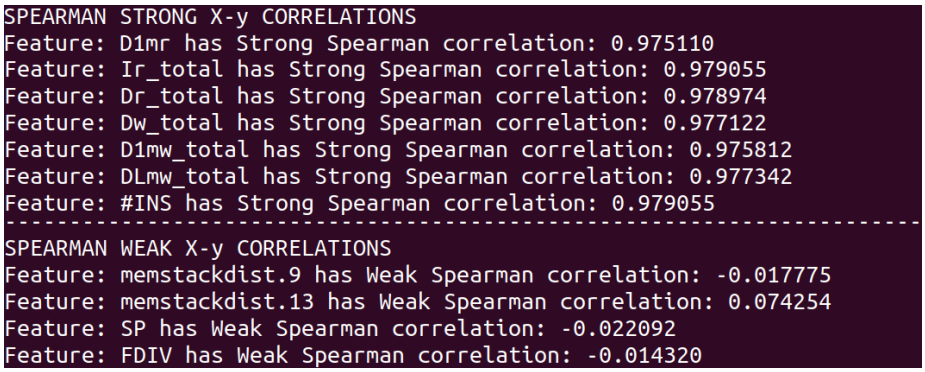


Figure 4.3: Strong and Weak Correlations between Features and Energy

As follows the strongly correlated features are:

- D1mr, which are D1 cache read misses of the loop section. This means that the read misses that happen inside the nested loops are very effective according to energy.
- Ir_total and #INS, which are both measuring the number of instructions executed and are the most correlated features to the energy consumption, as it practically describes the size of the executed program.
- Dr_total, which corresponds to the number of memory reads of the entire program. A memory read operation transfers the desired word to address lines and activates the read control line. As this number goes high, the energy requirements increase.
- Dw_total, which equals number of memory writes of the whole program. A memory write operation transfers the address of the desired word to the address lines, transfers the data bits to be stored in memory to the data input lines. This procedure has high effectiveness to the energy consumption.
- D1mw_total and DLmw_total, which correspond to the D1 cache and LL cache write misses. When a write miss happens,it requires the application to make a second attempt to locate the data, this time against the slower main database and this procedure is highly costly according to energy.

These results highlight the significance of the cache memory operations when it comes to energy consumption. As follows, it is extremely important to implement embedded systems with cache specifications that adapt to the application (e.g. multiple cache levels) and design the code in a way that reduces the cache misses in order to achieve energy efficiency.

On the other side, the weakly correlated features are:

- memstackdist, which refers to the memory reuse distances. For each memory read, the corresponding 64-byte cache block is determined and for each cache block accessed, the number of unique cache blocks accessed since the last time it was referenced is determined, using a LRU stack. This is a very detailed metric that may help in further analysis, but not that much in terms of energy consumption.
- SP, which is the number of single precision floating point operations. This metric is logically weakly correlated, as it counts a very specific type of arithmetic operations.
- FDIV, which refers to the division operations (floating point) and is not so correlative to the energy consumption, due to similar reasons with SP.

4.2 Model Implementation

By the time the theoretical analysis is done, and the X and y datasets are imported, it is time for the preprocessing part. The X train dataset goes through the scaling process. For this task, the StandardScaler() is used to reduce the width of the feature values. StandardScaler() standardizes a feature by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by the standard deviation. StandardScaler() makes the mean of the distribution 0, and finally most of the values will lie be between -1 and 1.

Afterwards, we import the Lasso Regressor and define it with the parameters resulted by the set_up_Regressor.py script that was shown in the previous chapter. The resulting parameters are also reminded below:

- Parameter settings: 'regressor__alpha': 0.3, 'regressor__max_iter': 200, 'regressor__tol': 0.01

Then, the X train scaled and the y train datasets are fitted to the regressor. Our model is now ready to be evaluated through the real benchmarks. Firstly, the X test dataset gets transformed by the scaler, as the model was trained through scaled data. Then, the regressor uses this X test scaled dataset to predict the benchmarks' energies. The y pred dataset, that includes the predicted energy values, is created and our model is ready for evaluation.

4.3 Evaluation

Accuracy in a regression model is slightly hard to illustrate. It is impossible for a model to predict the exact value but it is appropriate to grade how close the prediction is against the real value. For grading out the performance of a regression model, there are various metrics that evaluate the accuracy in a regression model. The evaluation metrics that were chosen for this project are shown below:

- **Spearman correlation between true and predicted values** Spearman correlation is a non-parametric test that is used to measure the degree of association between the true energy values that were measured in the target machine, and the energy values that the model predicted. A perfect Spearman correlation of $+1$ or -1 occurs when each of the variables is a perfect monotone function of the other, when a correlation of 0 indicates that there is no tendency for the variables to be a monotone function of each other.

- **Mean Squared Error (MSE)** MSE is an absolute measure of the goodness for the fit, as it gets calculated by the sum of square of prediction error which is real output minus predicted output and then divide by the number of data points. It gives an absolute number on how much the predicted results deviate from the actual number, and it is often used to compare against other model results. It is always a positive value that decreases as the error approaches 0.

- **Mean Absolute Error (MAE)** MAE is a simple metric which calculates the absolute difference between actual and predicted values. It is an arithmetic average of the absolute errors for each true and predicted values, it is considered robust to outliers. The aim is to get a minimum MAE, ideally approaching 0.

- **R squared (R^2)** R^2 score is a metric that shows the performance of a regression model and it is also known as Coefficient of Determination. Unlike MAE and MSE which depend on the data values, the R^2 score is independent of these values. Thus, with R^2 we have a baseline to compare other models, a capability that none of the other metrics provide. The value of R^2 ranges between 0 and 1, and a higher value indicates a better fit between prediction and actual value.

Additionally, the prediction accuracy is fully understood through graphs. Diagrams of actual vs predicted are one of the richest form of data visualization. They true energy values, which are measured in the lab's target machine, are set in the x axis. The machine learning predicted energy values are set in the y axis. The energy values that are set in the x and the y axis have a unit of measurement of 1 Joule. The points on the graph would identically form the function's $y = x$ graph line, which is fades with blue color. The actual points are highlighted with red color.

Finally, as in Section 4.3.2 we evaluate this model through 56 benchmarks, there is a chart that compares the actual and the predicted energies of every benchmark, mentioning the name each benchmark. These benchmarks are set in the x axis while the predicted energy values are set in the y axis. The two bars that refer to the same benchmark would identically reach the same height.

4.3.1 Evaluation through Dataset

In this step, the dataset that includes the 700 generated codes is split to a scale of 1/5. That means that the codes used for training are 560 and the codes that are used for testing are 140.

Prediction Rate Metrics

The four metrics that show the predictions quality of our model are presented below:

```
Spearman coefficient between true and predicted is: 0.98250
Mean Squared Error (MSE): 0.417378
Mean Absolute Error (MAE): 0.192683
R squared (R^2): 0.988932
```

Figure 4.4: Dataset Evaluation Metrics

The results show a very good accuracy in the predictions, especially regarding R^2. That was expected, as the model predicts the energy values of codes that have specific characteristics, similar to the codes that trained it. In more detail:

- The high Spearman correlation between the true energy values and the predicted ones is approaching 1, which means that the actual energies are facsimile to the predicted energies.
- The MSE is not very low. This probably happens because of the large test dataset that is used.
- The MAE lies in a good level, indicating that the sum of the absolute differences between actual and predicted values is at low levels.
- The R^2 score is great and shows that the true and the predicted energy values are almost identical. This means that 98.89% of all variations of the test codes' energy consumption are captured by the proposed energy estimation model.

Visual Display of the Results

The actual and the predicted energy values comparison is illustrated in 4.5. The energy values in the x axis and the y axis range between [0, 8] Joules.

Figure 4.5: True vs Predicted Energy Values

As we see there are many points concentrated at the beginning of the axes. For this reason, Figure 4.6 is illustrated below and ranges the x axis and the y axis between [0, 0.6] Joules, focusing on the low energy consumed codes.

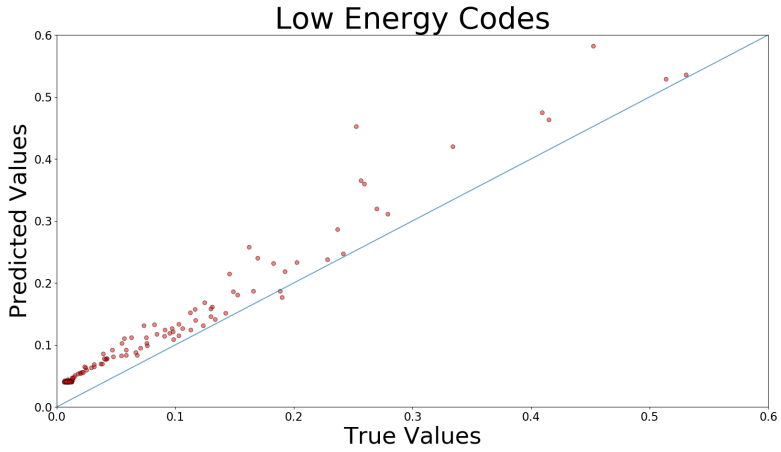


Figure 4.6: True vs Predicted Low Energy Values

It is obvious that the R^2 of 0.9889 is verified through these graphs. Most of the points are concentrated very near the ideal diagonal line, confirming the high level of prediction accuracy.

4.3.2 Evaluation through Polybench Suite

The most important stage of the model evaluation is the application of 56 benchmarks of Polybench Suite to our model in order to review the quality of the predictions in non-generated codes.

Prediction Rate Metrics

The four metrics that show the predictions quality of our model are presented below:

```
Spearman coefficient between true and predicted is: 0.98551
Mean Squared Error (MSE): 0.152317
Mean Absolute Error (MAE): 0.170158
R squared (R^2): 0.960387
```

Figure 4.7: Model Evaluation Metrics

As it easy too see, the results are justifying the purpose of this project. In more detail:

- The Spearman correlation between the true energy values and the predicted ones is approaching 1, which means that the measured energy list of the 56 benchmarks are facsimile to the predicted energy list of these exact programs.
- The zero approaching MSE declares that there are no seriously large errors between the true and the predicted values.
- The MAE lies much about in the same order, indicating that the sum of the absolute differences between actual and predicted values is at a low level.
- The R^2 score is great and shows that the true and the predicted energy values are almost identical. This means that 96.03% of all variations of the test applications energy consumption are captured by the proposed energy estimation model. In fact, the level of 0.960387 may be the baseline of this machine learning model, and a score to overcome in the future.

Visual Display of the Results

The actual and the predicted energy values comparison is illustrated in Figure 4.8. The energy values in the x axis and the y axis range between [0, 8] Joules.

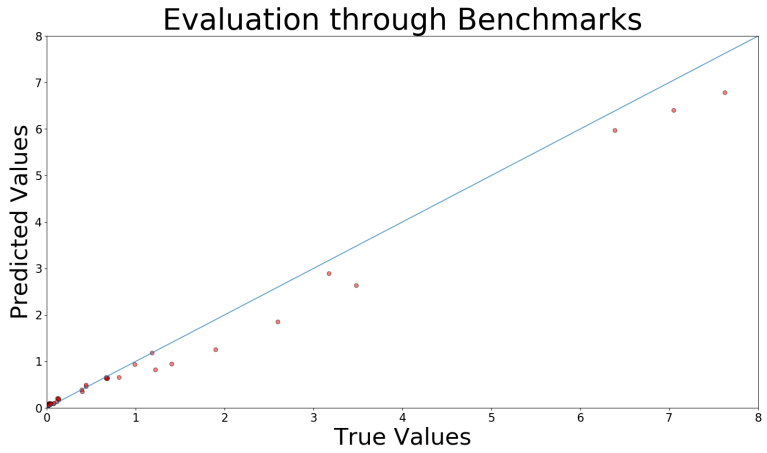


Figure 4.8: True vs Predicted Energy Values

As we see there are many points concentrated at the beginning of the axes. This is happening because there have been a lot of testing codes with low energy consumption. For this reason, Figure 4.9 is illustrated below and ranges the x axis and the y axis between [0, 1.2], focusing on the low energy consumed codes. Just like the previous graph, the x axis represents the measured in the lab's target machine true energy values, while the y axis represents the machine learning predicted energy values. The $y = x$ graph line is also fading and the actual points are coloured red.

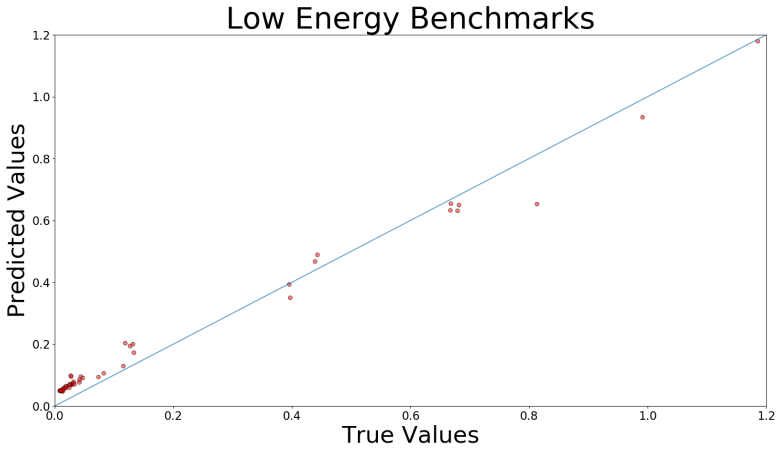


Figure 4.9: True vs Predicted Low Energy Values

It is obvious that the R^2 is verified through these graphs. Most of the points

are concentrated very near the ideal diagonal line, confirming the high level of prediction accuracy. It is a matter of concern that in the graph, the prediction was mostly overestimating the actual outcome ($y > x$), but not on a level to reduce the success of the predictions.

These prediction results can be completely understandable by comparing the actual and the predicted energy consumed by each individual benchmark. The following Figures 4.10 and 4.11 show the actual and the predicted energy values of low and medium energy benchmarks separately. It also indicated the name of each benchmark.

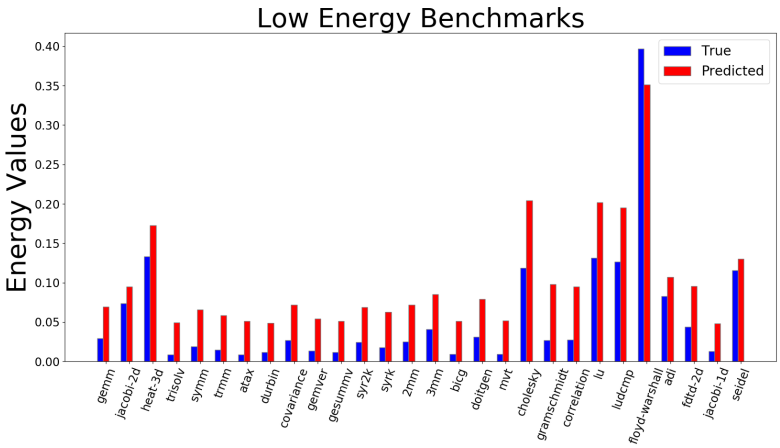


Figure 4.10: True vs Predicted Low Energy Values

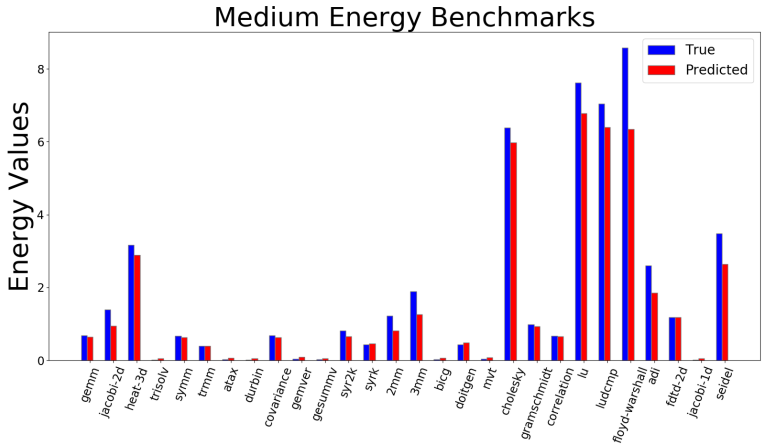


Figure 4.11: True vs Predicted Medium Energy Values

As can be clearly seen, the true and predicted energy values always belong to the same size class and in most of the cases the predicted energy values are very close to the actual ones. The failure rate in the prediction of the extremely low energy benchmarks, that we mentioned before, can be can be ascertained in Figure 4.10. On the other hand, energy prediction in codes that have energy requirements in the range [0.4, 1.2] is very successful.

Concluding, the accuracy of the prediction is at a very good level and the R^2 score of 0.960387 is a challenging baseline to overcome.

4.4 Test the method on alternative device

One of the tool's benefits is retargetability. The model has been trained through the energies measured on the Nvidia Jetson TX1 Board, using the INA3221 Power Sensor. Therefore, the predictions made by the model refer to energy values that will be consumed if the user runs his application on this particular board. Although, this tool can be adapted to any other device. An example of this application was done using the NVIDIA Jetson Xavier NX board, which was used to measure the energies of both the training and the benchmark datasets. The model was not modified and the results are been shown below:

```
Spearman coefficient between true and predicted is: 0.98018
Mean Squared Error (MSE): 0.030702
Mean Absolute Error (MAE): 0.102754
R squared (R^2): 0.769088
```

Figure 4.12: Evaluation of the method in alternative device

These results show a very high correlation between the predicted and the true values, almost approaching 1. Besides, the MSE score of 0.03 is considered to be very low and shows that there are almost no errors between the true and the predicted values. This is also confirmed considering the very low MAE score. On the other hand, the R^2 score is not fulfilling compared to the 0.96 achieved with the Nvidia Jetson TX1 Board energy dataset, but the level of 0.77 is considered to be satisfying.

In conclusion, this test proves that this model can be used to predict accurately the energy consumption on different boards. In a future implementation, the user will be able to "add" a target machine in the target device list and predict the energy cost of the code when run in this exact target device.

4.5 Comparison with alternative approaches

This tool is made for predicting the energy consumption of low and medium energy-wise codes. The predicting accuracy, as we showed in the previous chapter, is fully satisfactory. Especially according the R^2, which is also known as Coefficient of Determination or Goodness of fit, the results are startling, considering any possible additional experimental error like measurement fluctuations. Looking back to other approaches that we presented in Section 1.1, our energy estimation reaches a high level of success. More analytically:

- The approach [3] was undoubtedly an inspiration for us. An important difference in the architecture approach is that in [3] the focus is on splitting the program into phases and analyse them, while in this project we analyse the whole program and focus on the loops. The prediction accuracy of 0.97 was almost reached, according to R^2. Besides, the training dataset we chose to use is much larger (700 sample programs instead of 157 sample programs).

- The approach [4] is slightly different as it is designed on the basis of a specific microcontroller and a restricted instruction set, while in this tool is designed for managing any low and medium C language program. Therefore, the prediction accuracy of 0.9987 in terms of R^2 was almost unreachable. Despite this, the dataset we used is an order of magnitude larger (700 sample programs instead of 60 sample programs).

- Compared to [6], the prediction accuracy of 0.95 was slightly overtaken. Besides, in [6] the focus is on predicting the worst case scenario in energy consumption. Such approaches seem to be extremely slow and they can support only specific architecture models.

It is important to mention that in the approach we designed, there has been a feature analysis section in order to highlight the important features energy-wise. Additionally, the project is structured as a tool that can be easily used by a developer. These comparisons consolidate the quality of our approach and motivate us for more successful implementation and results in the future.

Chapter 5

Discussion Conclusions and Future Work

The aim of this project was to create an easy to use, handy tool that may help a developer in programming with better energy efficiency. The knowledge of the program's energy consumption, without using any energy measuring sensor, can lead to sustainable ways of programming. Once the developer ends up writing the code, he is able to check the predicted energy consumption and re-design the code in a more efficient way, if needed. The developer can also learn important information about the code behaviour through the resulting dataset, and maybe change the used hardware.

An accurate energy estimation model for low and medium energy programs was presented. By mainly profiling the memory behaviour of the program, the presented model leads to great results. It is structured for being a simple tool, much simpler than most proposed models that can speed up the energy estimation process and therefore the development of the embedded systems. The model was validated against a physical hardware platform, and the accuracy score of estimation for a number of applications from the PolyBench Benchmark Suite was 96.03%.

Retargetability is another advantage of our proposed model. By measuring the energy consumption of the 700 generated codes (train dataset) on a different target-platform, it is easy to use the model -without any modification- for predicting the energy consumption when running a program on this new platform. As a consequence, this tool would be very helpful in selecting the right hardware to suit a particular application.

Considering that an energy estimation tool with the capability of proposing design transformations could also be very helpful, we consider that a Loop Transformation Indicator is a possible candidate for our future research works. In this direction, the "#pragma scop" and "#pragma endscop" directives could help in indicating the "heaviest loop" and thereafter transform this loop in a more sufficient energy-wise way.

Appendix A

How to use the tool

Before using the tool, there are some requirements that the user has to manage. First of all, he has to install Valgrind and IntelPin on his computer. In addition, the paths in the real_test_analysis.py and the predict_energy.py scripts have to be changed in order to adapt those scripts to his computer. After managing the above, the user is ready to use the tool.

The use of this tool is very simple and consists of three steps. First of all, the user has to specify the "heaviest" loop by putting "#pragma scop" and "#pragma endscop" around it, in order to help the analysis. Then, he has to run the real_test_analysis.py script for his mycode.c file in order to create the dataset.csv file, just like the example below:

$python real_test_analysis.py mycode

Finally, the developer has to run the predict_energy.py script for his mycode.c file to see the predicted energy consumption of his application, just like the example below:

$python predict_energy.py mycode

These steps are been shown in Figure A.1.

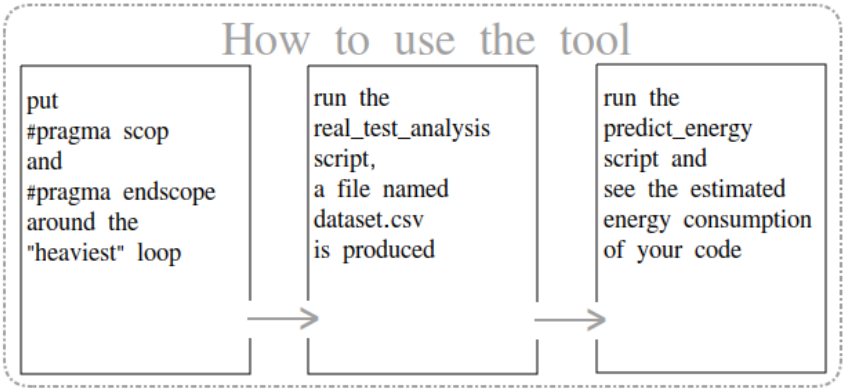


Figure A.1: Steps for using the tool

Bibliography

[1] N. Hossein Motlagh, M. Mohammadrezaei, J. Hunt, and B. Zakeri, "Internet of things (iot) and the energy sector," *Energies*, vol. 13, no. 2, p. 494, 2020.

[2] L. Hilty, B. Aebischer, G. Andersson, and W. Lohmann, "Ict4s–ict for sustainability: Proceedings of the first international conference on information and communication technologies for sustainability," 2013.

[3] X. Zheng, L. K. John, and A. Gerstlauer, "Accurate phase-level cross-platform power and performance estimation," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2016.

[4] M. Bazzaz, M. Salehi, and A. Ejlali, "An accurate instruction-level energy estimation model and tool for embedded systems," *IEEE transactions on instrumentation and measurement*, vol. 62, no. 7, pp. 1927–1934, 2013.

[5] R. Heckmann and C. Ferdinand, "Worst-case execution time prediction by static program analysis," in *In 18th International Parallel and Distributed Processing Symposium (IPDPS 2004, pages 26–30. IEEE Computer Society*, 2004.

[6] G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo, and P. Cunha, "Energy consumption and execution time estimation of embedded system applications," *Microprocessors and Microsystems*, vol. 35, no. 4, pp. 426–440, 2011.

[7] C. Mendis, A. Renda, S. Amarasinghe, and M. Carbin, "Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks," in *International Conference on machine learning*, pp. 4505–4515, PMLR, 2019.