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

Basic Block energy prediction using machine learning methods

-

Εφαρμογή Τεχνικών Μηχανικής Μάθησης στη Μοντελοποίηση Κατανάλωσης Ενέργειας Στοιχειωδών Δομών Λογισμικού

Διπλωματική Εργασία

του

Σιώζου Θεόδωρου

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Απρίλιος 2023

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

Basic Block energy prediction using machine learning methods

-

Εφαρμογή Τεχνικών Μηχανικής Μάθησης στη Μοντελοποίηση Κατανάλωσης Ενέργειας Στοιχειωδών Δομών Λογισμικού

Διπλωματική Εργασία

του

Σιώζου Θεόδωρου

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4 Απριλίου 2023.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δημήτριος Σούντρης	Παναγιώτης Τσανάκας	Σωτήριος Ξύδης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π.

Αθήνα, Απρίλιος 2023

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

(Υπογραφή)

.....

Σιώζος Θεόδωρος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τις τελευταίες δεκαετίες έχει καταβληθεί σημαντική προσπάθεια για την εξεύρεση ενεργειακά αποδοτικών λύσεων λόγω της κλιματικής αλλαγής. Η πρόβλεψη της ενεργειακής κατανάλωσης των στοιχειωδών δομών κώδικα (basic block) είναι ένα ουσιαστικό βήμα για τη δημιουργία ενεργειακά βελτιστοποιημένων μεταγλωττιστών και χρονοπρογραμματιστών που μπορούν να μειώσουν σημαντικά την ενεργειακή κατανάλωση του κώδικα. Ωστόσο, το έργο αυτό παρουσιάζει αρκετές προκλήσεις λόγω της πολύ μεγάλης λεπτομέρειας που χαρακτηρίζει τα basic blocks καθώς και των απαιτούμενων αισθητήρων του υπολογιστικού συστήματος. Για την αντιμετώπιση αυτών των προκλήσεων, η παρούσα έρευνα διερευνά την αποτελεσματικότητα των προσεγγίσεων μηχανικής μάθησης στην πρόβλεψη της κατανάλωσης ενέργειας basic block που μπορεί να παρέχει ένα εργαλείο διαγνωστικό ως προς το υλικό. Ένα σύνολο δεδομένων μέτρησης ενέργειας basic block που δημιουργήθηκε σε προηγούμενη έρευνα χρησιμοποιήθηκε για την εκπαίδευση διαφόρων αρχιτεκτονικών μηχανικής μάθησης, συμπεριλαμβανομένων τόσο παραδοσιακών μεθόδων παρεμβολής (regression) όσο και αρχιτεκτονικών βαθιών νευρωνικών δικτύων. Ειδικότερα, χρησιμοποιήθηκαν αναδρομικά νευρωνικά δίκτυα, όπως τα δίκτυα μακράς και βραχείας μνήμης (LSTMs), για να επιτευχθεί υψηλή ακρίβεια στην πρόβλεψη της κατανάλωσης ενέργειας του basic block, ενώ οι παραδοσιακές μέθοδοι όπως η γραμμική παρεμβολή (linear regression) και οι μηχανές διανυσμάτων υποστήριξης (SVMs) αποδείχθηκαν επίσης πολύ καλές. Τα τελικά μοντέλα με τις καλύτερες επιδόσεις ήταν σε θέση να προβλέψουν την κατανάλωση ενέργειας του basic block με μέσο απόλυτο σφάλμα που κυμαίνεται γύρω στο 0,25 $(*61\mu J)$ για μετρήσεις ενέργειας με τυπική απόκλιση γύρω στο 0,7 $(*61\mu J)$. Παρά αυτά τα πολλά υποσχόμενα αποτελέσματα, η έρευνα ανέδειξε πως η κατανάλωση ενέργειας των basic block δεν σχετίζεται μόνο από τα basic blocks καθ' αυτά. Οι τελικές συνεισφορές αυτής της έρευνας μπορούν να συνοψιστούν ως εξής: μια εκτεταμένη εμπειρική μελέτη της μηχανικής μάθησης για την πρόβλεψη της κατανάλωσης ενέργειας, η συνειδητοποίηση ότι η πρόβλεψη της ενέργειας των basic block απαιτεί πληροφορίες σχετικά με το πλαίσιο για τα προηγούμενα basic block, η αναγνώριση ότι το σφάλμα ολόκληρου του προγράμματος είναι ένα ανεπαρκές μέτρο για την αξιολόγηση ενός συνόλου δεδομένων basic block και, τέλος, ένα τεκμηριωμένο περίγραμμα των μελλοντικών εργασιών.

Λέξεις Κλειδιά

μηχανική μάθηση, νευρωνικά δίκτυα, κατανάλωση ενέργειας basic block, παρεμβολή

Abstract

Over the past few decades, there has been a significant effort to find energy-efficient solutions due to climate change. One approach to reducing energy consumption is to minimize the energy consumption of code, particularly for large-scale scenarios such as data centers. Basic block energy consumption prediction is an essential step in creating energy-optimized compilers and schedulers that can significantly decrease code energy consumption. However, this task poses several challenges due to its fine granularity and hardware-specific sensors.

To address these challenges, this research investigates the efficiency of machine learning approaches in basic block energy consumption prediction. A basic block energy measurement dataset created in previous research was used for training several machine learning architectures, including both traditional regression methods and deep neural network architectures. Sophisticated code representation techniques were implemented to enable this task, given the nature of the basic block as a sequence of Assembly instructions.

In particular, sequential neural networks, such as LSTMs, were used to achieve high accuracy in predicting basic block energy consumption, while traditional methods like linear regression and support vector machines proved to perform great as well. The final best-performing models were able to predict basic block energy consumption with a Mean Absolute Error fluctuating around 0.25 $(*61\mu J)$ for energy measurements with standard deviation around 0.7 $(*61\mu J)$. Despite these promising results, the research has discovered that basic block energy consumption is highly correlated with its preceding basic blocks, therefore the findings were deteriorated due to the lack of data for basic blocks sequences.

The final contributions of this research can be summed up as: an extensive empirical study of machine learning for energy consumption prediction, the realization that basic block energy prediction necessitates contextual information for preceding basic blocks, the recognition that whole-program error is an inadequate measure for evaluating a basic block dataset, and lastly, an informed outline of future work.

Keywords

machine learning, neural networks, basic block energy consumption, regression models

Acknowledgements

At this point I would like to personally thank the people that greatly helped me through this research and through my journey for completing my studies on Electrical and Computer Engineering at the National Technical University of Athens.

First and foremost I would like to thank my thesis supervising professor, Prof. Dimitrios Soudris, for giving me the opportunity to collaborate with the Micro-Processors Lab for the purposes of my research. Also, this work would not be possible without the crucial help, supervision and constant support of the PhD candidate, Christos Labrakos. Finally, my great friend and classmate Aristotelis Koutris offered significant guidance for many of the first and second-year courses, and ultimately brought me closer to understanding and completing my studies.

Απρίλιος 2023

Σιώζος Θεόδωρος

Contents

List of Figures

List of Tables

Abbreviations

Adaptive Moment Estimation	Adam
Application Programming Interface	API
Convolutional Neural Network	CNN
Deep Neural Network	DNN
Fully Connected	FC
Inverse Document Frequency	IDF
Low Level Virtual Machine	LLVM
Language Model	LM
Long Short Term Memory Network	LSTMs
Mean Absolute Error	MAE
Mean Absolute Percentage Error	MAPE
Mask Language Model	MLM
Mean Squared Error	MSE
Running Average Power Limit	RAPL
Radial Basis Function	RBF
rectified linear unit	ReLU
Root Mean Squared Error	RMSE
Recurrent Neural Network	RNN
Stochastic Gradient Descent	SGD
Support Vector Regression	SVR
Support Vector Machine	SVM
Term Frequency	TF
Term Frequency-Inverse Document Frequency	TFIDF
Tree-structured Parzen Estimator	TPE

Εκτεταμένη Περίληψη

Η συγκεκριμένη έρευνα παρουσιάζει την ανάπτυξη μεθόδων πρόβλεψης ενέργειας basic block χρησιμοποιώντας τεχνικές μηχανικής μάθησης και υπάγεται στο ευρύτερο πλαίσιο της ενεργειακής βελτιστοποίησης συστημάτων. Οι βασικές συνεισφορές συνοψίζονται ως εξής:

- Ανάλυση δεδομένων που παρείχε βασικές πληροφορίες για τα δεδομένα ενέργειας basic block.
- Υλοποιήθηκε η πρόβλεψη ενέργειας με τη χρήση παραδοσιακών αλγορίθμων regression με τη χρήση scikit-learn, παράγοντας χρήσιμα αποτελέσματα για σύγκριση με τις τεχνικές βαθιάς μάθησης.
- Πρόβλεψη ενέργειας με τη χρήση πλήρως συνδεδεμένων βαθιών νευρωνικών δικτύων (Dense Neural Networks) που βασίζονται στο μέσο εμφύτευμα (embedding) εκ των PalmTree embeddings των εντολών Assembly.
- Ανάπτυξη νευρωνικών δικτύων LSTM με χρήση PalmTree embeddings, καθώς και με δημιουργία προσαρμοσμένων embeddings, για την πρόβλεψη της ενέργειας με χρήση της διαδοχικής φύσης των εντολών που αποτελούν ένα basic block.
- Διαπιστώσεις σχετικά με την ακολουθία των προηγούμενων basic block ως απαραίτητο νοηματικό πλαίσιο για την πρόβλεψη της κατανάλωσης ενέργειας του basic block.
- Διαπιστώσεις σχετικά με το σφάλμα ολόκληρου του προγράμματος ως παραπλανητική μετρική για την αξιολόγηση των δεδομένων ενέργειας των basic block.
- Λεπτομερής περιγραφή μελλοντικής έρευνας.

Δεν υπάρχουν πολλές προσεγγίσεις σχετικά με πρόβλεψη κατανάλωσης ενέργειας κώδικα σε στοιχειώδες επίπεδο. Οι τρεις βασικότερες που έχουν αναπτυχθεί παρουσιάζονται στις έρευνες [1], [12] και [2]. Η πρώτη εξ αυτών, παρουσιάζει το εργαλείο HAECER, το οποίο παρέχει ακριβή εκτίμηση των βραχυπρόθεσμων μετρήσεων κατανάλωσης ενέργειας που προέρχονται από το εργαλείο RAPL της Intel[1]. Παρόλο που η συγκεκριμένη τεχνική παράγει ακριβή αποτελέσματα, προαπαιτεί την ύπαρξη του RAPL. Το ALEA, χρησιμοποιεί μετρήσεις ισχύος που λαμβάνονται από ενσωματωμένους αισθητήρες ενέργειας ή από κατάλληλο λογισμικό και τις αξιοποιεί για την πρόβλεψη ενέργειας basic block με χρήση πιθανοτικών τεχνικών[12]. Τέλος, μια προσέγ-

γιση που χρησιμοποιεί βαθιά νευρωνικά δίκτυα, και συγκεκριμένα ένα σύστημα ιεραρχικών LSMT, παρουσιάζεται στο Ithemal, που στοχεύει στην ακριβή πρόβλεψη των κύκλων του επεξεργαστή που απαιτεί η εκτέλεση ενός basic block, πετυχαίνοντας έγκυρα αποτελέσματα[2]. Ορισμένες ιδέες που χρησιμοποίουνται στην έρευνα του Ithemal, αξιοποιούνται και στην παρούσα εργασία.

Τα δεδομένα, βάσει της μορφής τους, μπορούν σε γενικές γραμμές να ταξινομηθούν σε δύο κατηγορίες: δομημένα και μη δομημένα δεδομένα. Τα δομημένα δεδομένα είναι οργανωμένα σε μια σαφώς καθορισμένη μορφή, ενώ τα μη δομημένα δεν ακολουθούν ένα συγκεκριμένο σχήμα, παρουσιάζοντας προκλήσεις στην όσον αφορά την επεξεργασία και την ανάλυση τους. Ο κώδικας, όντας μια υποπερίπτωση κειμένου, ανήκει στα μη δομημένα δεδομένα και για την αναπαράσταση σε μορφή αξιοποιήσιμη από τεχνικές μηχανικής μάθησης, χρησιμοποιούνται μαθηματικά διανύσματα που τον κωδικοποιούν. Μια από τις πιο βασικές τεχνικές για κωδικοποίηση κειμένου, αποτελεί η κωδικοποίηση one-hot, κατά την οποία μία λέξη, ή στην συγκεκριμένη περίπτωση μια εντολή Assembly, αναπαριστάται ως ένα διάνυσμα μηδενικών, μεγέθους όσο και οι διαφορετικές εντολές που περιέχονται στα δεδομένα, με μοναδικό άσο την θέση που αντιστοιχεί στην συγκεκριμένη εντολή. Με τον τρόπο αυτό ένα basic block παρουσιάζεται ως το διάνυσμα-άθροισμα των one-hot διανυσμάτων των εντολών που το αποτελούν. Μια επέκταση της παραπάνω αναπαράστασης προτείνεται με την χρήση του αλγορίθμου συχνότητας όρου-αντίστροφης συχνότητας κείμενου (TFIDF), κατά την οποία αντιστοιχίζεται στην κάθε λέξη-εντολή ένα κόστος που υπολογίζεται βάσει του αριθμού εμφανίσεων της εμφάνισης της σε όλα τα έγγραφα-basic blocks και του αριθμού εμφανίσεων της σε ένα συγκεκριμένο έγγραφο-basic block[13]. Ωστόσο, οι προαναφερόμενες τεχνικές δεν αντικατοπτρίζουν τα σημασιολογικά χαρακτηριστικά των εντολών. Για τον λόγο αυτό, δημιουργούνται διανύσματα που σηματοδοτούν τα εννοιολογικά χαρακτηριστικά των λέξεων (ή εντολών Assembly στην συγκεκριμένη περίπτωση) και ονομάζονται εμφυτεύματα (embeddings)[14]. Ειδικές αρχιτεκτονικές βαθιών νευρωνικών δικτύων έχουν εγκαθιδρυθεί με στόχο την παραγωγή πλούσιων νοηματικά embeddings και ονομάζονται μετασχηματιστές (Transformers)[15]. Το μοντέλο PalmTree βασίζεται στην εκπαίδευση του BERT Transformer[16] για εντολές Assembly, με σκοπό την παραγωγή σημασιολογικά πλούσιων embeddings για τις εντολές Assembly[6]. Στην παρούσα έρευνα, πειραματιζόμαστε και με τα εν λόγω embeddings, καθώς και με εκ νέου δημιουργία embeddings ειδικών για την πρόβλεψη κατανάλωσης ενέργειας.

Το πρόβλημα πρόβλεψης της κατανάλωσης ενέργειας ενός basic block εντάσσεται στην οικογένεια προβλημάτων regression, κατά την οποία, το μοντέλο ή ο αλγόριθμος μηχανικής μάθησης καλείται να προβλέψει μία τιμή από ένα συνεχές σύνολο αριθμών. Ο βασικότερος αλγόριθμος regression είναι η γραμμική παρεμβολή (linear regression), κατά την οποία δημιουργείται μία γραμμική εξίσωση εισόδου-εξόδου, της οποίας οι παράμετροι-βάρη βελτιστοποιούνται ώστε να επιτυγχάνεται το μικρότερο σφάλμα μεταξύ της πρόβλεψης και των πραγματικών τιμών των δεδομένων[17]. Οι μέθοδοι Lasso, Ridge και ElasticNet Regression[18],[19], [20] αποτελούν επεκτάσεις της Linear Regression και βασίζονται στην ένταξη ενός όρου κανονικοποίησης βαρών στην συνάρτηση κόστους. Επιπλέον, η στοχαστική κάθοδος βασισμένη στην κλίση (Stochastic Gra-

dient Descent), παρουσιάζει έναν παραδοσιακό αλγόριθμο εκπαίδευσης βαρών βάσει των παραγώγων των παραμέτρων και την επιλογή ενός τυχαίου δεδομένου ανά βήμα εκπαίδευσης[21]. Τέλος, οι αλγόριθμοι υποστηρικτικών διανυσμάτων (SVMs) και ενίσχυσης καθόδου κλίσης (Gradient Boosting) ολοκληρώνουν τους παραδοσιακούς αλγόριθμους μηχανικής μάθησης για regression προβλήματα. Τα SVMs καθορίζουν υποστηρικτικά διανύσματα για την εύρεση μίας συνάρτησης που μοντελοποιεί τα δεδομένα[22], ενώ ο Gradient Boosting βασίζονται στην εκπαίδευσης δεντρών αποφάσεων (decision trees) με χρήση gradient descent.

Τα βαθιά νευρωνικά δίκτυα αποτελούν ένα εργαλείο για την δημιουργία ισχυρών μοντέλων regression. Βασίζονται στον συνδυασμό μεγάλου αριθμού στοιχειωδών δομών που ονομάζονται νευρώνες και δημιουργούν στρώματα. Ο κάθε νευρώνας περιέχει συγκεκριμένες παραμέτρους που χρησιμοποιούνται σε μαθηματικούς μετασχηματισμούς και συγκεκριμένα σε πολλαπλασιασμούς πινάκων που καταλήγουν στον υπολογισμό της εξόδου του νευρωνικού δικτύου. Οι παράμετροι του δικτύου εκπαιδεύονται συνολικά χρησιμοποιώντας των αλγόριθμο gradient descent, καθώς και περαιτέρω παραμέτρους που καθορίζουν την εκπαίδευση του δικτύου. Τα νευρωνικά δίκτυα που δημιουργήθηκαν για την πρόβλεψη της ενέργειας των basic block, ανήκουν στις κατηγορίες Dense Neural Networks και LSTMs. Τα πρώτα χαρακτηρίζονται από στρώματα νευρώνων που συνδέονται με όλους τους νευρώνες των γειτονικών στρωμάτων, ενώ τα LSTMs χρησιμοποιούν πολύπλοκους μετασχηματισμούς για την επεξεργασία διαδοχικών εισόδων, αξιοποιώντας ένα εξελιγμένο σύστημα μνήμης σημαντικών πληροφοριών της ακολουθίας[23]. Η συγκεκριμένη περίπτωση βασίζεται πάνω στην ακολουθία των εντολών Assembly που απαρτίζουν το basic block. Τέλος, σε μία συγκεκριμένη μέθοδο που αναπτύξαμε γίνεται χρήση στρωμάτων εμφυτευμάτων, τα οποία δημιουργούν κατάλληλα embeddings εντολών Assembly για την πρόβλεψη κατανάλωσης ενέργειας basic block.

Η ανάπτυξη των μοντέλων μηχανικής μάθησης που παρουσιάζονται στην συγκεκριμένη διπλωματική, βασίστηκε σε ένα σύνολο δεδομένων που περιέχει basic blocks μαζί με τις αντίστοιχες ενέργειες τους και δημιουργήθηκε στα πλαίσια προηγούμενης έρευνας. Για την δημιουργία του συγκεκριμένου συνόλου δεδομένων, αξιοποιήθηκε το εργαλείο RAPL για τις μετρήσεις της ενέργειας των basic block, καθώς και εξελιγμένοι μέθοδοι για την απομόνωση τους. Τα basic blocks προήλθαν από ανάλυση του ίχνους εκτέλεσης προγραμμάτων C, που αποτέλεσαν και τα προγράμματα αναφορά των δεδομένων. Τα συνολικά δεδομένα είναι περίπου 550.000.

Η εκτενής ανάλυση και προεπεξεργασία του συνόλου δεδομένου basic block-ενέργειας αποτέλεσε το πρώτο βήμα της συγκεκριμένης εργασίας. Με τον τρόπο αυτό εξάχθηκαν χρήσιμα και λεπτομερή στατιστικά και χαρακτηριστικά των δεδομένων. Επίσης, η προεπεξεργασία ακολούθησε την προεπεξεργασία που προτείνει η έρευνα του PalmTree[6], αφαιρώντας από κάθε εντολή τους σταθερούς τελεστές που δηλώνουν θέσεις μνήμης, καθώς η μόνη επίδραση τους στην κατανάλωση ενέργειας πηγάζει στα cache hits και misses, πληροφορία που δεν υπάρχει στο συγκεκριμένο πρόβλημα.

Οι πρώτες μέθοδοι μηχανικής μάθησης που δοκιμάστηκαν για την πρόβλεψη κατανάλωσης ενέργειας, απαρτίζονται από την εφαρμογή των παραδοσιακών αλγόριθμων

regression χρησιμοποιώντας την βιβλιοθήκη scikit-learn και θέτοντας σαν είσοδο κάθε δεδομένου το διανυσματικό άθροισμα των one-hot αναπαραστάσεων των εντολών που περιέχει κάθε basic block. Η βελτιστοποίηση των υπερπαραμέτρων των μοντέλων, εκτός από την εύρεση των κατάλληλων παραμέτρων των αλγορίθμων, εμπεριέχει και την εύρεση των επιπλέον βημάτων που παράγουν τα καλύτερα αποτελέσματα και επιτυγχάνεται με τη χρήση του λογισμικού Optuna[11]. Τα εν δυνάμει βήματα αυτά είναι η μετατροπή του διανύσματος σε διάνυσμα TFIDF και η κανονικοποίηση του μέτρου και της απόκλισης του διανύσματος.

Τέλος, αναπτύχθηκαν και τρεις διαφορετικές τεχνικές που χρησιμοποιούν βαθιά νευρωνικά δίκτυα για την πρόβλεψη κατανάλωσης ενέργειας basic block. Οι πρώτες δύο μέθοδοι βασίστηκαν στην αναπαράσταση των εντολών Assembly του κάθε basic block ως PalmTree embeddings και διακρίνονται στο LSTM μοντέλο και στο Dense μοντέλο. Το LSTM μοντέλο αξιοποιεί το embedding κάθε εντολής του basic block ως ένα καρέ της ακολουθίας και προβλέπει την ενέργεια χρησιμοποιώντας στρώματα LSTM και στο τέλος κάποια Dense στρώματα. Το Dense μοντέλο υπολογίζει ένα αντιπροσωπευτικό embedding για κάθε basic block ως το μέσο των PalmTree embeddings των εντολών που το αποτελούν και προβλέπει την ενέργεια χρησιμοποιώντας Dense στρώματα. Το τρίτο μοντέλο βαθιάς μάθησης βασίζεται και αυτό σε στρώματα LSTM για την πρόβλεψη ενέργειας, με τη βασική διαφορά ότι δημιουργείται ένα καινούριο λεξικό εντολών Assembly για το σύνολο δεδομένων και τα embeddings των εντολών εκπαιδεύονται αξιοποιώντας ένα Embedding στρώμα στην αρχή του δικτύου. Η βελτιστοποίηση υπερπαραμέτρων για τις συγκεκριμένες τεχνικές, συμπεριλαμβάνει εύρεση των λεπτομερών αρχιτεκτονικών των νευρωνικών δικτύων καθώς και των υπαρπαραμέτρων εκπαίδευσης των μοντέλων και επιτυγχάνεται με την χρήση του λογισμικού Optuna[11].

Η ανάλυση του συνόλου δεδομένων επέφερε σημαντικά ευρήματα για την ερμηνεία των αποτελεσμάτων. Αρχικά, τα 565.679 basic blocks ανάγονται σε 3.828 μοναδικά basic blocks, από τα οποία τα περισσότερα εμφανίζονται πολλές φορές και με διαφορετική ενέργεια. Το συγκεκριμένο στοιχείο αποκαλύπτει, πως υπάρχει και συσχέτιση μεταξύ τον κώδικα που προηγείται του basic block και της ενέργειας που το basic block καταναλώνει, καθώς γίνεται φανερό πως η ενέργεια δεν εξαρτάται αποκλειστικά και μόνο από το ίδιο το basic block. Επομένως, τα αποτελέσματα των μοντέλων μας επηρεάζονται σε μεγάλο βαθμό από την έλλειψη της επιπλέον απαιτούμενης πληροφορίας. Επίσης, οι ενέργειες των basic block χαρακτηρίζονται από μέσο μέτρο 0.39 ($*61\mu J$), από διάμεσο 0.18 ($*61\mu J$) και από τυπική απόκλιση 0.7 ($*61\mu J$). Ακόμη, παράχθηκαν το ιστογράμματα των κατανομών των ενεργειών των basic block για τα διάφορα προγράμματα του συνόλου δεδομένων και εξάχθηκε το συμπέρασμα πως η σύγκριση θα πρέπει να διεξαχθεί αγνοώντας το πρόγραμμα από το οποίο προέρχεται κάθε basic block, λόγω της ανισορροπίας δεδομένων ανά τα διαφορετικά προγράμματα καθώς και της μεγάλης λεπτομέρειας που διακρίνει τα basic block έναντι των προγραμμάτων. Τέλος, αναδείξαμε πως η αξιολόγηση των δεδομένων με χρήση της συνολικής ενέργειας προγράμματος είναι παραπλανητική, καθώς μπορεί να επιδεινωθεί ή να αμβλυνθεί από συγκεκριμένα ακραίες τιμές δεδομένων και αποτυγχάνει να αξιολόγηση

την ακρίβεια σε επίπεδο ενέργειας basic block.

Για την αξιολόγηση των μοντέλων πρόβλεψης ενέργειας, Θεσπίστηκαν τρία διαφορετικά σενάρια, με το ένα εν τέλει να θεωρείται το πιο αξιόπιστο δεδομένων των συνθηκών. Στο πρώτο σενάριο, τα μοντέλα αξιολογούνται σε ένα σύνολο δεδομένων δοκιμών, που δεν συμπεριλαμβάνεται στην εκπαίδευση των μοντέλων και αποτελεί το 10% των συνολικών δεδομένων. Αυτή η διαδικασία επαναλαμβάνεται πολλαπλές φορές με εκ νέου τυχαίο διαχωρισμό δεδομένων δοκιμών και εκπαίδευσης, ώστε να εξασφαλιστεί η σταθερή απόδοση, ανεξαρτήτως της διαλογής συνόλων. Στην περίπτωση αυτή, που θεωρείται και η πιο αξιόπιστη, τα μοντέλα LSTM με εκπαίδευση νέων embeddings, SVMs και Gradient Boosting δίνουν τα καλύτερα αποτελέσματα πετυχαίνοντας μέσω σφάλμα 0.25 ($*61\mu J$), ενώ τα μοντέλα Linear και Ridge regression πετυχαίνουν ελάχιστα χειρότερα αποτελέσματα. Τα επόμενα δύο σενάρια αξιολόγησης πηγάζουν από αξιολόγηση βάσει προγράμματος προέλευσης basic block και από αξιολόγηση των μοναδικών basic block με ενέργεια την διάμεση τιμή των ενεργειών όλων των εμφανίσεων του basic block, αντίστοιχα. Ωστόσο, αιτιολογήθηκαν τα μειονεκτήματα της σύγκρισης βάσει προγράμματος, ενώ η περίπτωση των μοναδικών basic block δεν ενδείκνυται για τις μεθόδους μηχανικής μάθησης που βασίζονται σε μεγάλο αριθμό δεδομένων.

Συμπερασματικά, η παρούσα έρευνα περιλαμβάνει μία εκτενή ανάπτυξη εργαλείων που βασίζονται στην μηχανική μάθηση, για την πρόβλεψη κατανάλωσης ενέργειας basic block. Τα τελικά αποτελέσματα είναι ενθαρρυντικά επιτυγχάνοντας μέσο σφάλμα 0.25 ($*61\mu J$) για δεδομένα με τυπική απόκλιση 0.7 ($*61\mu J$). Οι κατευθύνσεις που προτείνονται για συνέχεια της συγκεκριμένης έρευνας επικεντρώνονται στην ένταξη κώδικα που προηγείται κάθε basic block, ως απαραίτητη πληροφορία για την ενέργεια που εν τέλει καταναλώνει το basic block. Επίσης, ενθαρρύνεται η χρήση αρχιτεκτονικών Transformer για περαιτέρω βελτίωση των αποτελεσμάτων ή ιεραρχικών αρχιτεκτονικών LSTM. Ακόμη, προτείνεται η μελέτη επίδρασης της κατάστασης του συστήματος στην ενέργεια του basic block, καθώς παρατηρήθηκαν διαφορετικές μετρήσεις ενέργειας για διαφορετικές εκτελέσεις του ίδιου προγράμματος. Τέλος, η δημιουργία ενός συστήματος που λαμβάνει υπόψιν τις προβλέψεις της ενέργειας για την μεταγλώττιση ενεργειακά αποδοτικότερου κώδικα ολοκληρώνει τον τομέα της συγκεκριμένης έρευνας.

Chapter 1

Introduction

1.1 Motivation

Energy consumption inflicted by software suggests a significant issue. Minimizing programming code energy demands, provides both environmental and cost-effective benefits, especially when it comes to large data centers. Extracting an accurate energy consumption assessment consists a step in the direction of proposing sustainable solutions regarding energy demands. The higher the granularity of code that the energy prediction corresponds to, the more the applications that are available. Achieving fine grain results can enable compiling more efficient resource-demanding programs, energy dependent operating system scheduling and generally highly applicable solutions. A basic block consists a series of consecutive Assembly instructions that are not interrupted by jump or branch, and its energy consumption constitutes a high resolution approach.

Energy measurements of basic blocks rise many challenges, mainly due to the high granularity of the task and the hardware-specific sensors. Intel's Running Average Power Limit (RAPL), a sensor providing energy measurement in software-function level , consists an Intel-dependent solution with limitations when it comes to granularity as the measurement time interval is close to 1ms[1],[24].

1.2 Thesis Objective

This thesis examines basic block energy consumption prediction techniques based on machine learning and deep learning methods, in order to create a framework that provides fine-grained results. The work is based on a basic block - energy dataset developed for this purpose, so as to enable supervised training of the models. For the basic block energy labels of the dataset basic blocks to be extracted, a sophisticated approach using RAPL was implemented and validated.

In the direction of providing meaningful features for the machine learning models, a basic block semantic representation must be established. Two main approaches were introduced based on the concept of embeddings[14] - vectors that capture the meaning of data. By applying the specific concept in the task at hand, we wish to create instruction embeddings. The first approach uses pretrained assembly x86 instruction embeddings,

created by the PalmTree framework[6]. We also implemented a second approach, that produces custom embeddings inside the model, given an instruction vocabulary which was created from the dataset's basic block instructions. In the end, each basic block is represented by a sequence of embeddings corresponding to the sequence of the instructions consisting it.

Using the basic block representations as described, various machine learning architectures were introduced with the goal of producing accurate predictions regarding the basic block energy consumption. Taking advantage of the sequential nature of instructions in each basic block, two Long Short-Term Memory (LSTM) network based approaches were developed - one using the pretrained embeddings and the other making use of the custom vocabulary. We also implemented a simpler architecture - inspired by the sentence embeddings idea - where we consider each instruction as a word and each basic block as a sentence, producing a basic block embedding by computing the mean PalmTree embedding of its instructions[25]. Finally, multiple well-known regressors (Linear Regressor, Lasso Regressor, etc.) were investigated in order to produce some baseline results.

1.3 Thesis Contribution

The main contributions of the thesis include:

- Data analysis was performed providing key insights on the energy - basic-block dataset.
- Prediction using traditional scikit-learn regressors was implemented producing promising results to compare the deep learning techniques.
- Energy prediction using dense deep neural networks based on mean embedding of PalmTree instructions.
- Development of LSTM neural networks using PalmTree embeddings, as well as custom embeddings, for energy prediction making use of the sequential nature of instructions inside a basic block.
- Insights regarding the preceding basic blocks sequence as context for the basic block energy consumption prediction.
- Insights regarding the whole-program error as a misleading metric for basic block dataset evaluation.
- Detailed description of future work.

1.4 Thesis Structure

The rest of the thesis is organized as follows:

- Chapter 2 presents research and tools related to code energy prediction.
- In Chapter 3, the theoretical background of the concepts used in this thesis is analyzed in great detail.
- The implementation of the proposed techniques is explained in Chapter 4.
- The results of said techniques and developments are presented in great length in Chapter 5, along with their interpretation.
- Chapter 6 concludes the thesis and suggests directions in which the research can continue.

Chapter 2

Related Work

There are not many approaches regarding fine-grained energy-related predictions, but the ones that have been developed can be divided into two categories. The first one aims to use sensor measurements as the base for energy prediction with some additional post-processing, while the second uses deep neural network techniques in order to produce accurate basic block throughput predictions.

The groundbreaking work that tries to attain highly granular energy profiling is documented in[1]. The authors analyze in great detail the functioning of RAPL measurements and put forth the HAECER framework, which provides a precise assessment of short-term energy consumption. The challenges that HAECER aims to address, originates in the RAPL measurements interval, as the time difference between RAPL reads and function starting and ending point result in misleading estimation regarding energy consumption (Figure 2.1a). For the measurement to be accurate, the instruction call should be aligned with a RAPL read (Figure 2.1b) and the energy after the function return should be subtracted along with the cost of the synchronization mandated by the method[1]. Although this methodology yields accurate results, it is restricted to hardware that supports RAPL and to function-level measurements. The underlying concept of the HAECER framework was leveraged in the creation of the basic-block energy dataset.

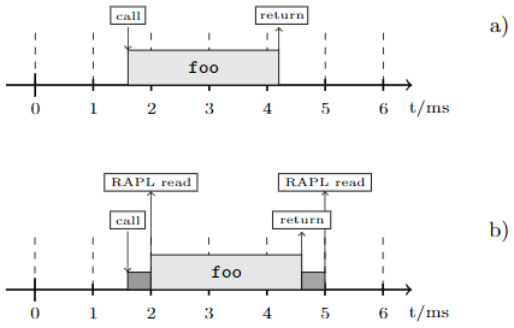


Figure 2.1: *Interval challenge that HAECER aims to solve[1]*

A more recent and broader approach that generates precise basic-block energy measurements is proposed by ALEA: A Fine-Grained Energy Profiling Tool. ALEA uses power

measurement readings obtained by on-board energy sensors (including RAPL), or software power models that are based on hardware counters. The process begins with the disassembly of the executable program to identify the basic blocks. Through the implementation of multiple power measurements for the exact same sequences of code blocks and by implementing probabilistic techniques, ALEA calculates the execution time and the power for each basic block. The final energy predictions, which can be utilized for intervals as short as $10\mu s$, are derived by applying the formula $energy = time * power$ and are highly accurate. We can notice that ALEA constitutes a highly sophisticated tool that provides great results and can be applied to a diverse range of platforms[12]. However, it requires hardware sensors and its use is not feasible in a hardware-agnostic scenario.

The requirement for a fine-grained tool that relates to energy prediction is imperative. A novel approach to basic block throughput prediction framework named Ithemal demonstrates promising results and manages to operate without any hardware dependencies. Ithemal predicts the number of processor clock cycles required by a sequence of assembly instructions, which consist a basic block, through a deep learning approach incorporating of a hierarchical LSTM architecture. The instructions of the basic block undergo tokenization, resulting in the conversion of each instruction into its fundamental tokens, which are then mapped to their corresponding embeddings. These embeddings constitute the input of the first LSTM layer. The output of all the instruction LSTM layers are subsequently passed to the second LSTM layer, which ultimately produces the final throughput prediction[2]. The model architecture is presented in Figure 2.2. Even though the objective of Ithemal is throughput prediction instead of energy prediction, elements of its neural network architecture can be adapted for our case.

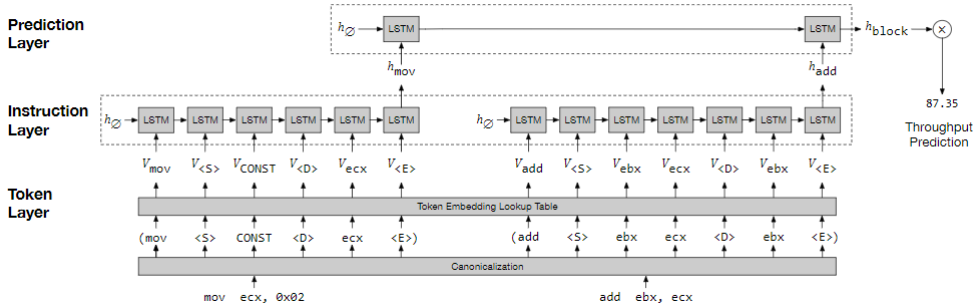


Figure 2.2: *Ithemal model architecture[2]*

Chapter 3

Background

In this section, we will analyze in depth the theoretical background underlying the tools and techniques employed in the development of this thesis. Initially, we shall examine the ways in which the code can be represented in order to be utilized by machine learning approaches. Subsequently, classical regression algorithms will be presented, followed by a comprehensive introduction to deep neural networks. Finally, we shall investigate the Optuna hyperparameter optimization framework.

3.1 Code Representation

Data can be broadly classified into two categories: structured and unstructured data. Structured data is organized in a well-defined format, following a pre-defined data model or schema. A customer database constitutes an example of structured data, as each data instance is characterized by specific attributes such as the customer name, address, phone number, and so forth. In contrast, unstructured data, such as images, texts, audio recordings, and so on, does not conform to a specific schema presenting challenges in terms of processing and analysis. This free-form nature of unstructured data requires advanced techniques for effective extraction and utilization of information.

3.1.1 The Notion of Word Embedding

In this scope, a plethora of advanced methodologies have been established for the various types of unstructured data. Images, for instance, are usually represented as three-dimensional arrays consisting of three different channels: red, green, blue. Each of these channels is characterized by a two-dimensional array of pixel values. In the text data scenario, one of the pioneering and elementary representations was the one-hot encoding approach, where a vocabulary is created and each word is mapped to a vector with dimension equal to the size of the vocabulary. The elements of this vector are set to zero, except for the element corresponding to the index of the word in the vocabulary. The document is then represented as the concatenation of all its word vectors (Figure 3.1).

Bag of words vectorization constitutes an extension of this method, as it represents

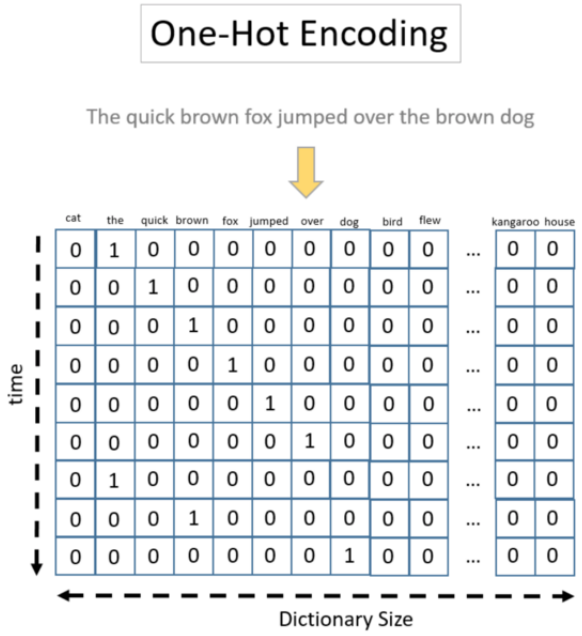


Figure 3.1: *Document representation as the concatenation of one-hot word encodings*

each document as a vector that captures the frequency of occurrence of each word in the document. The vector dimensions correspond to the vocabulary words, resulting in a high-dimensional and sparse vector (Figure 3.2α′). Term Frequency-Inverse Document Frequency (TFIDF) vectorization, offers a more sophisticated approach to occurrence-based representation. Similar to bag of words vectorization, the TFIDF vector dimensions respond to the vocabulary words. However, in this approach, each dimension value equals to the corresponding word's TFIDF score, which is calculated as the product of two factors: the Term Frequency (TF) and the Inverse Document Frequency (IDF) (Eq. 3.1). The TF of a word is the number of times it appears in the document, while the IDF measures how rare the word is in the corpus, as it is computed by the logarithm of the total number of documents in the corpus divided by the number of documents that contain the word [13]. An example of TFIDF-vectorized document is presented in Figure 3.2α′.

$$w_{i,j} = tf_{i,j} x log(\frac{N}{df_i}) \tag{3.1}$$

where:

$w_{i,j}$: score of word i in document j
$tf_{i,j}$: term frequency of word i in document j
df_i : number of documents containing word i
N : number of documents

These methods though, lack the capability to capture and semantic information in the text, triggering extensive research towards meaningful text representation. To address this issue, the notion of word-embedding is introduced as a high dimensional word

Count Vectorizer

	blue	bright	sky	sun
Doc1	1	0	1	0
Doc2	0	1	0	1

(α′) *Documents vecorized base on Bag of Words*

TD-IDF Vectorizer

	blue	bright	sky	sun
Doc1	0.707107	0.000000	0.707107	0.000000
Doc2	0.000000	0.707107	0.000000	0.707107

(β′) *Documents vecorized base on TFIDF*

Figure 3.2: *Bag of Words and TFIDF vectorization examples*

vector that incorporates its semantics. Thus, words concerning similar concepts are mapped to vectors with close proximity in the high dimensionality vector space (Figure 3.3), whereas vectors of words that are not related are far apart[14].

Machine learning networks are utilized to create embedding vectors rich in semantic information. The landmark paper ”Attention is All You Need”, proposed a Language Model (LM) based on deep neural network architecture, named Transformer, which employs multi-head attention blocks to capture word semantics[15]. The BERT architecture (Figure 3.4) constitutes the base Transformer architecture model, and presents the Mask Language Model (MLM) training technique. This technique is based on masking a random word of a sentence and training the model to finding the missing word. This approach results in the creation of trained word vectors with high semantic value[16].

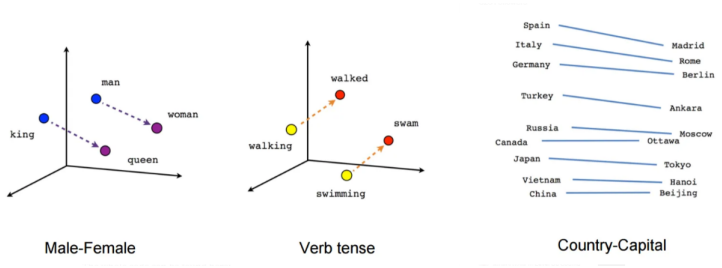


Figure 3.3: *Two dimensional projection of word embeddings concerning similar concepts*

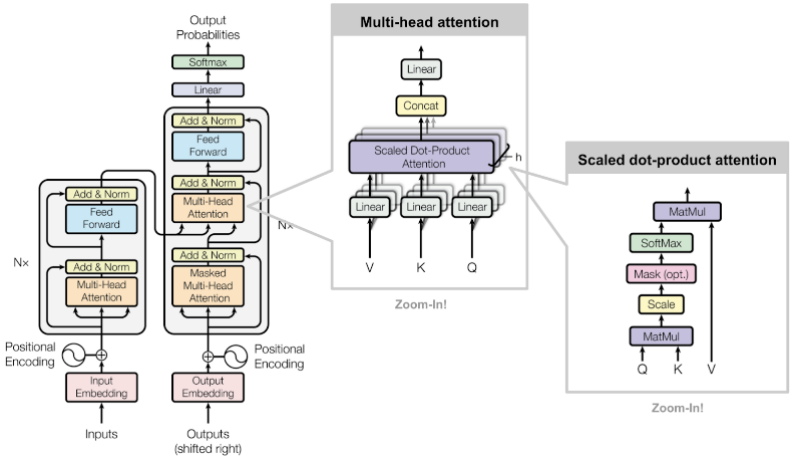


Figure 3.4: *BERT architecture with softmax layer producing word probabilities for MLM training*

3.1.2 Assembly Instruction Embeddings

Similar to natural language, code instructions are a form of unstructured data. Thus an appropriate representation must be established in order to capture the semantics of x86 Assembly instructions. Several different approaches have been developed that aim to create a vector representation suitable for the Assembly instructions.

The DEEPVSA methodology is an innovative approach for investigating the correlation between software failures and assembly instructions. It leverages instruction semantics and dependencies to achieve its goals. To retrieve the necessary information, a vocabulary with assembly instruction bytes as tokens, is introduced. Each instruction is partitioned into the corresponding vocabulary tokens, which are then converted to one-hot encodings. Subsequently, the one-hot encodings serve as the input to a LSTM layer, producing the instruction embedding. It is important to note, that the DEEPVSA does not generate the instruction embeddings but uses them in a latent stage of the network for the ultimate goal of software crash prediction[26].

Instruction2vec constitutes an Assembly instruction vectorization tool, developed to provide an instruction vector representation that is used to classify whether a function has software weaknesses or not. The Instruction2vec vectorization is based on creating a 9-dimensional vector for each instruction, utilizing the fact that most of the Assembly instructions structure follow include one opcode and two operands. The first dimension of the instruction vector is used for the opcode, while dimensions 2-5 and 6-9 correspond to operands 1 and 2, respectively[3]. The Instruction2vec approach is presented in Figure 3.5.

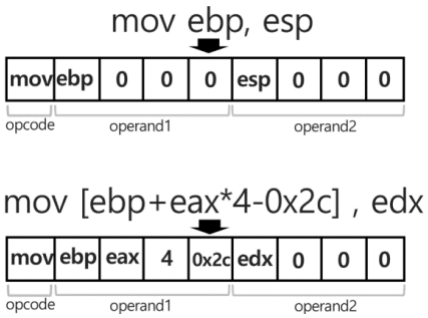


Figure 3.5: *Instruction2vec vectorization[3]*

The notion of a basic block embedding consists an extension of the idea of instruction embedding. Work [4] focuses on capturing semantics of basic blocks, with the aim of producing embeddings that are close for blocks of similar semantics and far apart for blocks with no relevance. This complex task is achieved through a two-step process. In the first step, the instruction embedding generated using the word2vec architecture[27]. The instructions of each function are preprocessed and are fed vectorized to the word2vec model which outputs the instruction embedding matrix denoted by **W**. The rows of **W** correspond to the instruction embeddings, while the columns to the embedding dimensions. The precise calculation of **W** is presented in Figure 3.6, where $\pi(B_i^{(j)})$ denotes the preprocessed *i* instruction of *j* basic block. The second step involves

utilizing the instruction embeddings for each block to generate the final basic block embedding, achieved through an LSTM network. The training process for the basic block embeddings relies on a dataset of basic block similarities, which contains pairs of basic blocks as features and their corresponding similarity as label. The similarity of each pair is identified as the distance between the embeddings (Figure 3.7).

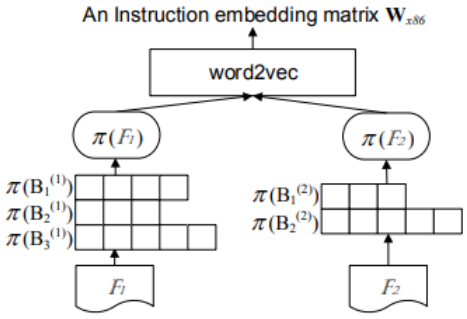


Figure 3.6: ***W*** *matrix computation [4]*

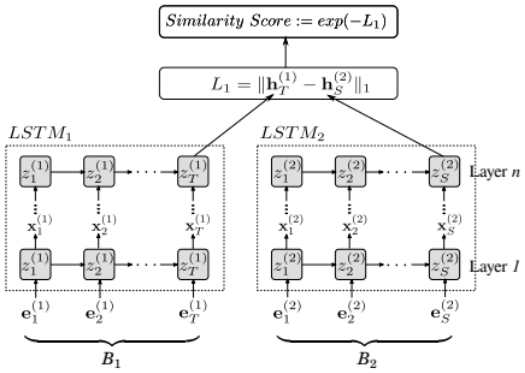


Figure 3.7: *Basic block embeddings training approach [4]*

Work [5] proposes an innovative model architecture known as Asm2Vec, with aims of producing both instruction and function embeddings simultaneously. The instructions are split to opcodes and up to two operands. Given context by neighboring instructions and by the function vector that includes the instruction, the Asm2Vec model trains the opcode and operand vectors and generates the instruction embedding by concatenating these vectors (Figure 3.8).

PalmTree presents a novel approach that aims to create an Assembly Instruction LM, based on the BERT transformer architecture[16]. To achieve this objective, the inputs to the BERT model consist of pairs of instructions that follow two distinct strategies: data flow pairs and control flow pairs. These pairs are established using the data dependency and the control flow relations that are identified through the binary disassembly process. Like the other methods, each instruction is split into its basic token, which in addition to opcodes and operands, also includes a CLS token that denotes the beginning of the input and a SEP token that indicates the start of a new instruction. These tokens are then fed into the BERT model to generate the instruction embedding, which is achieved through

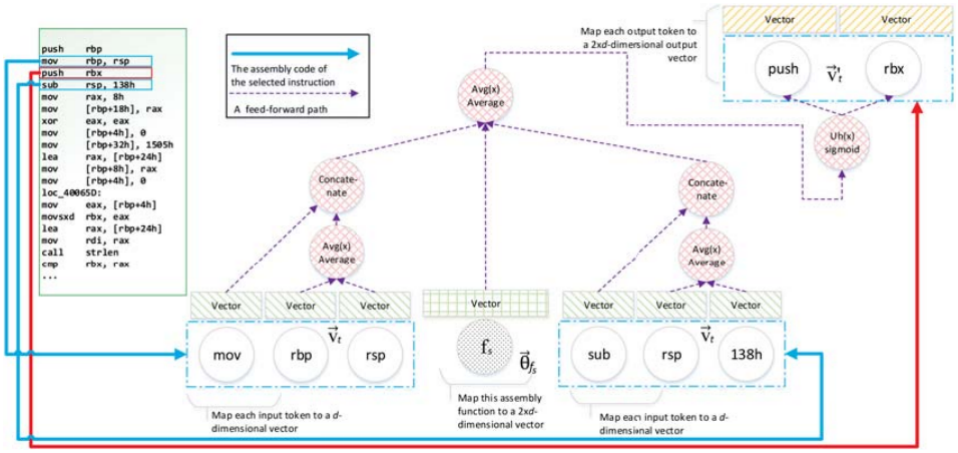


Figure 3.8: *Asm2Vec model architecture [5]*

the mean pooling of the hidden states of the second last layer of the BERT model. In order to train the BERT model for the Assembly Instructions and ultimately get embeddings that effectively capture the instruction semantics, PalmTree incorporates three distinct training tasks. The first one consist of MLM training, imitating the way that the original BERT LM was trained. In this case, a random token of each instruction is masked or replaced by a incorrect one, with the training process aiming to predict the correct token 3.9. Context Window Prediction (CWP) constitutes the second training task, which seeks to capture contextual relations between instructions. This is achieved by creating a binary classification problem with the output indicating whether two instructions can co-occur within the same context window or not. Finally, the third training task is the Def-Use Prediction (DUP), which incorporates data dependency information across instructions. In detail, DUP predicts the probability that a pair of instructions is swapped or not, aiming to learn which instruction has to be first. After the successful completion of these three training tasks, the BERT model is converted to an Assembly Instruction LM[6]. Figure 3.10 presents the PalmTree approach.

Figure 3.9: *MLM in PalmTree [6]*

3.2 Traditional Regression Techniques

With the remarkable ascent of deep learning, conventional machine learning algorithms have been eclipsed in recent years. Nevertheless, they can still be highly suitable for certain applications and may even surpass more intricate approaches. In any event, they merit exploration for generating baseline results that typically do not demand high computational power. Therefore, a comprehensive analysis of the most popular traditional supervised regression algorithms will be presented.

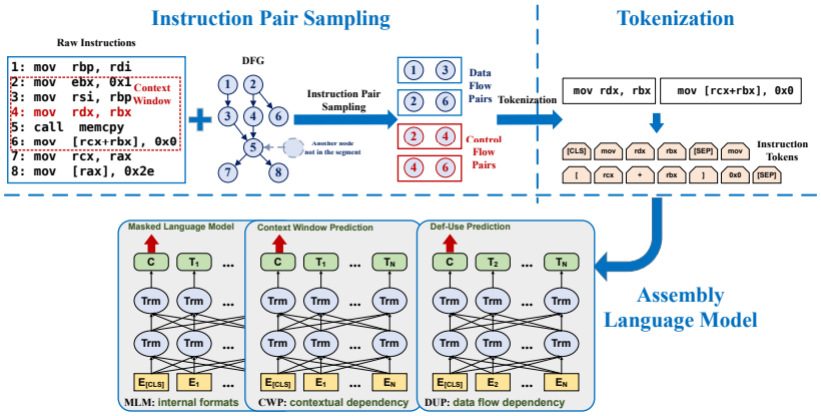


Figure 3.10: *PalmTree architecture [6]*

3.2.1 Linear Regression

Linear regression serves as the foundation of all the regression models developed over the years. Simple linear regression involves a singular input and output variable, with the aim of modeling the data according to Eq. 3.2. Meanwhile, multiple linear regression arises when there are more than one input variable, and the output representing a linear transformation of all the inputs (Eq. 3.3). The prediction of the linear model is denoted as $\hat{y}$. The coefficients of the linear model are trained to minimize the residual sum of square between the observed targets in the dataset and the targets predicted by the linear approximation (Eq. 3.4)[17]. In Figure 3.11 an example of linear regression modeling with two input variables is presented.

$$\hat{y}_i = \beta_0 + \beta_1 x_i, i = 1, ..., n \tag{3.2}$$

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon_i, i = 1, ..., n \tag{3.3}$$

where:

i : number of data instance
y_i : output for i_{th} data instance
x_{ij} : input variable j for i_{th} data instance
β_j : linear model coefficient for j input variable
ϵ_i : error for i_{th} data instance
p : total number input variables
n : total number of data

$$L(\beta) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_\beta(x_i) - y_i)^2 \tag{3.4}$$

where:

i : number of data instance
$L(\beta)$: least squares loss function
y_i : label of i_{th} data instance
x_i : input of i_{th} data instance
$\hat{y}_\beta$: linear function prediction
n : total number of data

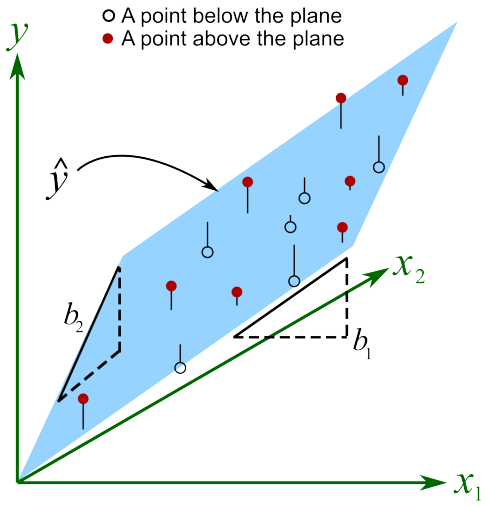


Figure 3.11: *Linear Regression example*

3.2.2 Lasso Regression

Lasso Regression is the same as Linear Regression but trained with an additional L1 norm regularizer in the loss function. Accordingly, the lasso loss function is defined as shown in Eq. 3.5. Lasso regression penalizes large coefficients and thus helps to reduce the model complexity and multi-collinearity. The penalty increases in direct proportion to the value of the α[18].

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_\beta(x_i) - y_i)^2 + \alpha\sum_{j=1}^{p}|\beta_j| \tag{3.5}$$

where:

α : L1 regularization term multiplier.

3.2.3 Ridge Regression

Ridge Regression, same as Lasso Regression, is an extension of Linear Regression. Unlike Lasso Regression, Ridge Regression penalizes large coefficients by incorporating the square of the coefficients in the loss function. (Eq. 3.6). Therefore, ridge regression penalizes large coefficients even more.In this case, α is the L2 regularization term multiplier[19].

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_\beta(x_i) - y_i)^2 + \alpha\sum_{j=1}^{p}\beta_j^2 \tag{3.6}$$

where:

α : L2 regularization term multiplier.

3.2.4 ElasticNet Regression

ElasticNet Regression proposes a combination of Lasso and Ridge loss functions, as it adds both the L1 and the L2 regularization terms to the least squares loss (Eq. 3.7). Consequently, the optimization process involves adjusting the L1 and L2 regularization multiplier constants, denoted as λ_1 and λ_2, respectively, which results in an effective trade-off between the L1 and L2 regularization methods[20].

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_\beta(x_i) - y_i)^2 + \lambda_1\sum_{j=1}^{p}|\beta_j| + \lambda_2\sum_{j=1}^{p}\beta_j^2 \tag{3.7}$$

where:

λ_1 : L1 regularization term multiplier
λ_2 : L2 regularization term multiplier

3.2.5 Stochastic Gradient Descent (SGD) Regression

Stochastic Gradient Descent (SGD) consists a training approach that in each step a random data sample is randomly chosen and the weights of the model are updated based on the gradient of the weights and the loss function (Eq. 3.8). Due to the random data picking, this method scales good with big data and it can process new examples on the fly in a deployed system[21].

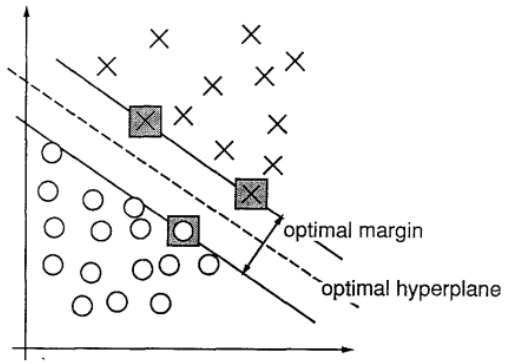
$$\beta_{t+1} = \beta_t - \gamma\nabla_\beta L(f(x_i, b_t), y_i)) \tag{3.8}$$

where:

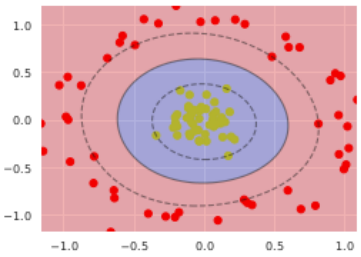
γ : learning rate
x_i, y_i : random data sample
f : model function to map x to y
β_{t+1} : updated model weights
β_t : current iteration's weights
L : loss function

3.2.6 Support Vector Regression (SVR)

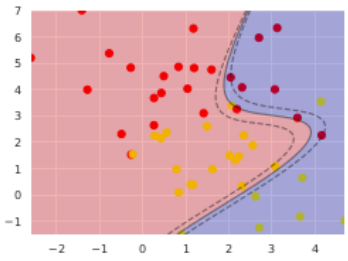
Support Vector Regression (SVR) consists a generalization of the Support Vector Machine (SVM) algorithm, which was initially designed for tackling binary classification problems. SVMs approach binary classification by calculating the optimal hyperplane, also known as decision boundary, which lies at the middle of two hyperplanes defined by the closest data points of each class, known as support vectors (Figure 3.12α′. The optimal hyperplane is the one that maximizes the margin between the support vectors. During the hyperplane optimization, SVMs incorporate kernels that map the data into higher dimensionality, making them easier to be separated. Remarkably, SVMs with radial basis function (RBF) and polynomial kernels (Figures 3.12β′,3.12γ′) can provide great solutions for non-linear cases [28][7].



(α′) *SVM with linear kernel[28]*



(β′) *SVM with rbf kernel*



(γ′) *SVM with polynomial kernel*

Figure 3.12: *SVMs with different kernels*

In the context of SVR, the goal is to identify a function that exhibits a deviation of at most ϵ from the data labels. The simple case of a linear function is presented in Eq. 3.9, while the optimal hyperplane is found according to Eq. 3.10.

$$f(x) =< w, x > +b \tag{3.9}$$

$$\begin{aligned} &minimize\ \frac{1}{2}||w||^2 \\ &subject\ to \begin{cases} y_i - < w, x_i > -b & \leq \epsilon \\ < w, x_i > +b - y_i & \leq \epsilon \end{cases} \end{aligned} \tag{3.10}$$

where:

$f(x)$: model function
w : function weights
β : bias
y_i : i_{th} data label
x_i : i_{th} data input
ϵ : error margin

However, it is often necessary to introduce a soft margin that permits some degree of errors, in order to achieve higher performance by excluding outliers. This soft margin is applied by using slack variables $\xi, \xi*$ that modify the admissible error rate, alongside a constant C that determines the trade-off between the flatness of f and the extent of permissible deviations beyond ϵ. The soft margin approach is described by Eq. 3.11[22][7]. Figure 3.13 presents the regression solution using SVR algorithm with linear kernel and soft margin.

$$\begin{aligned} &\textit{minimize } \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}(\xi + \xi*) \\ &\textit{subject to } \begin{cases} y_i - < w, x_i > -b & \leq \epsilon + \xi \\ < w, x_i > +b - y_i & \leq \epsilon + \xi* \\ \xi, \xi* & \geq 0 \end{cases} \end{aligned} \tag{3.11}$$

where:

ξ_i, ξ_i* : slack variables
C : constant

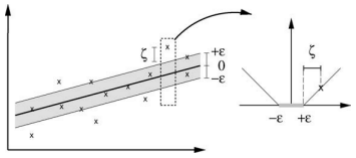


Figure 3.13: *Linear SVR with soft margin[7]*

3.2.7 Histogram-Gradient Boosting Regressor

Histogram-Gradient Boosting consists a technique aimed at accelerating the training of decision trees used in gradient boosting. Gradient boosting proposes a robust methodology for building an ensemble of decision trees that iteratively improves the accuracy of the model by minimizing the function loss - most commonly the mean squared error[29]. The weights of the tree are updated using gradient descent (Eq. 3.12).

$$w_{t+1} = w_t - \gamma \sum_{i=1}^{n} \nabla_w L(f(x_i, w_t)), y_i) \tag{3.12}$$

where:

γ : learning rate
x_i, y_i : i^{th} data instance
w_t : current tree weights
w_{t+1} : updated tree weights
L : loss function

During gradient boosting, decision trees are created and added sequentially in every iteration, increasing considerably the computational cost. The complexity can be greatly reduced if the number of possible values for the input features is reduced. Incorporating the histogram of data input values tailors the value reduction achieving great speed-up with minimal - if any - impact on model accuracy[30].

3.3 Deep Neural Networks

In recent years, the field of machine learning has experienced a significant shift towards deep learning, largely attributed to the unprecedented surge in available computational resources. Deep Neural Networks (DNNs) enable building large robust models that, when tuned correctly, they can perform exceptionally well capturing high detail dependencies between data input and output. DNNs can be broadly classified into four distinct categories: Dense Neural Networks, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers. For the purposes of this thesis, our investigation will be focused on Dense Neural Networks and Long Short-Term Memory (LSTM) networks, which constitute a specialized subset of RNNs.

3.3.1 Gradient Descent

To be able to comprehend the way that the deep neural networks function, some basic deep learning concepts should be analyzed first. Gradient descent consists the foundation of neural network training. Consider a model function f, designed to predict the ground truth of a task based on an input x (Eq. 3.13), and a loss function L, which measures the discrepancy between the prediction $f(x)$ and the actual value y_{true}.

$$f(x) = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{3.13}$$

where:

$f(x)$: model function
$\mathbf{W}$: weights matrix
$\mathbf{x}$: input matrix
$\mathbf{b}$: bias matrix

Gradient descent, proposes to train the weights and the biases of the model by updating them based on their partial loss derivatives (Eq. 3.14, 3.15). It is should be noted,

that at the beginning of the training process, both the weights and the biases are randomly initialized[31]. The intuition underlying the gradient descent algorithm can be captured by Figure 3.14. We can see that with each training step the loss function comes approaches to its global minimum.

$$W_t = W_{t-1} - \alpha \frac{\partial L}{\partial W_{t-1}} \tag{3.14}$$

$$b_t = b_{t-1} - \alpha \frac{\partial L}{\partial b_{t-1}} \tag{3.15}$$

where:

w_t : new weights
w_{t-1} : old weights
α : learning rate
$\frac{\partial L}{\partial W}$: partial derivative of loss function with respect to weights
b_t : new bias
b_{t-1} : old bias
$\frac{\partial L}{\partial b}$: partial derivative of loss function with respect to bias

The weight and bias derivatives are computed using the chain of differentiation rule (Eq. 3.16,3.17).

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial f(x)} * \frac{\partial f(x)}{\partial W} \tag{3.16}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial f(x)} * \frac{\partial f(x)}{\partial b} \tag{3.17}$$

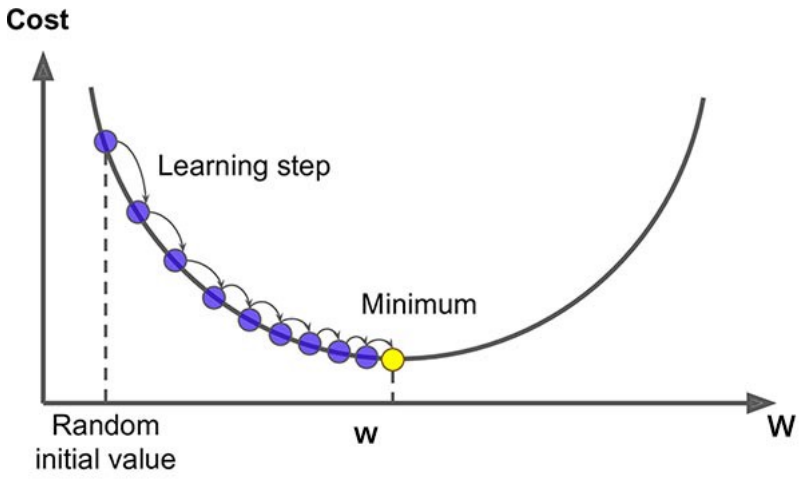


Figure 3.14: *Gradient Descent algorithm intuition*

The Vanishing and Exploding Gradients problems

The challenges of vanishing and exploding networks in the course of training of a DNN, were proposed in the work [32]. For deep neural networks, the input to each layer

is derived from the output of its preceding layer, while its own output consists the input to the next layer, thus the prediction of the model can be generally described by Eq. 3.18.

$$\hat{y} = w^L w^{L-1} ... w^2 w^1 x \tag{3.18}$$

where:

$\hat{y}$: output of the DNN
w^i : weights of the layer i
L : number of layers
x : model's input

The process of weight multiplication employed in DNNs can result in extremely high or low output values. For instance, if the weights are less than 1, the output becomes exceedingly small, while if the weights are higher than 1, the output approaches infinity. The same issue rises for the derivative computation as well. To address the challenge of exploding gradients, the technique of gradient clipping has been proposed as a solution[33]. This method involves setting the gradients to a predefined threshold value if they exceed this threshold during training.

Optimizers

When using the gradient descent algorithm, the training steps may not approach the cost minimum directly, resulting in slower convergence and prolonged the training time. To mitigate this issue, the momentum algorithm proposes the utilization of an exponentially weighted average of gradients, which are then employed to update the weights instead [34]. In order to incorporate the momentum technique in the gradient descent, the velocities of the derivative are introduced (Eq. 3.19) and the weights and biases are computed according to Eq. 3.20. The velocities are initialized as 0. Figure 3.15 presents the convergence acceleration when using momentum, utilising the contour visualization.

$$\begin{aligned} V_{dw,t} &= \beta V_{dw,t-1} + (1-\beta)\frac{\partial L}{\partial w_{t-1}}, \\ V_{db,t} &= \beta V_{db,t-1} + (1-\beta)\frac{\partial L}{\partial b_{t-1}} \end{aligned} \tag{3.19}$$

$$\begin{aligned} w_t &= w_{t-1} - \alpha V_{dw,t}, \\ b_t &= b_{t-1} - \alpha V_{db,t} \end{aligned} \tag{3.20}$$

where:

$V_{dw,t-1}$: old weight velocity
$V_{db,t-1}$: old bias velocity
$V_{dw,t}$: new weight velocity
$V_{db,t}$: new bias velocity
β : momentum, generally set as 0.9

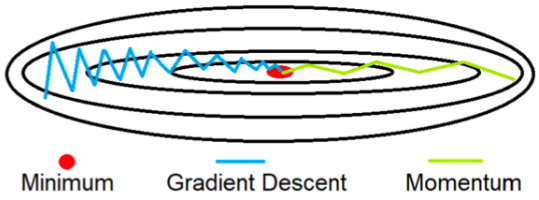


Figure 3.15: *Gradient descent acceleration using momentum*

RMSprop presents an optimization technique by automatically adapting the learning rate. The term S is introduced for the adaptation of learning rate and it is defined by Eq. 3.21 (initially set to 0). The weights and biases are subsequently updated based on Eq. 3.22[35]. To avoid division by zero, the parameter ϵ, represented by a very small value, is incorporated into the denominator.

$$\begin{aligned} S_{dw,t} &= \beta S_{dw,t-1} + (1-\beta)(\frac{\partial L}{\partial w_{t-1}})^2, \\ S_{db,t} &= \beta S_{db,t-1} + (1-\beta)(\frac{\partial L}{\partial b_{t-1}})^2 \end{aligned} \tag{3.21}$$

$$\begin{aligned} w_t &= w_{t-1} - \frac{\alpha}{\sqrt{S_{dw,t}} + \epsilon} \frac{\partial L}{\partial w_{t-1}}, \\ b_t &= b_{t-1} - \frac{\alpha}{\sqrt{S_{db,t}} + \epsilon} \frac{\partial L}{\partial b_{t-1}} \end{aligned} \tag{3.22}$$

where:

$S_{dw,t-1}$: old weight S
$S_{db,t-1}$: old bias S
$S_{dw,t}$: new weight S
$S_{db,t}$: new bias S
β : rms term, generally set as 0.999
ϵ : small number, generally set to 10^{-8}

Adaptive Moment Estimation (Adam) algorithm is introduced as an improvement of RMSProp and momentum algorithms, by combining them. Adam has gained immense popularity due to its consistent capacity to generate optimal outcomes in the majority of cases. Adam utilizes a corrected version of momentum velocity terms and the RMSProp S (Eq. 3.23)[36]. Finally, the model parameters are optimized following the Eq. 3.24.

$$
\begin{aligned}
V_{dw,t} &= \beta_1 V_{dw,t-1} + (1-\beta_1)\frac{\partial L}{\partial w_{t-1}}, \\
V_{db,t} &= \beta_1 V_{db,t-1} + (1-\beta_1)\frac{\partial L}{\partial b_{t-1}}, \\
S_{dw,t} &= \beta_2 S_{dw,t-1} + (1-\beta_2)(\frac{\partial L}{\partial w_{t-1}})^2, \\
S_{db,t} &= \beta_2 S_{db,t-1} + (1-\beta_2)(\frac{\partial L}{\partial b_{t-1}})^2,
\end{aligned}
\tag{3.23}
$$

$$
\begin{aligned}
V_{dw(corrected)} &= \frac{V_{dw,t}}{1-\beta_1^2}, \\
V_{db(corrected)} &= \frac{V_{db,t}}{1-\beta_1^2}, \\
S_{dw(corrected)} &= \frac{S_{dw,t}}{1-\beta_2^2}, \\
S_{db(corrected)} &= \frac{S_{db,t}}{1-\beta_2^2}
\end{aligned}
$$

$$
\begin{aligned}
w_t &= w_{t-1} - \frac{\alpha}{\sqrt{S_{dw(corrected)}} + \epsilon} V_{dw(corrected)}, \\
b_t &= b_{t-1} - \frac{\alpha}{\sqrt{S_{db(corrected)}} + \epsilon} V_{db(corrected)}
\end{aligned}
\tag{3.24}
$$

where:

$V_{corrected}$: the corrected momentum velocities
$S_{corrected}$: the corrected RMSProp S
β_1 : momentum, generally set as 0.9
β_2 : RMSProp term, generally set as 0.999

Learning Rate Decay

During the training process of a DNN, it is common for oscillations to occur around the optimal result of the loss function. This happens, due to the large value of learning rate, which lacks the capability of making smaller steps to approach the minimum cost. On the other hand, a very small learning rate would require a great deal of time in order to train the model. Therefore, the concept of learning rate decay is proposed. When using learning rate decay, the training begins with a relatively high learning rate, which decreases every a predefined number of epochs,enabling the model to approach the optimal outcome more efficiently. This way, there is no unnecessary steps during the initial phases of training, and the model converges closer to the minimum point of teh loss function[37]. The effect of the use of learning rate decay can be depicted if Figure 3.16.

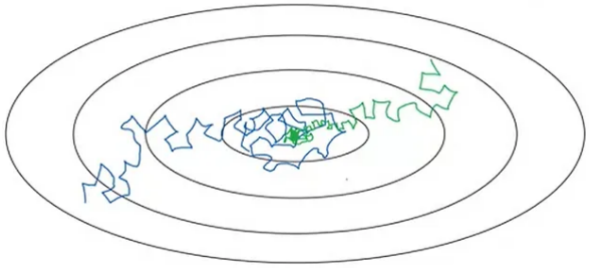


Figure 3.16: *Gradient descent with (green line) and without (blue line) learning rate decay*

Batching the data

The last of the basic machine learning concepts that will be analyzed regards batch size. Batching refers to the even partitioning of data into smaller subsets, and it can have a significant impact on both training time and model accuracy. If there is no data batching, the training of the model requires a great deal of time and computer memory, especially when dealing with large datasets. On the other side of the coin, if the batch size is too small, meaning that data are split into many subsets, the training process becomes more noisy and is more difficult for the model to converge. Subsequently, a well-tuned batch size helps the model to converge to better results[38].

3.3.2 Dense Neural Networks

Dense Neural Networks, often referred to as Fully Connected (FC) Neural Networks (Figure 3.17), consist the most basic and simple form of neural networks. They contain a series of FC layers with numerous neurons. Each neuron constitutes a computational unit that is connected with all the neurons of the preceding layer and is identified by trainable weights - one for each edge connected to the neuron - and a bias. Forward propagation (Eq. 3.25) is utilized to calculate the output of each layer.

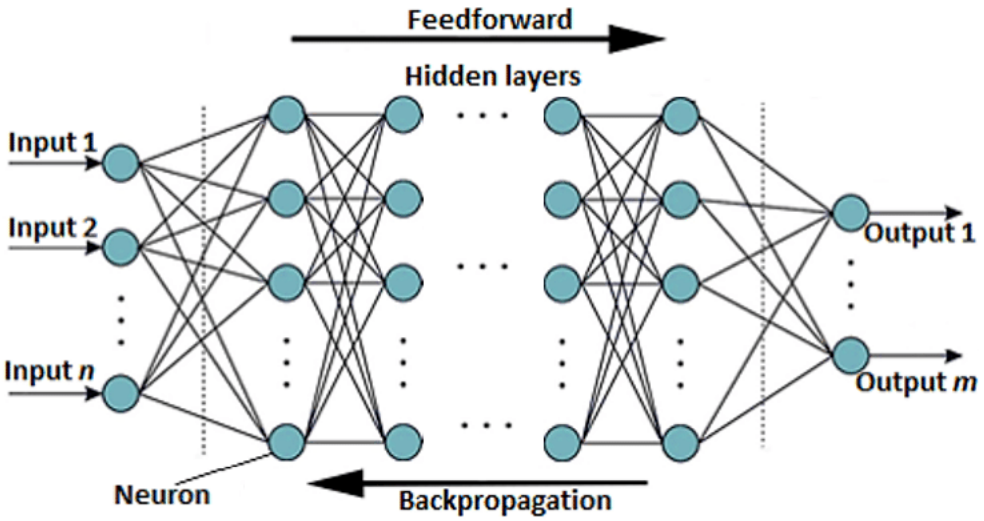


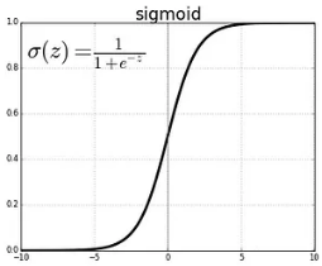
Figure 3.17: *Example of dense neural network*

$$y_i = activation(\mathbf{W_i^T} * y_{i-1} + b_i) \tag{3.25}$$

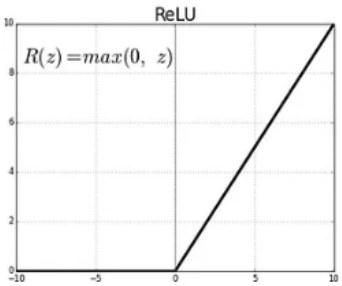
where:

y_i	: output of layer i
$\mathbf{W_i^T}$	: weights matrix for layer i
y_{i-1}	: output of layer i-1
w_i	: biases of layer i
activation	: activation function

The dimension of the output of each layer is equivalent to the number the neurons of the layer, while the input of the network is considered as the first layer. The activation function provides non-linearity and is usually a sigmoid (Figure 3.18α′) or a rectified linear unit (ReLU) (Figure 3.18β′) function[31].



(α′) *Sigmoid activation function*



(β′) *ReLU activation function*

Figure 3.18: *Common activation functions*

The training of the weights and biases is accomplished through the utilization of backward propagation (Eq. 3.26,3.27), which is based on the gradient descent algorithm.

$$W_{i,t} = W_{i,t-1} - \alpha \frac{\partial L}{\partial W_{i,t-1}} \tag{3.26}$$

$$b_{i,t} = b_{i,t-1} - \alpha \frac{\partial L}{\partial b_{i,t-1}} \tag{3.27}$$

where:

$w_{i,t}$	: new weights of layer i
$w_{i,t-1}$	: old weights of layer i
α	: learning rate
$\frac{\partial L}{\partial W}$	: partial derivative of loss function with respect to weights
$b_{i,t}$	: new bias of layer i
$b_{i,t-1}$	: old bias of layer i
$\frac{\partial L}{\partial b}$	: partial derivative of loss function with respect to bias

In this case, the derivatives are computed using the chain rule of differentiation, utilizing the output of each layer (Eq. 3.28, 3.29).

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i} * \frac{\partial y_i}{\partial W_i} \tag{3.28}$$

$$\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial y_i} * \frac{\partial y_i}{\partial b_i} \tag{3.29}$$

where:

y_i : the output of layer i

3.3.3 Long Short Term Memory Networks (LSTMs)

RNNs are used for tackling machine learning problems where the input exhibits a sequential nature. Some examples of sequential problems include language translation and speech recognition. Sequential problems can be broadly classified into three distinct categories based on the length of the input and output sequences: one-to-many (e.g. text completion based on one word), many-to-one (e.g. voice sentiment classification), and many-to-many (e.g. language translation). The case of one-to-one is deemed obsolete as there is no sequential dependency. Moreover, the case of many-to-many can be split into two categories, based on whether the output sequence frames correspond to the output of each cell or not (Figure 3.19).

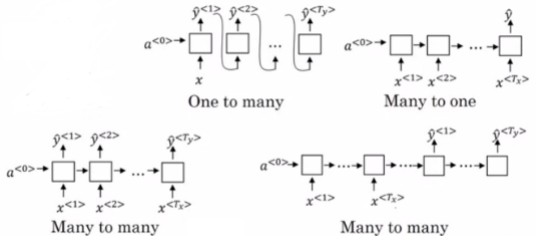


Figure 3.19: *Categories of sequential problems*

Probably the most sophisticated category of RNNs are the Long Short Term Memory Networks (LSTMs), which were first introduced by [23] in 1997. LSTMs are capable of learning long-term dependencies by enabling the passage of sequence information throughout the entirety of the sequence. The basic building component of LSTMs are the LSTM cells (Figure 3.20), which are usually comprised of three gates: the forget gate, the input gate and the output gate (Figure 3.21)[39]. The primary inputs to an LSTM cell are the frame of the sequence, the hidden state and the cell state, denoted by x_t, h_{t-1} and c_{t-1} respectively. The hidden state represents the output of the previous cell, whereas the cell state encapsulates the information passed along by the sequence preceding the cell. Subsequently, each cell outputs the updated cell state as well as the new hidden state. The mathematical components of the LSTM cell are described by the 3.30 equations, and are trained using their corresponding loss derivatives during backwards propagation[9].

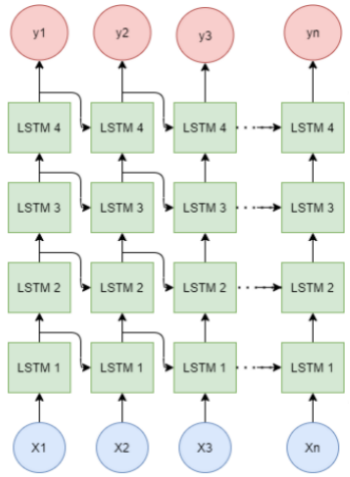


Figure 3.20: *Example of a multi-layered LSTM[8]*

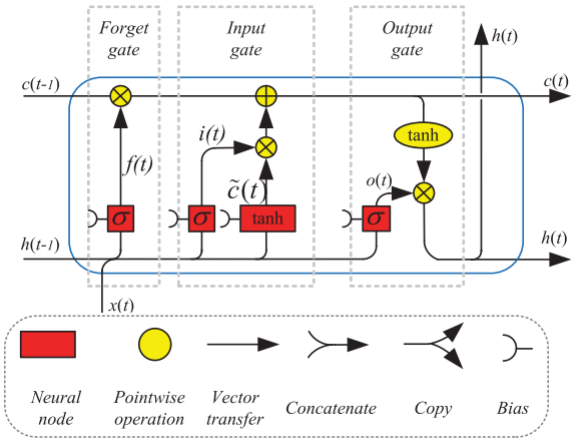


Figure 3.21: *LSTM cell[9]*

$$
\begin{aligned}
f_t &= \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f),\\
i_t &= \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i),\\
\widetilde{c}_t &= \sigma(W_{\widetilde{c}h}h_{t-1} + W_{\widetilde{c}x}x_t + b_{\widetilde{c}}),\\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \widetilde{c},\\
o_t &= \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o),\\
h_t &= o_t \cdot tanh(c_t)
\end{aligned}
\tag{3.30}
$$

where:

W : the corresponding weight matrices
b : the corresponding biases
σ : the sigmoid function

It should be noted that there are significant extensions to the presented LSTM, such as bidirectional LSTMs or LSTMs that incorporate peephole connections. However, as these extensions fall beyond the scope of the current work, a detailed analysis of these techniques shall be omitted.

3.3.4 Dropout

Deep neural networks that include a significant number of parameters often susceptible to overfitting, whereby they perform very well for the training set data, but struggle to generalize to new data. Dropout is proposed as a regularization technique, to enhance the robustness of deep neural networks. It can be applied on multiple types of layers, and it involves shutting off a small percentage of neurons within a given layer (Fig. 3.22). By doing so, the network is forced to adapt to be trained for a broader range of scenarios, improving its ability to generalize[10].

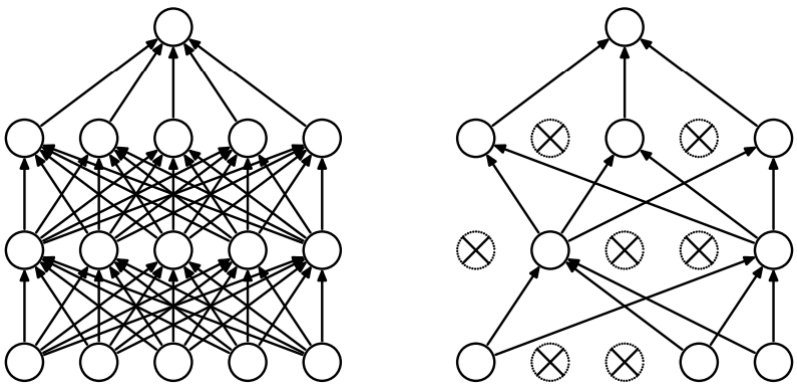


Figure 3.22: *DNN with (right) and without (left) dropout[10]*

3.4 Hyperparameter Optimization with Optuna

Hyperparameter optimization represents a crucial objective in the development of machine learning models. Both traditional machine learning algorithms and DNNs include a handful of parameters (e.g. gradient descent learning rate, number of dense layers, etc.), that require appropriate tuning to yield optimal models, posing a complex, multi-variable problem. Exhaustive techniques such as grid search are often deemed inefficient due to the exponential growth in cost as the number of parameters increases. Therefore, machine learning scientists should possess a thorough understanding of the role of each hyperparameter to efficiently limit the search space. Also, a sophisticated hyperparameter search framework should be employed to improve the efficiency and efficacy of the optimization process.

Optuna is a state-of-the-art hyperparameter optimization framework that leverages cutting-edge searching and pruning strategies to efficiently guide the search towards promising hyperparameter configurations, thereby avoiding the exploration of infeasible or sub-optimal regions. The search process is carried out with a view to minimizing or maximizing the metric of interest, as specified by the user (e.g. minimizing the validation loss). Each individual configuration that is evaluated during the search is referred to as a trial. Optuna also includes an easy to use Application Programming Interface (API), which enables seamless integration with widely-used machine learning frameworks, such as PyTorch, TensorFlow, and Scikit-learn. It also provides an additional dashboard that contains useful hyperparameter insights, such as the importance of each hyperparameter to the performance of the task at hand[11]. An overview of the system design of Optuna is in Figure 3.23.

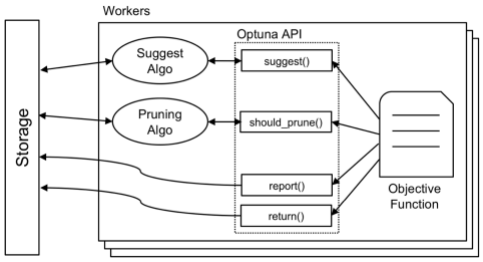


Figure 3.23: *Optuna's system design overview[11]*

The default algorithm for selecting hyperparameters in Optuna is the Tree-structured Parzen Estimator (TPE), which utilizes Bayesian optimization to efficiently search for optimal hyperparameters by modeling the search space as a probability distribution function[40]. As for the pruning component of the optimization process, Optuna proposes a novel algorithm, based on successive halving[41]. Figure 3.24 presents the implemented pruning algorithm. First, the algorithm computes the rung variable of the trial, which identifies the number of time the trial has survived the pruning. Subsequently, the trial investigation continues only if its provisional ranking resides within the top $\frac{1}{\eta}$[11].

Input: target trial `trial`, current step `step`, minimum resource r, reduction factor η, minimum early-stopping rate s.
Output: **true** if the trial should be pruned, **false** otherwise.
`rung` $\leftarrow \max(0, \log_\eta(\lfloor \texttt{step}/r \rfloor) - s)$
if `step` $\neq r\eta^{s+\texttt{rung}}$ **then**
  return false
end
`value` ←get_trial_intermediate_value(`trial`, `step`)
`values` ←get_all_trials_intermediate_values(`step`)
`top_k_values` ←top_k(`values`, $\lfloor |\texttt{values}|/\eta \rfloor$)
if `top_k_values` $= \emptyset$ **then**
  `top_k_values` ←top_k(`values`, 1)
end
return `value` $\notin$ `top_k_values`

Figure 3.24: *Pruning algorithm implemented in Optuna[11]*

3.5 Dataset

The basic block energy prediction, we present on the current work, was based on a custom developed dataset in the scope of the diploma thesis of Dimitris Bouras. This dataset encompasses approximately 560,000 basic blocks that serve as features, alongside their corresponding energies, which act as the labels. The basic blocks included in this dataset originate from 24 distinct programs that function as benchmarks.

The dataset creation process incorporated multiple sophisticated processes along with program instrumentation, in order for the energy of each basic block to be extracted. The first step of this process includes executing a benchmark alongside with RAPL reads at the start and end of the program, resulting in an unmodified energy calculation. The executable is also passed through an Low Level Virtual Machine (LLVM) instrumentation path in order for the execution trace and the energy file to be produced. The RAPL reads energy costs are being accounted for and subsequently combined with the energy file, subtracting the RAPL overhead from the energy measurements. The final step involves matching the energy measurements with their corresponding basic blocks and analyzing the library functions used in the program into basic blocks in order for their respective energy to be attributed. The basic energy unit is $61\mu J$ due to the Skylake architecture of the processor. The overview of the dataset creation is presented in Figure 3.25.

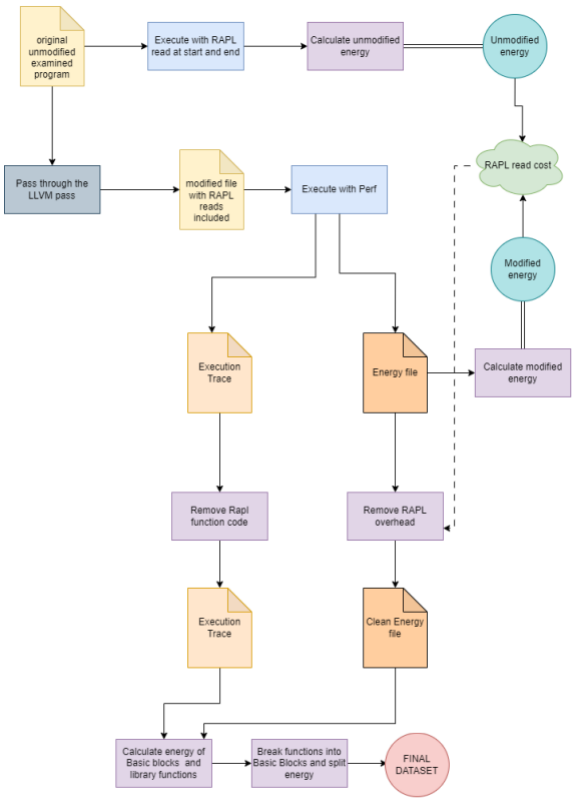


Figure 3.25: *Overview of the dataset creation process*

Chapter 4

Proposed Techniques/Designs & Developments

This chapter comprehensively presents the systems and approaches developed for the purpose of the current work. The novel developments include the intricate machine learning methods that were implemented to successfully tackle the task of basic block energy prediction, along with secondary tools that were built as utilities for the primary components. A general overview of the basic block prediction system is depicted in Figure 4.1, with the various models represented by the model black box.

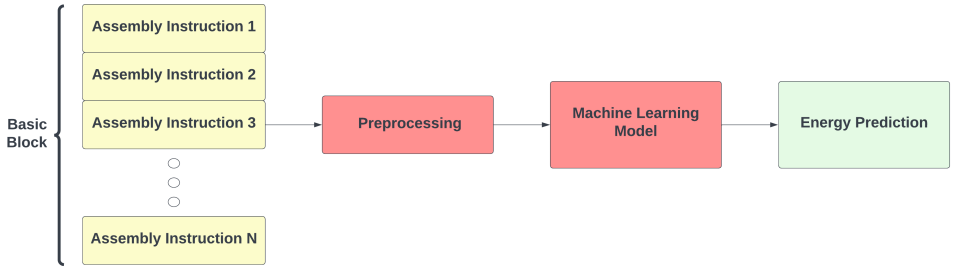


Figure 4.1: *Overview of the energy prediction system*

The rest of this chapter is organized as follows: The first section delves into the preprocessing concepts that were implemented based on the thorough exploration and analysis of the dataset. The remaining sections elaborate on the distinct approaches that were created and thoroughly investigated.

4.1 Data exploration and preprocessing

Assembly instructions exhibit a highly structured form, thus unlike free text they do not require much cleaning. The main preprocessing that was applied to the basic blocks was removing constants that correspond memory addresses (Figure 4.2). This step was motivated by the fact that the information associated with energy consumption cannot be reliably linked to memory addresses when the cache state is indeterminate. Additionally, PalmTree removes the memory addresses as well, during the tokenization[6] and

being in sync with PalmTree preprocessing is important for the instruction encoding that is employed in the approaches that utilize the PalmTree representation.

Figure 4.2: *Memory address removal example*

It is worth mentioning that the basic blocks utilized for training were filtered in order to eliminate outliers in terms of instruction length and associated energy labels. This decision was based on the dataset analysis findings presented in Section 5.1.

4.2 Energy Prediction using sklearn

The development of the traditional machine learning approaches, that were presented during the background theoretical analysis, was based on the sklearn framework. Sklearn provides a rich and a efficient machine learning library that offers a wide variety of implemented tools used for regression and classification tasks. Aside from the machine learning algorithms, sklearn integrates a comprehensive suite of end to end pipelines and metric functions, required for the training and the evaluation of these algorithms. The framework is utilized through an easy to use python API[42][43].

For the developments based on the sklearn, the initial code representation of choice entails the utilization of the bag of words methodology. Sklearn, employs a count vectorization technique that represents text input based on the bag of words technique. Since the bag of words representation is characterized by the sparse matrix it produces, the count vectorizer implemented, produces an efficient sparse representation, that greatly accelerates the training process. Subsequently, each basic block is represented as a bag of words, where each word corresponds to an Assembly token: an opcode, a register or an immediate (Figure 4.3).

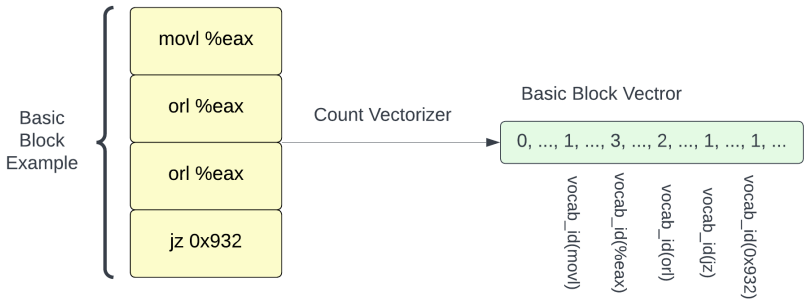


Figure 4.3: *Example of basic block representation as a bag of words*

Figure 4.4 depicts an end-to-end overview of the sklearn components utilized for the

training of conventional machine learning regressors, with the ultimate objective of predicting basic block energy. The initial step involve the splitting of the basic block dataset into a train and a test set, with the former being randomly sampled and constituting 90% of the dataset, while the latter comprises the remaining portion and is reserved for model evaluation purposes. Furthermore, the introduction of sklearn pipelines permits the incorporation of supplementary stages in the training process. Hence, the pipeline is trained a a whole. The possible steps of each pipeline are the following: TFIDF vectorization, Normalization, Standard Scaling, sparse to dense transformation and finally the machine learning regression algorithm.

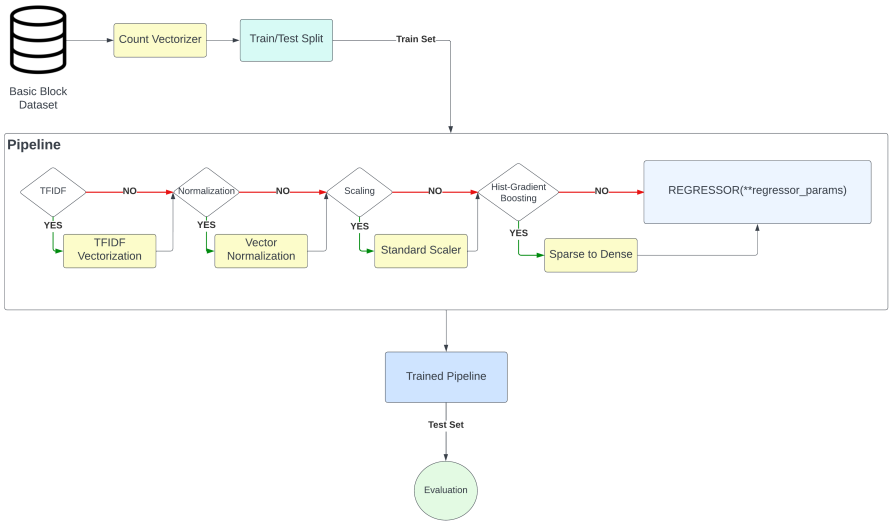


Figure 4.4: *Overview of the sklearn approach*

If a pipeline includes TFIDF vectorization, then the input vector is converted from the bag of words representation to the corresponding TFIDF representation, according to the respective scores to the Assembly tokens. Additionally, if Normalization is used, then the vectors are normalized so that the length of each vector is equivalent to one4.1. In work [44] it is demonstrated that this normalization technique can lead to faster convergence during training, improve generalization performance, and reduce the sensitivity of the network to the choice of learning rate. Moreover, we experiment with Standard Scaling which sets the mean of the vectors to zero and the variance of the vectors to one4.2. Standard scaling can potentially improve the performance of various machine learning algorithms, including logistic regression, support vector machines, and neural networks, by reducing the impact of outliers and improving the conditioning of the optimization problem. Vector normalization is applied for each feature independently, while standard scaling is applied across features. It is also noted, that in the case of the Histogram-Gradient Boosting Regressor, a sparse to dense transformation is required since the Gradient Boosting Regressor does not work with the sparse representation of sklearn as input.

$$x = \frac{x}{||x||} \tag{4.1}$$

$$
\begin{aligned}
z &= \frac{x - \mu}{\sigma}, \\
\mu &= \frac{1}{N}\sum_{i=1}^{N} x_i, \\
\sigma &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}
\end{aligned}
\tag{4.2}
$$

where:

- z : the transformed vector,
- x : the original vector,
- $||x||$: the norm of x,
- μ : the mean of the vectors,
- σ : the standard deviation of the vectors,
- N : the number of vectors,
- i : i^{th} vector

Seven distinct pipelines were implemented using sklearn, one for each of the machine learning regression techniques presented in Section 3.2. These regressors constitute the final step of each pipeline, where the processing steps that proved most efficient during the optimization investigation are included. The most influential hyperparameters for the respective regressors were fine-tuned during to provide the optimum model. In the Linear Regression model, no hyperparameters required tuning, while for the Lasso and Ridge regressors, the multiplier terms were calibrated for the L1 and L2 regularization respectively. For the ElasticNet model, search was conducted for the terms that control the weight of the L1 and L2 regularization, while the hyperparameter search for the SGD model included the three distinct regularization techniques (L1, L2, elasticnet) along their respective multiplier terms. Moreover, while developing the SVR pipeline, a wide range for the constant C was tested, as well as two different type of kernels: linear and rbf. In the case of the rbf kernel, the search for the optimal value of gamma hyperparameter (γ) was also included in the optimization process. Gamma parameter defines how far the influence of a single training example reaches, with low values meaning far and high values meaning close. Last but not least, the hyperparameters tuned for the Histogram-Gradient Boosting Regressor, included the learning rate, the max leaf nodes of the decision tree, as well as the L2 regularization term. Tables 5.4-5.10 contains the optimal parameters for each case.

The Optuna framework was used for the hyperparameter optimization of the pipelines. In order to ensure that the results of each trial are representative and therefore the final models are robust and not reliant on the chance event of data shuffling, 3-fold cross-validation was used for each experiment. Cross-validation constitutes an evaluation technique, where the model is trained multiple times, each time measuring the performance on a different fraction of the training data that is set aside of the training (Figure 4.5). The final model evaluation results are computed as the mean of the

evaluation results of the distinct cross-validation fits. Subsequently, since we use 3-fold validation, each trial is trained 3 times on 3 different dataset fractions and the Optuna evaluation metric is the mean of these 3 outcomes, thereby minimizing the effect of random data sampling.

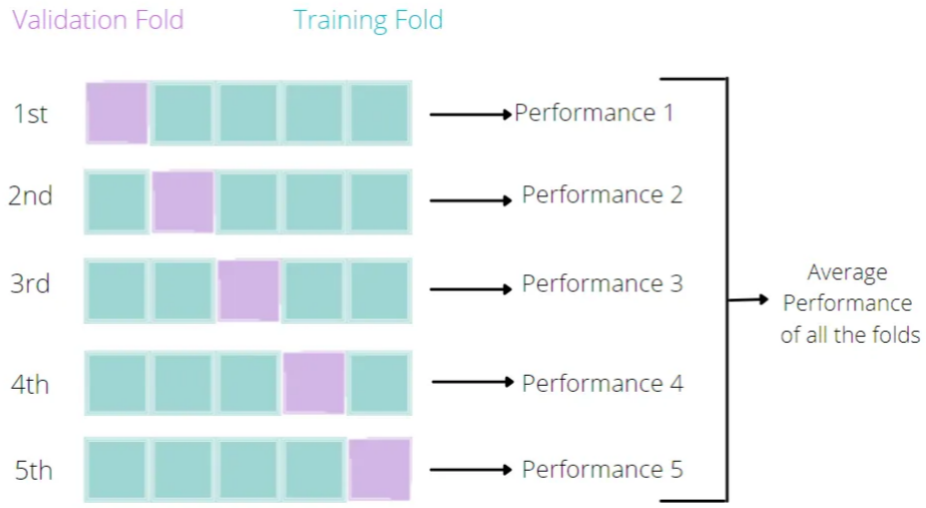


Figure 4.5: *Cross-validation technique*

4.3 Energy Prediction using Deep Learning Models

The core of this thesis lies in the deployment of sophisticated deep learning models aimed at predicting the energy consumption of basic blocks. To this end, we have implemented three distinct approaches, each with its unique attributes. Two of these methods are based on the PalmTree instruction representation, since they encode each basic block as the concatenation of the PalmTree embeddings of its Assembly instructions. The third approach, incorporates the creation of an instruction vocabulary and a custom embedding layer that is trained to output a rich and precise representation of the Assembly instruction for the energy prediction task. Given the sequential of basic block as series of Assembly instructions, LSTMs are utilized in two of the deep learning architectures, as a way of handling these sequences. Furthermore, dropout has been incorporated in all architectures to enhance their robustness and improve their generalization ability.

In Figure 4.6, the general training pipeline of the developed deep learning approaches is presented. The models were implemented and trained using the PyTorch open-source framework. The PyTorch library offers rich and efficient components used for deep learning. Aside from neural network layers, PyTorch also offers greatly implemented training tools such as optimizers, training schedulers and functions used for forward and backward propagation steps. Its highly customizable and production-ready characteristics, make PyTorch a great tool for creating a wide variety of systems, while its ready-to-use python API is highly compatible with many data science python libraries[45].

The dataset of basic blocks is randomly split into a train, a validation and a test set.

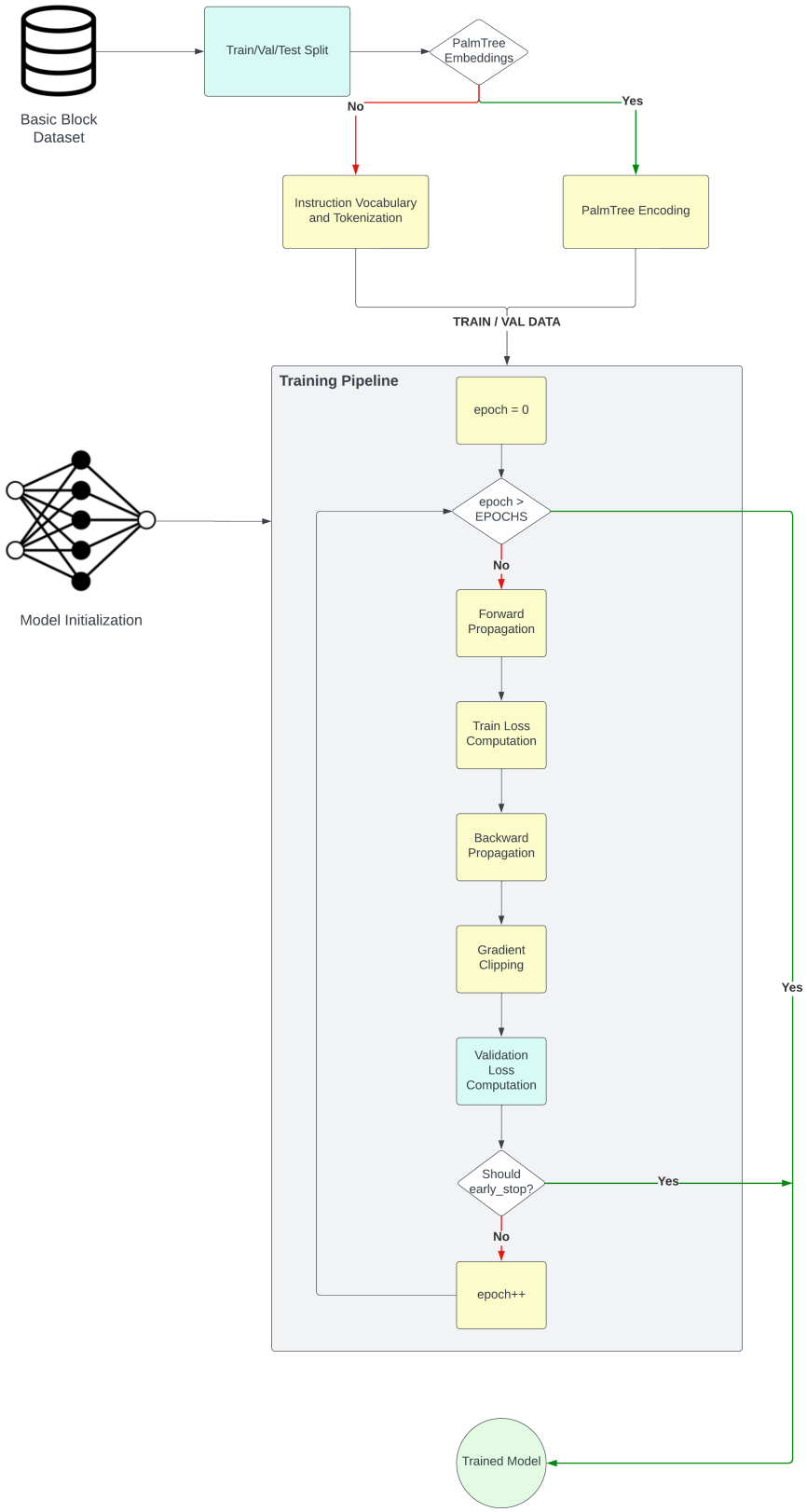


Figure 4.6: *Training pipeline for the Deep Learning Models*

The train set is used for the training of the models, while the validation is used for the models' evaluation after each epoch during training. The hyperparameter optimization is based on the validation set performance. The test set is held aside, ensuring that the performance of the models is measured on data that have not been employed in either training or hyperparameter optimization. This way, we can safely evaluate the generalization of the models to new data. For the train and validation sets PyTorch's DataLoader tool is utilized in order to provide an efficient representation for the models. Moreover, batching is used for the training and validation data, so as to accelerate and smooth the convergence of the model, while simultaneously decreasing the time each iteration requires. The batch size is tuned during the optimization process. It should be also noted, that for the LSTM-based approaches, all sequences for each batch must have the same length for the model to be trained, due to the vectorized computations which require the same matrix sizes. To overcome this challenge, sequence padding is introduced, which involves filling shorter sequences with 0 values to match the length of the longest sequence within each batch. The LSTM hidden states of each batch are initialized to a matrix consisting of zeros.

The training pipeline includes the Adam optimization algorithm, learning rate decay, gradient clipping and early stopping. Early stopping is incorporated in the training process to bring the training to a halt after attaining convergence for the validation set. This way, both the training time is greatly reduced and any excessive training that only lead to overfitting on the training data is avoided. In detail, we measure the validation results at the end of each epoch, and if the performance has not increased for a specific number of epochs the training comes to an end.

4.3.1 Models using PalmTree embeddings

Two distinct deep learning architectures were implemented using the PalmTree representation. The dataset underwent an ingestion process before the training of the models. During this stage, its basic block was mapped to the sequence of the PalmTree embeddings corresponding to its x86 Assembly instructions (Fig. 4.7). The PalmTree embedding dimension for each instruction is 128.

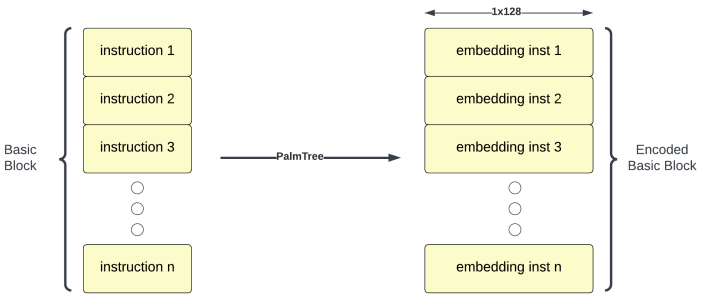


Figure 4.7: *Basic Block encoding using PalmTree*

LSTM model with PalmTree embeddings

The first architecture built with the PalmTree embeddings as input, is a model with LSTM layers connected to fully-connected layers that ultimately yield the energy consumption prediction. The inputs to the LSTM are fed sequentially, one embedding per each basic block instruction, resulting into a many-to-one case. The exact parameters of the models are presented in Section 5.4. The general model architecture, named palmtree lstm model, is presented in Figure 4.8.

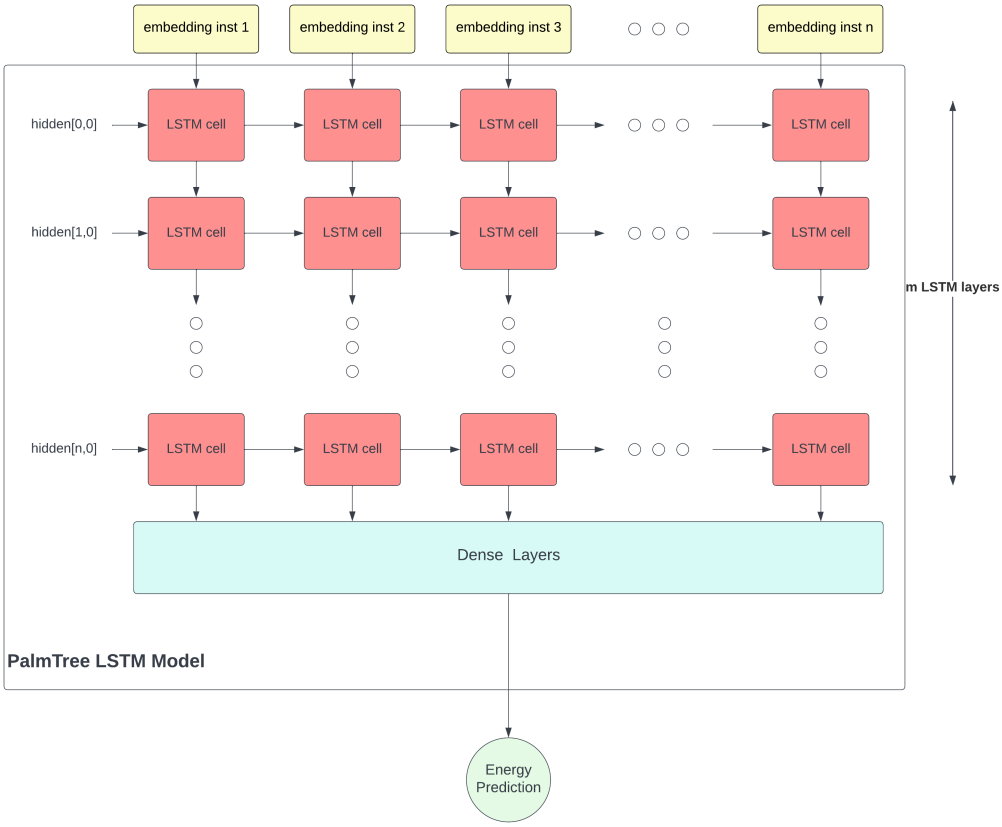


Figure 4.8: *LSTM model with PalmTree embeddings input*

Simple Dense model with PalmTree embeddings

Unlike the first PalmTree-input based architecture, the second model was built with a representative basic block embedding as input. Influenced by the concept of Sentence Transformers[25], where each text passage is represented by the mean embedding of all its words embeddings. Similarly, we consider each basic block as a text passage and a basic block embedding is created as the mean of all of each PalmTree instruction embeddings. As the input has been processed into one single embedding per data instance, there is no longer any sequence sequence to consider, hence there LSTM layer is no longer necessary. Therefore, a dense neural network was implemented as a simpler approach for the energy consumption prediction (Fig. 4.9), which we named simple palmtree model.

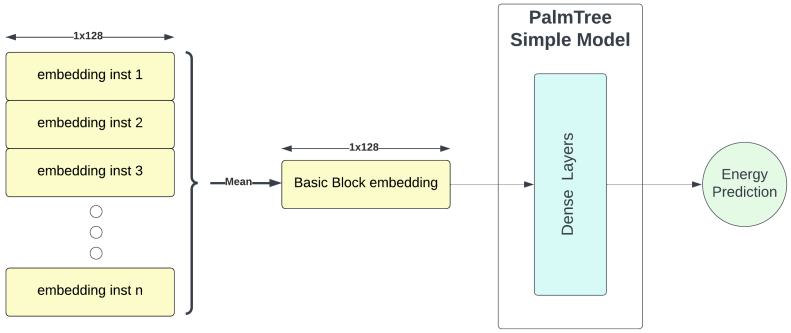


Figure 4.9: *Dense Neural Network model with PalmTree basic block embedding as input*

4.3.2 LSTM with custom embeddings

For the third and final deep neural network approach, an Assembly instruction vocabulary was built, representing each basic block as a sequence of its instructions corresponding tokens (Fig. 4.10), which are fed as input to the model. The model includes an embedding layer followed by LSTM layers, that are finally connected with dense layers to predict the energy consumption (Fig. 4.11). The embedding layer is trained along with the entire model, allowing it to learn instruction embedding representations that optimize energy prediction task. The third approach is named custom lstm model, due to the custom code representation and its detailed parameters are presented along the parameters of the rest of the approaches in Section 5.3.

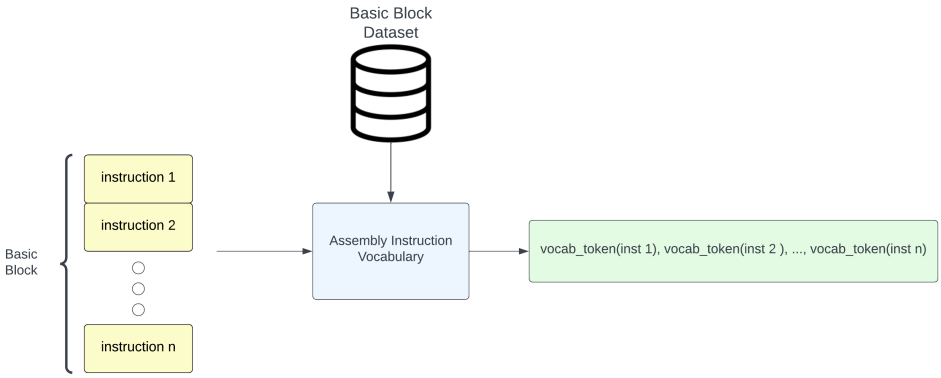


Figure 4.10: *Basic block tokenization based on instruction vocabulary*

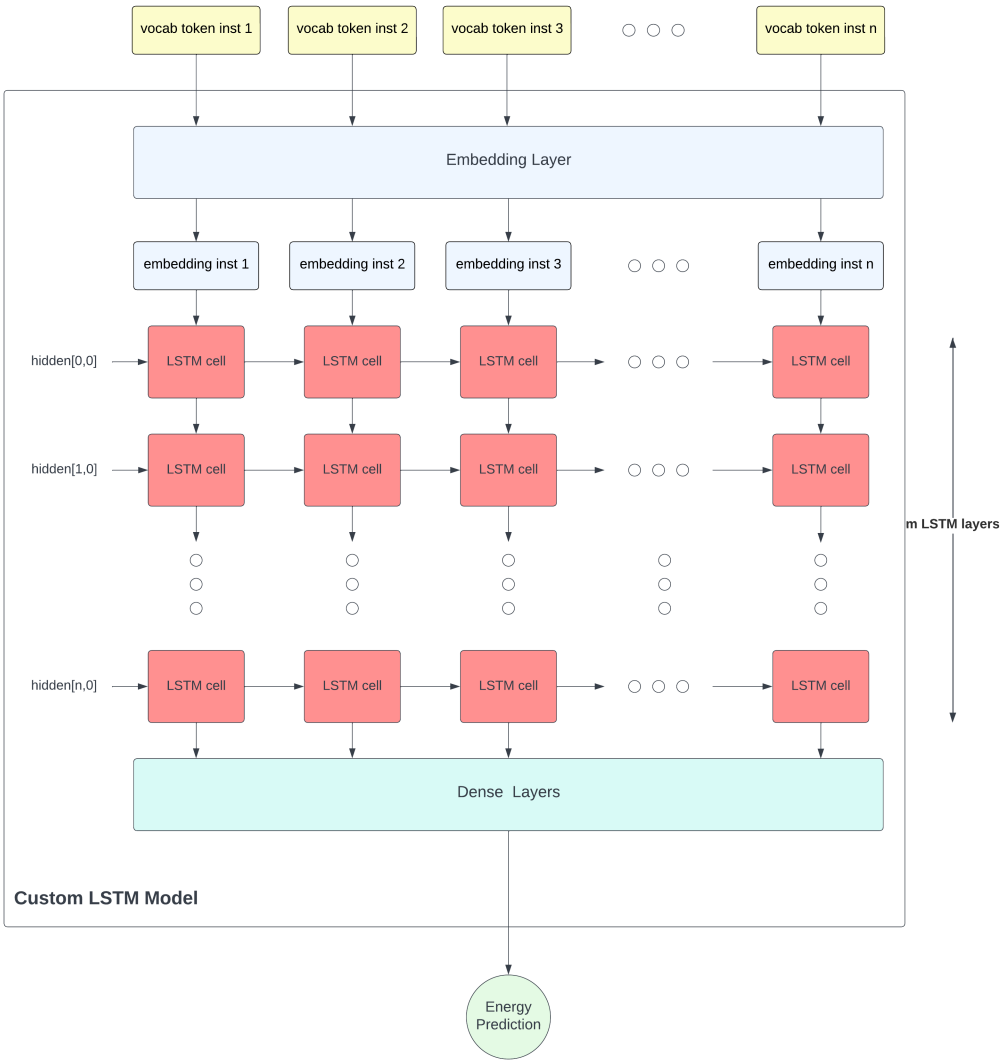


Figure 4.11: *LSTM with custom embeddings*

Chapter 5

Experiments Presentation and Evaluation

The experiments section includes an in-depth demonstration of the evaluation metrics employed for the comparison of the different approaches, an extensive presentation of the evaluation scenarios, the detailed hyperparameters of the developed models, as well as the training parameters and finally the overall performance comparison. To validate the choices made in the experimental design and achieve a comprehensive understanding and interpretation of the final results, we first present an exhaustive analysis of the main characteristics and attributes of the dataset.

5.1 Dataset Analysis Results

The entire basic block energy dataset, comprises precisely 565,679 basic blocks along with their respective energy consumption measurements. These basic blocks originate from 24 distinct executable files that correspond to 24 benchmark programs written in C programming language. An initial statistical analysis regarding the number of Assembly instructions per basic block and the distribution of the energy measurements is presented in Figure 5.1 and Figure 5.2 respectively.

The mean number of instructions per basic block is approximately 5.04, while the mean energy measurement is 0.44 (*61μJ). Moreover, the 99.79% of the data labels are less than 10 (*61μJ), and 98.26% of the basic blocks contain less than 21 Assembly instructions. Hence, outliers that could deteriorate the results were removed, and the dataset was cleaned based on these thresholds. Table 5.1 provides the most crucial statistical metrics of the dataset attributes without the outliers.

The dataset analysis also involved examining the contribution of each benchmark to the overall data (Fig. 5.3). Figure 5.4 illustrates the energy measurement distribution per benchmark. Upon closer examination of the distinct distribution, it becomes

Dataset attribute	Mean	Median	Standard Deviation
Instruction per basic block	5.04	3.0	3.74
Energy Measurements	0.44	0.18	0.7

Table 5.1: *Statistical characteristics of dataset's attributes*

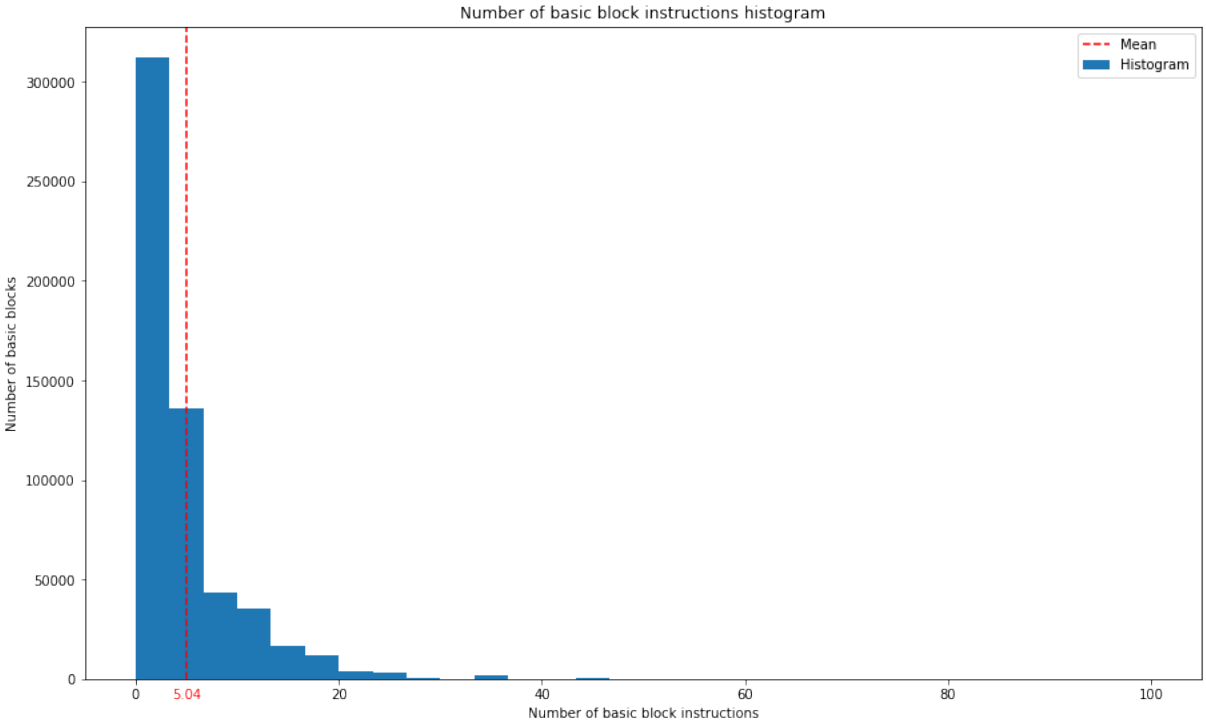


Figure 5.1: *Distribution of the number of instructions per basic block*

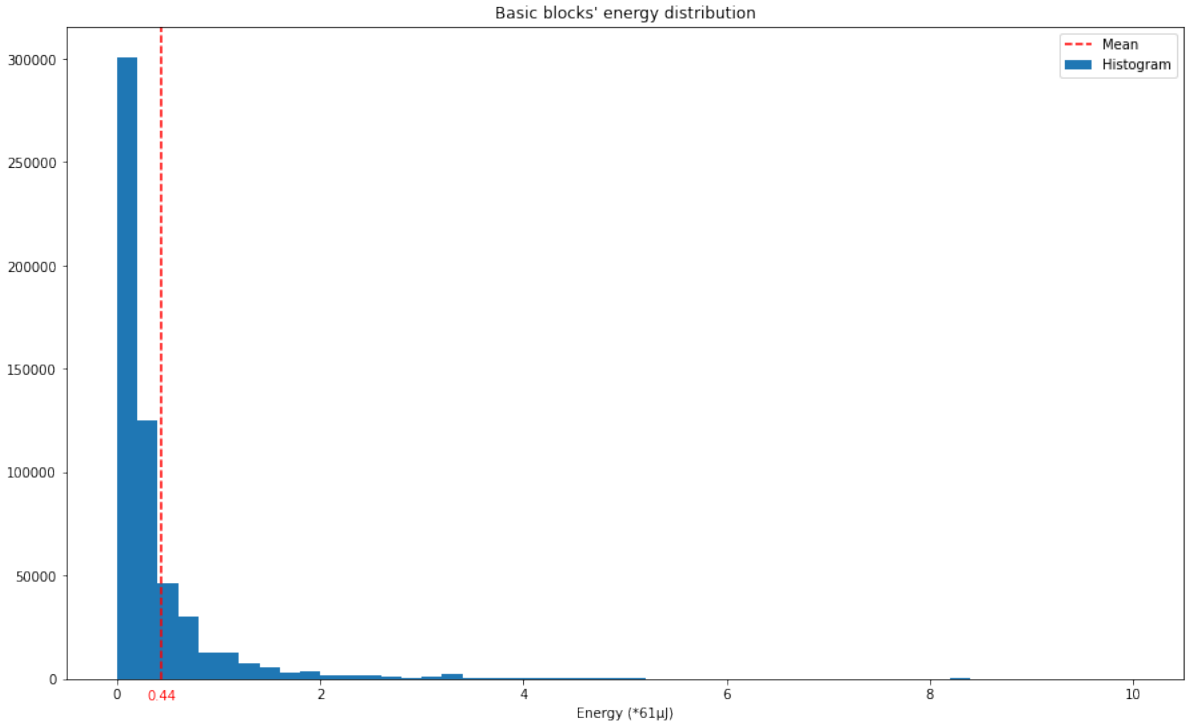


Figure 5.2: *Distribution of the energy measurements*

apparent that the data are not equally distributed with respect to the benchmarks. Also, some benchmarks do not adhere to the general label distribution (e.g. binary_search, count_file, etc.), and they are consisted of relatively small amount of basic blocks, making it harder to achieve the standard performance. Furthermore, the basic block prediction task should be benchmark-agnostic, since the basic blocks are characterized by significantly finer granularity than the programs. Consequently, benchmark-based evaluation may be misleading and is not recommended. However, in order to make this statement more solid, a benchmark-based scenario was included in the evaluation process, as we will demonstrate below.

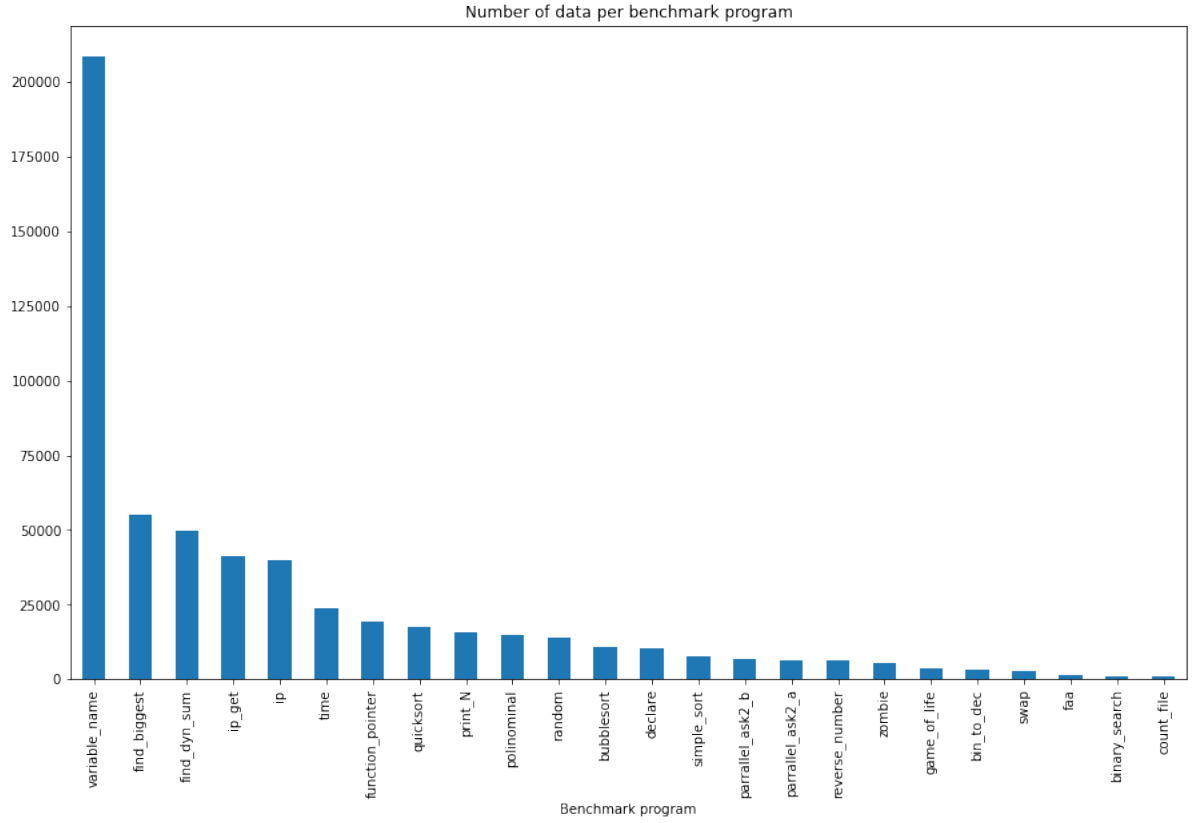


Figure 5.3: *Number of data per benchmark*

Since the dataset includes one measurement per basic block execution, there are multiple measurements per basic block. As a result, there are numerous duplicates of basic blocks within the dataset. Specifically, the dataset is consisted from only 3,828 unique basic blocks, which is due to the fact that a single basic block may be looped inside a program numerous times, and the same basic block may appear in other programs as well. The 64.81% of these unique basic blocks appear multiple times in the dataset, while 1,347 have only one occurrence. The large number of duplicates, raises the concerns regarding the variance of energy consumption measurements for the same basic block. After a careful examination of the basic blocks that appear more than once, we present a histogram of the number of occurrences per basic block (Fig. 5.5), a histogram of the standard deviation of the labels (Fig. 5.6), and the energy measurements of four randomly selected basic blocks with numerous occurrences (Fig. 5.7).

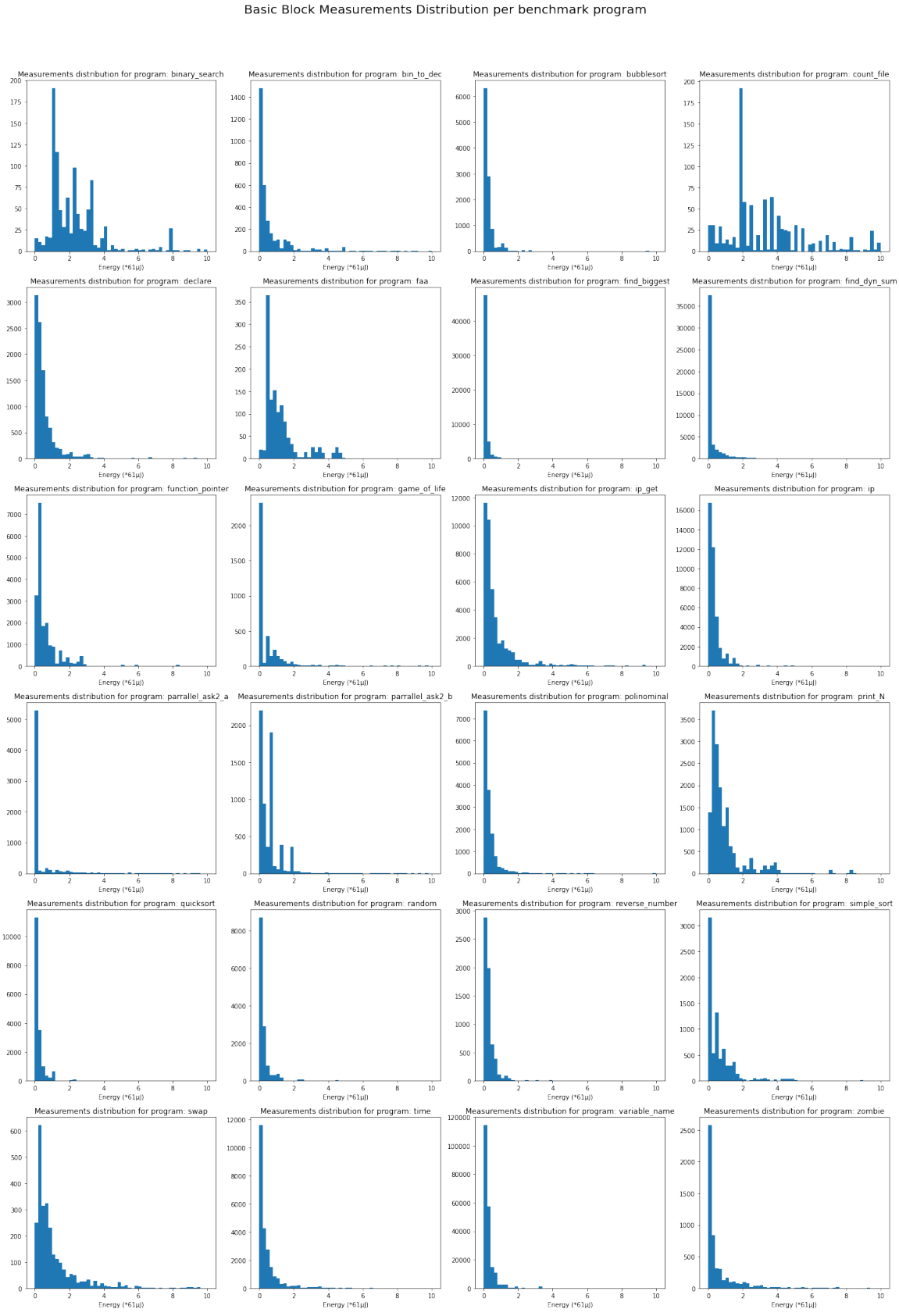


Figure 5.4: *Energy measurements distribution per program*

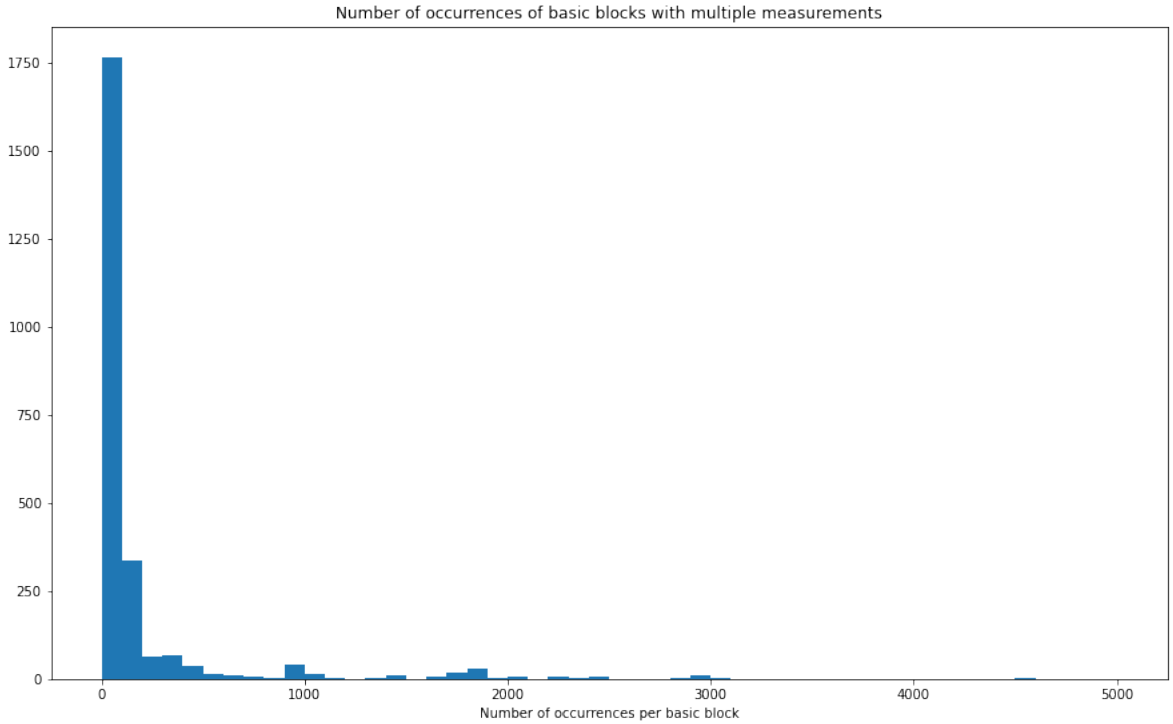


Figure 5.5: *Distribution of the number of occurrences per basic block*

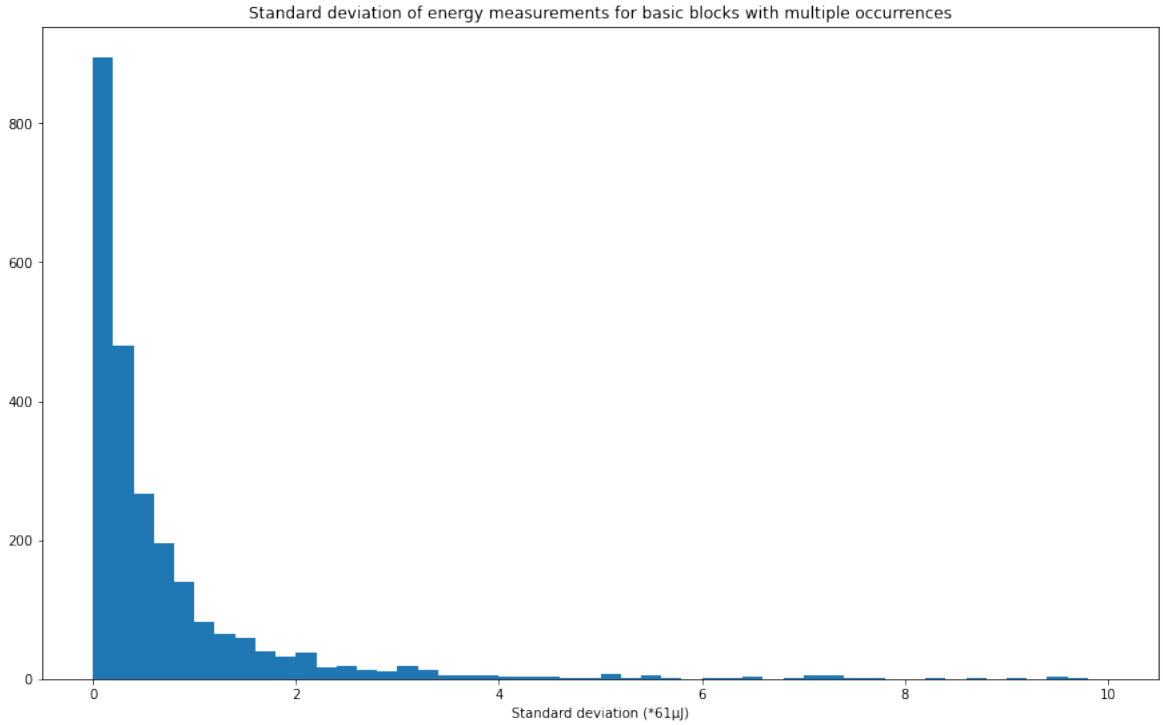


Figure 5.6: *Distribution of the standard deviation of energy labels of each basic block*

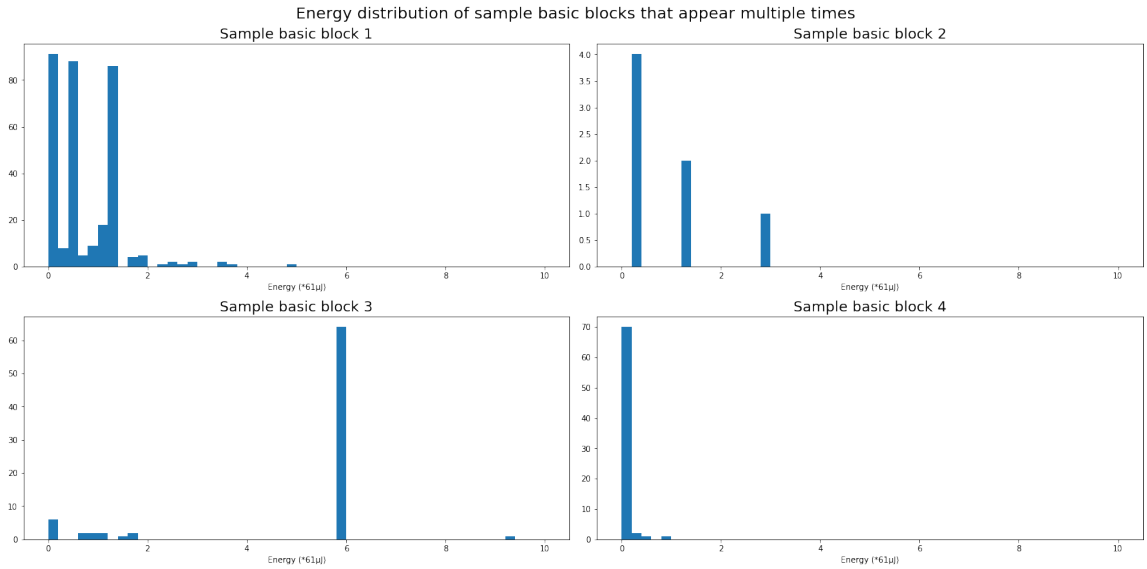


Figure 5.7: *Energy measurements of four randomly sample basic blocks with numerous occurrences*

Dataset attribute	Mean	Median	Standard Deviation
Instruction per basic block	5.96	4.0	4.81
Energy Measurements	0.9	0.39	1.35

Table 5.2: *Statistical characteristics of unique basic block dataset's attributes*

The process of training a machine learning model on data that include the same feature with different labels, constitutes an unorthodox strategy that can prove challenging, since the model has no clear understanding of what to predict. Therefore, an alternative approach has been developed to address the data inconsistencies. This new method involves selecting only one measurement per unique basic block, which is the median of all its labels. The median metric was chosen over the mean, due to its robustness against outliers. Subsequently, the new cleaned dataset contains 3,828 basic blocks, each with a corresponding energy label.

However, since the dataset is small, there is a risk that the proposed data-driven approaches may be heavily impacted. To avoid further reducing this dataset dataset, the technique loosens the maximum instructions per basic block threshold from 20 to 25, resulting in a final dataset of 3,681 basic blocks. Figures 5.8 and 5.9 present the distribution analysis concerning the dataset without the duplicates, while Table 5.2 includes the new statistical metrics. It is worth noting that the distribution of both the instructions per basic block and energy measurements has shifted lightly towards larger numbers.

Finally, we were provided with data measurements from two different runs of four benchmark programs. Although the dataset only includes the results from the latest run for each of the benchmarks, we conducted a comparative analysis between the old and new data measurements to examine any potential variations. This comparison revealed an additional concerning phenomenon regarding the data collection: significant inconsistencies in energy measurements across different executions of the same benchmark

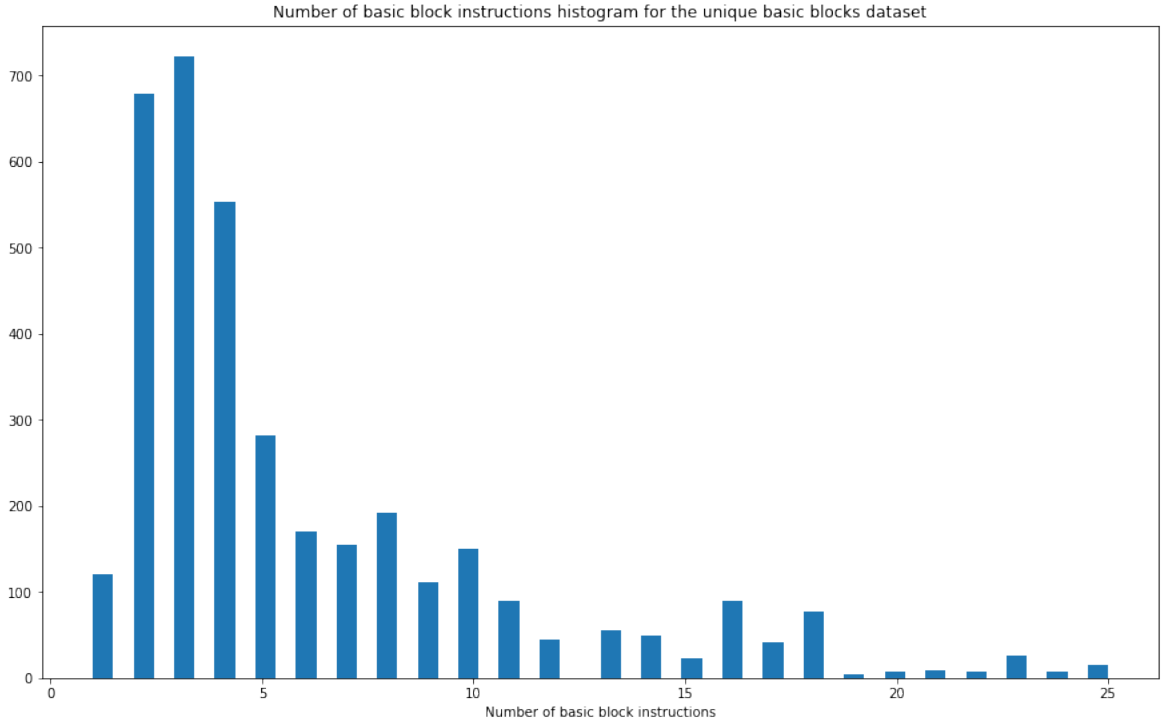


Figure 5.8: *Distribution of the number of instructions per basic block for the unique basic blocks*

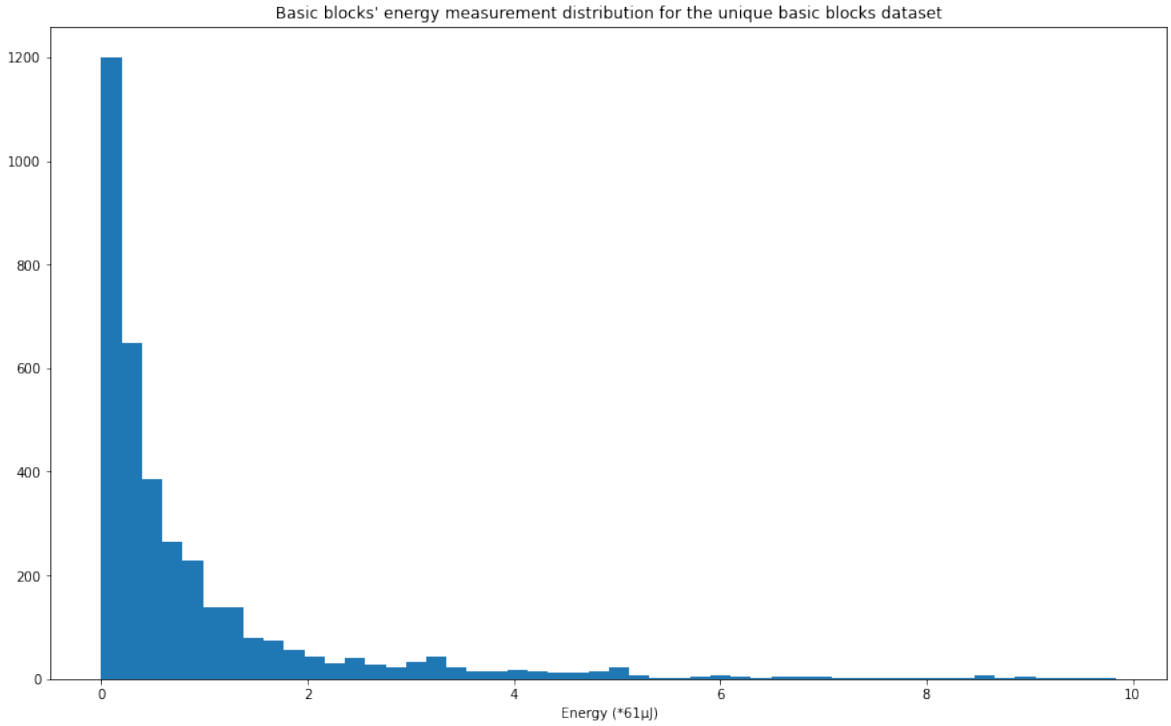


Figure 5.9: *Distribution of the energy measurements of the dataset that contains one label per basic block*

program. Figure 5.10, includes the distribution of the labels for each distinct run per benchmark, alongside the total energies for the two benchmark executions. It is worth noting that the thesis concerning the dataset creation, used as validation method the total program energy. However, a thorough investigation reveals that the total program energy is not a reliable validation technique, as there are instance where the total energies are similar, yet the energy measurements are significantly different (e.g. ”game_of_life” benchmark), while other cases exhibit the opposite behavior (e.g. ”print_N” benchmark). Such inconsistencies may arise due to outliers that may either balance or unbalance the total energy of a program. Additionally, basic block energy measurements offer high resolution, rendering validation based on a significant lower granular level counterproductive. In light of these findings, we propose a comprehensive research effort to investigate the relationship between energy measurement and the state of the system.

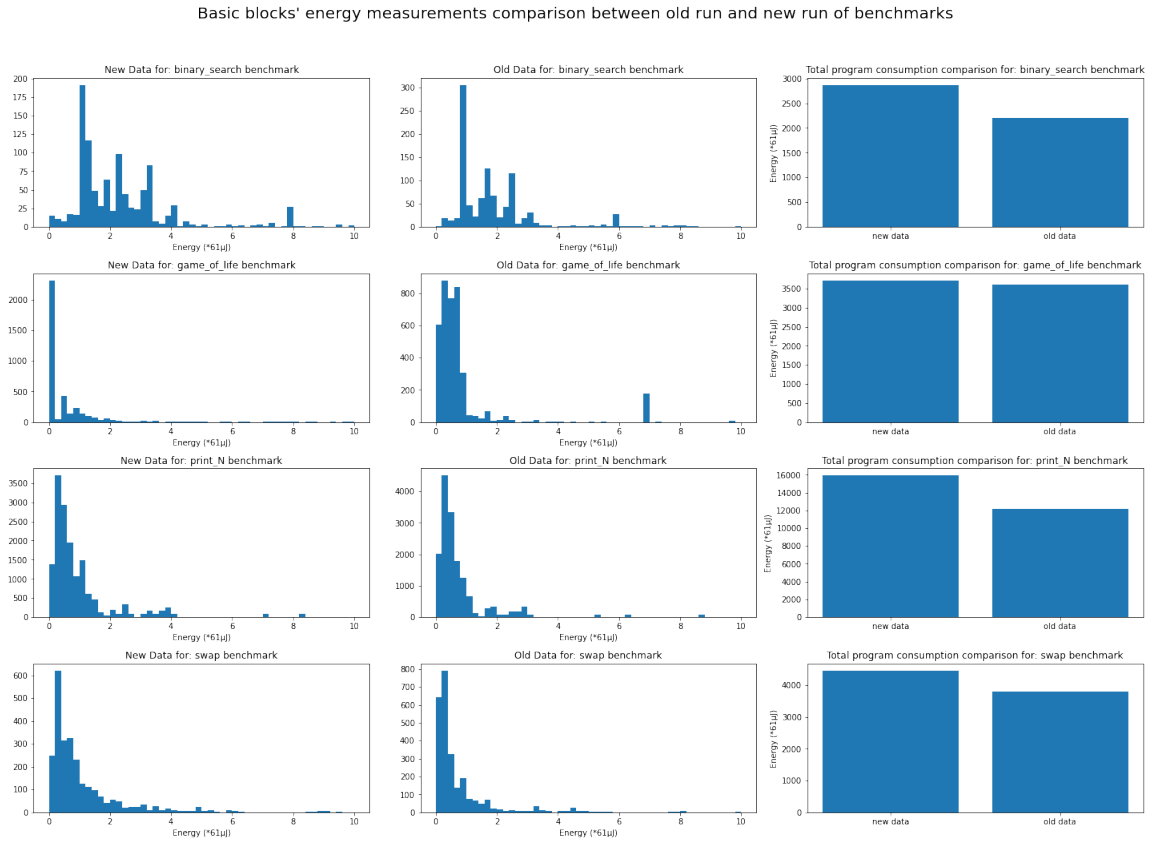


Figure 5.10: *Distribution of the energy measurements of the dataset that contains one label per basic block*

5.2 Evaluation scenarios presentation

Based on the findings of the extensive dataset analysis, the proposed final training and evaluation approaches include two distinct tasks for all of the machine learning architectures: one for the initial dataset and one for the dataset with only unique basic blocks. These two cases will be evaluated on multiple train/test data runs so as to assess

Test Group	Total test data
”find_biggest”, ”binary_search”, ”count_file”	56,422
”variable_name”	208,472
”find_dyn_sum”, ”faa”, ”swap”	53,228
”bin_to_dec”, ”ip_get”, ”game_of_life”, ”random”	61,604
”ip”, ”zombie”, ”reverse_number”, ”parrallel_ask2_a”, ”parrallel_ask2_b”	63,437
”simple_sort”, ”declare”, ”bubblesort”, ”time”	51,991
”function_pointer”, ”quicksort”, ”print_N”, ”polinominal”	66,500

Table 5.3: *Groups of benchmark programs tested together*

the robustness of the models.

An additional evaluation task will be presented for the entire dataset, where each model is trained on the majority of the benchmarks and tested on the remaining ones. The final task is repeated until the performance is tested on all the benchmarks, each time re-initializing the model. For instance, if a specific model is evaluated on ”find_dyn_sum”, ”faa” and ”swap” benchmarks after being trained on the rest. The model is then re-initializes and evaluated on ”bin_to_dec”, ”ip_get”, ”game_of_life” and ”random” benchmarks, after being trained on the rest. The test groups of benchmarks were selected to contain as balanced number of data as possible (Table 5.3). However, This task cannot be applied on the second dataset, since by merging the multiple occurrences for the basic blocks, they lose source benchmark attribute.

To sum up, the experiments consist of:

- **Scenario 1** : Training and evaluating on multiple train/test set variations of the dataset including multiple basic block instances
- **Scenario 2** : Training and evaluating on multiple train/test set variations of the dataset with median energy label for each basic block
- **Scenario 3** : Evaluating on all the benchmarks of dataset that includes multiple basic block instances

5.3 Evaluation Metrics

A wide variety of both sophisticated and simple metrics is employed to assess the performance of machine learning models. For regression tasks, the evaluation is focused on comparing the predictions of the model with the actual labels of the dataset. The most appropriate metrics for guiding the evaluation include the Mean Absolute Error (MAE), the Mean Absolute Percentage Error (MAPE), the Root Mean Square Error (RMSE) and the R-Squared metric (R^2), also known as coefficient of determination.

The MAE is perhaps the most commonly employed evaluation metric for regression tasks, as it directly indicates the error between the predictions and labels. It is calculated following the Eq. 5.1, and measured according to the same units as the data. The MAE should always be considered with respect to the data’s statistics, particularly the standard deviation, as it is highly dependent on the data value ranges and task at hand.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_{pred,i} - y_{true,i}| \tag{5.1}$$

where:

N : number of data
$y_{pred,i}$: model prediction for the i^{th} data instance
$y_{true,i}$: label of the i^{th} data instance

MAPE captures the relative error information between the true and predicted value, based on the scale of the true value, offering an evaluation method that does not require prior statistical analysis. Although, it generally provides a reasonably representative insight, it is not suitable for very small values, as the division with near-zero values disproportionately affects the result. Subsequently, the MAPE metric is not suitable for basic block energy measurement data, for which the median value is 0.18, as small values may dominate the results, leading to an overly pessimistic or inaccurate assessment of the performance of the models.

$$MAPE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}|\frac{y_{pred,i} - y_{true,i}}{y_{true,i}}|} \tag{5.2}$$

The RMSE metric is equivalent to the square root of Mean Squared Error (MSE), and it is generally preferred over MSE, since it includes the advantages of MSE, and is measured in the same units as the target value, unlike the MSE, which is measured in the residual of the units of the target values. RMSE is the metric selected as the loss function for training the machine learning models developed in this thesis, as it penalizes large errors more than MAE does.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_{pred,i} - y_{true,i})^2} = \sqrt{MSE} \tag{5.3}$$

The final widely used metric for the evaluation of regression models is the R^2 score. The R^2 score aims to evaluate how well the model fit the data, by comparing the variance of the predicted data with the variance of the true labels. The maximum value of R^2 is one, which corresponds to the case where the predicted data are precisely the same as the corresponding data labels. The close this metric is to one, the better the model captures the connection between the input of the model with the variance of the output. However, it should be noted that R^2 should be presented along with additional evaluation metrics, such as MAE or RMSE, since a high R^2 does not necessarily means that the model is appropriate for the data, and a low R^2 value does not mean that the model is inadequate.

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{N}(y_{pred,i} - y_{true,i})^2}{\sum_{i=1}^{N}(y_{true,i} - y_{mean})^2} \tag{5.4}$$

where:

Linear Regression			
Dataset Scenario	Pipeline Parameters		
	Normalization	Standard Scaling	TFIDF Transformation
Scenario 1	True	True	True
Scenario 2	False	False	True

Table 5.4: *Optimal parameters for the Linear Regression pipeline*

Lasso Regression				
Dataset Scenario	Pipeline Parameters			Regressor Parameters
	Normalization	Standard Scaling	TFIDF Transformation	α
Scenario 1	False	False	True	5.9
Scenario 2	False	True	False	0.05

Table 5.5: *Optimal parameters for the Lasso Regression pipeline*

RSS : residual sum of squares
TSS : total sum of squares
y_{mean} : mean of data labels

To conclude, the RMSE metric was chosen for training the machine learning models, while the MAE and the R^2 metrics were included in the evaluation of the models to provide comprehensive and accurate evaluation.

5.4 Optimal Hyperparameters

This section outlines the results of the hyperparameter optimization process, which was performed to determine the optimal training and architecture hyperparameters for each model. Specifically, the models were trained to optimize the validation set RMSE loss, as discussed in Section 5.2. The process employed the Optuna framework, with 100 trials for each of the deep learning approaches and 20 to 40 trials for each of the sklearn regressors. The number of trials varied due to the significantly greater range of hyperparameters involved in the deep learning architectures.

For the sklearn-based methods, the training parameters focused on the parameters of the pipeline, while for the deep learning approaches, they concerned the training loop parameters. The best hyperparameters for each of the implemented models for Scenarios 1 and 2, resulting in 20 distinct models, 10 for each dataset, are presented in Tables 5.4-5.13.

It is important to note that Linear Regression does not require hyperparameter tuning for the regressor component of the pipeline. Additionally, the SVR pipeline of the entire dataset was optimized on 50,000 randomly sampled data for practical reasons, as SVMs do not scale well with large amounts of data. The SVR pipeline halted in many cases when trying to train on the full dataset of 550,000 data points.

To reduce the overwhelming number of hyperparameters for all the models, key points of the deep neural network architectures were highlighted. It was observed that the models trained on the smaller dataset with median energy measurements as labels

Ridge Regression				
Dataset Scenario	Pipeline Parameters			Regressor Parameters
	Normalization	Standard Scaling	TFIDF Transformation	α
Scenario 1	True	True	True	7.6
Scenario 2	False	False	False	8.4

Table 5.6: *Optimal parameters for the Ridge Regression pipeline*

ElasticNet Regression					
Dataset Scenario	Pipeline Parameters			Regressor Parameters	
	Normalization	Standard Scaling	TFIDF Transformation	λ_1	λ_2
Scenario 1	False	True	True	0.006	6.95
Scenario 2	False	True	True	0.31	0.27

Table 5.7: *Optimal parameters for the ElasticNet Regression pipeline*

SGD						
Dataset Scenario	Pipeline Parameters			Regressor Parameters		
	Normalization	Standard Scaling	TFIDF Transformation	λ_1	λ_2	penalty
Scenario 1	False	False	False	0.32	0.63	”l2”
Scenario 2	False	True	False	0.89	3.69	”l2”

Table 5.8: *Optimal parameters for the SGD pipeline*

SVR						
Dataset Scenario	Pipeline Parameters			Regressor Parameters		
	Normalization	Standard Scaling	TFIDF Transformation	kernel	C	γ
Scenario 1	False	False	True	”rbf”	8.3	0.97
Scenario 2	True	False	True	”rbf”	6.9	0.25

Table 5.9: *Optimal parameters for the SVR pipeline*

Hist-Gradient Boosting						
Dataset Scenario	Pipeline Parameters			Regressor Parameters		
	Normalization	Standard Scaling	TFIDF Transformation	learning rate	max leaves	l2 regularization
Scenario 1	False	True	True	0.24	20	8.8
Scenario 2	False	False	False	0.035	11	2.75

Table 5.10: *Optimal parameters for the Hist-Gradient Boosting pipeline*

PalmTree LSTM Model							
Dataset Scenario	Training Parameters		Model Parameters				
	Learning Rate	Batch Size	LSTM hidden size	LSTM layers	Dense Layer Sizes	LSTM dropout	Dense dropout
Scenario 1	0.01	149	84	2	[175, 37]	0.09	0.15
Scenario 2	0.04	14	57	1	[255, 24]	None	0.08

Table 5.11: *Optimal parameters for the PalmTree LSTM model architecture*

PalmTree Simple Model					
Dataset Scenario	Training Parameters		Model Parameters		
	Learning Rate	Batch Size	Dense Layers	Dense dropout	Last layer dropout
Scenario 1	0.01	404	[251]	0.08	0.05
Scenario 2	0.0036	9	[147, 100]	0.13	0.07

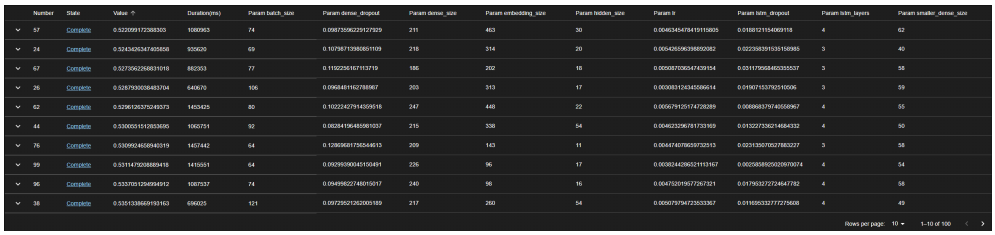
Table 5.12: *Optimal parameters for the PalmTree Simple model architecture*

Custom LSTM Model								
Dataset Scenario	Training Parameters		Model Parameters					
	Learning Rate	Batch Size	Embedding Size	LSTM hidden size	LSTM layers	Dense Layers Sizes	LSTM dropout	Dense dropout
Scenario 1	0.0046	74	463	30	4	[211, 62]	0.02	0.1
Scenario 2	0.006	13	245	35	2	[178, 121]	0.19	0.003

Table 5.13: *Optimal parameters for the Custom LSTM model architecture*

had considerably fewer hyperparameters than the ones trained for the entire dataset. This behavior was expected, since a model with more hyperparameters would increase overfitting. Additionally, the batch size adapted to the size of the dataset, while the custom embeddings created for the smaller dataset had nearly half the dimensions of the embeddings created for the entire dataset.

The Optuna framework provided useful insights for the hyperparameters of each architecture, through its dashboard. We focus on the case of the Custom LSTM Model for the entire dataset. Figure 5.11 present the best performing trials based on the RMSE of the validation set, with the top trial corresponding to the hyperparameters contained in Table 5.13 for Scenario 1. The most dominant hyperparameter for the Custom LSTM model for the entire dataset's performance was the batch size (Fig. 5.12). The Optuna trials showed that the values corresponding to better scores were included in many trials (Fig. 5.14), indicating that the optimization process was guided towards the most promising direction. Optuna also provides contour plots for hyperparameter pairs, indicating the best performing points with darker colour (Fig. 5.13).



Number	State	Value	Duration(ms)	Param batch_size	Param dense_dropout	Param dense_size	Param embedding_size	Param hidden_size	Param lr	Param lstm_dropout	Param lstm_layers	Param smaller_dense_size
57	Complete	0.522059172388303	1060963	74	0.0987359623912929	211	463	30	0.0046345478419115803	0.018812115140609118	4	62
24	Complete	0.5243426347405858	920620	69	0.10798713980851109	216	314	20	0.005426596398852082	0.022358391535159365	3	40
67	Complete	0.5273962268831018	882593	77	0.11922565167113719	186	202	18	0.005087036047429154	0.021175568460355537	3	58
26	Complete	0.5267930058483704	640670	106	0.09684811627881667	203	313	17	0.0030831243435896614	0.019071537925165096	3	59
62	Complete	0.5296126337024937	1453425	80	0.10222427914359516	247	448	22	0.005679125174728289	0.008868379740568967	4	55
44	Complete	0.5300551512853695	1060751	92	0.08284196488561037	215	338	54	0.004623296781733169	0.013227336214684332	4	50
76	Complete	0.5309924658940319	1457442	64	0.12869681756544615	209	143	11	0.004474078659732513	0.023136070527883227	3	58
99	Complete	0.5311475208889418	1415051	64	0.09299390045150491	226	96	17	0.003824428632111167	0.0025858925020970074	4	54
96	Complete	0.5337051254994912	1087337	74	0.0949982274815017	240	98	16	0.004752019577267321	0.017953272724647782	4	58
38	Complete	0.5351338669193163	696025	121	0.09735921262003189	217	260	54	0.0050797947233533367	0.011695332777275608	4	49

Rows per page: 10 1–10 of 100

Figure 5.11: *Top 10 Optuna trials for Custom LSTM Model for the entire dataset based on validation set RMSE*

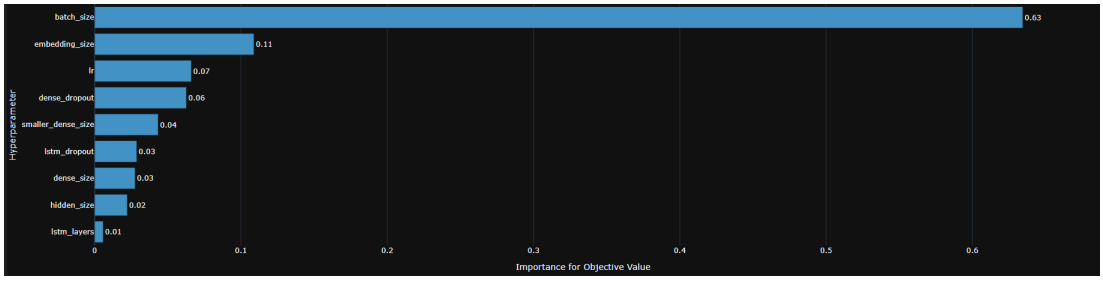


Figure 5.12: *Hyperparameter importance for the parameters Custom LSTM Model for Scenario 1*

Finally, the training and validation loss plot across the epochs for the final Custom LSTM Model for Scenario 1 (Fig. 5.15) was presented. The validation loss converged to 0.55, and training was terminated with early stopping to prevent further overfitting. It is worth noting that the training and hyperparameter optimization experiments for the deep learning architectures were conducted using the CUDA cores of an RTX 3060 GPU, which greatly accelerated the process.

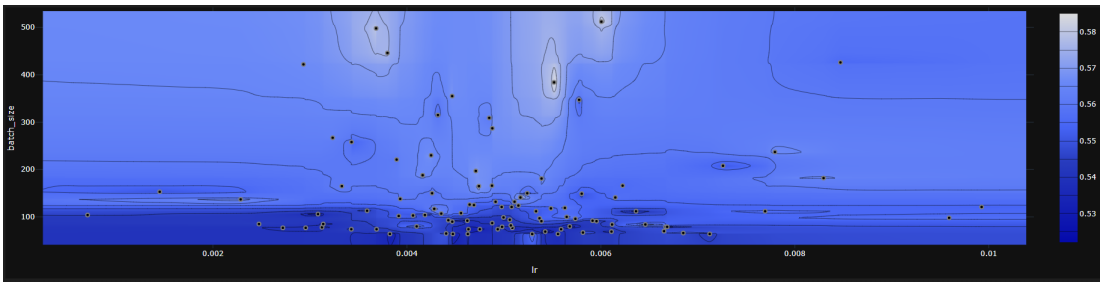


Figure 5.13: *Contour plot for the batch size in respect to the learning rate for the Custom LSTM Model for Scenario 1*

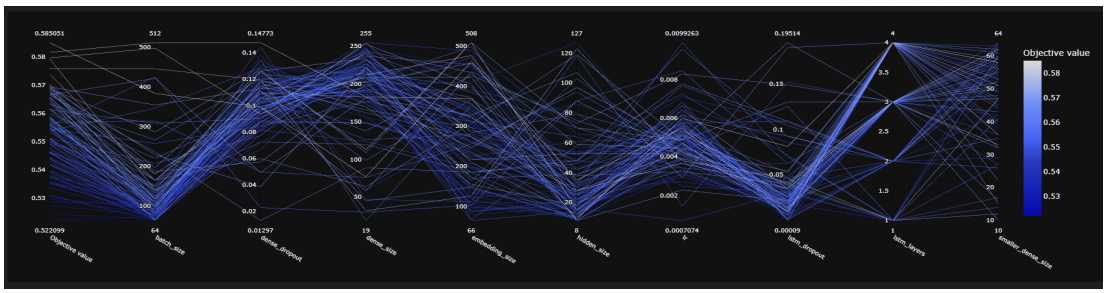


Figure 5.14: *Hyperparameter combinations for Optuna Trials of the Custom LSTM Model for Scenario 1*

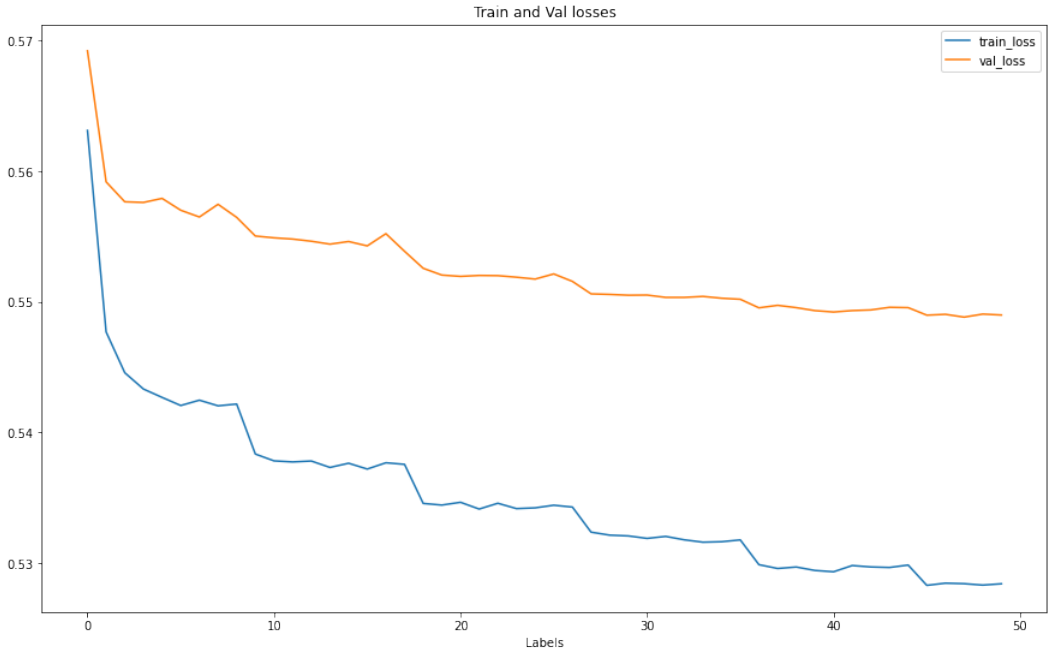


Figure 5.15: *Training and Validation Loss for the final Custom LSTM Model for Scenario 1*

5.5 Results Presentation

Since the specific model architectures have been analyzed, the detail results for the test sets of each scenario will be presented, in order to enable a conclusive interpretation the performance of each approach. The performance is evaluated based on the MAE, RMSE and R^2 metrics.

5.5.1 Scenario 1

In order to comprehensively evaluate the efficacy of the models developed for the entire dataset, a train/test process was performed five times on different samples of the dataset. Each iteration involved training and validating the models on 90% of the dataset and testing on the remaining 10%. Figure 5.16 illustrates the Mean Absolute Error (MAE) of the models implemented for the distinct test cases, while Figures 5.17 and 5.18 detail the Root Mean Squared Error (RMSE) and coefficient of determination (R^2) results respectively.

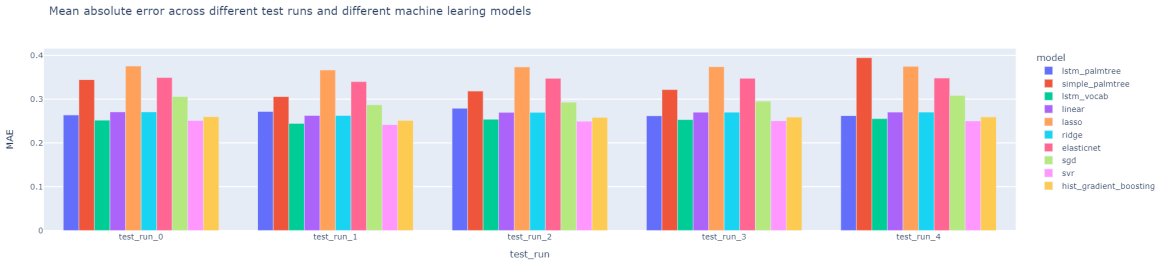


Figure 5.16: *MAE for the implemented models for the entire dataset across different train/test runs*

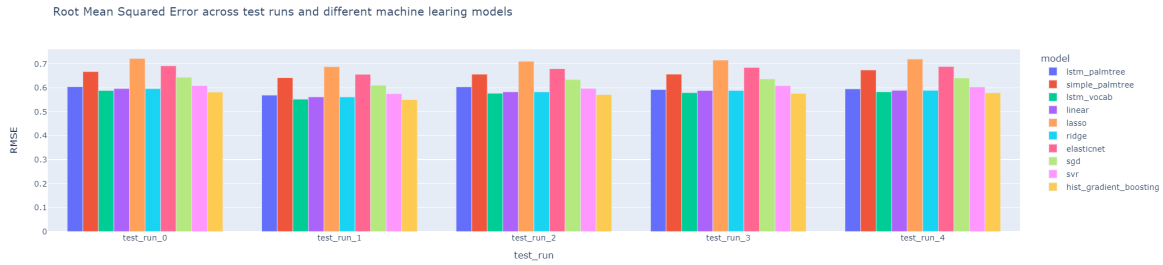


Figure 5.17: *RMSE for the implemented models for the entire dataset across different train/test runs*

The results obtained from the various train/test runs exhibit incremental differences, with the exception of the simple PalmTree model, which performed poorly for test set 5. Therefore, we can safely assume that the models are robust and their performance is independent of the training and testing data sampling. The Custom LSTM model, the SVR, and the Histogram-Gradient Boosting models proved the top-performing models, achieving an MAE of approximately 0.25, while Linear and Ridge Regression perform great as well. This is noteworthy given that the standard deviation of the energy mea-

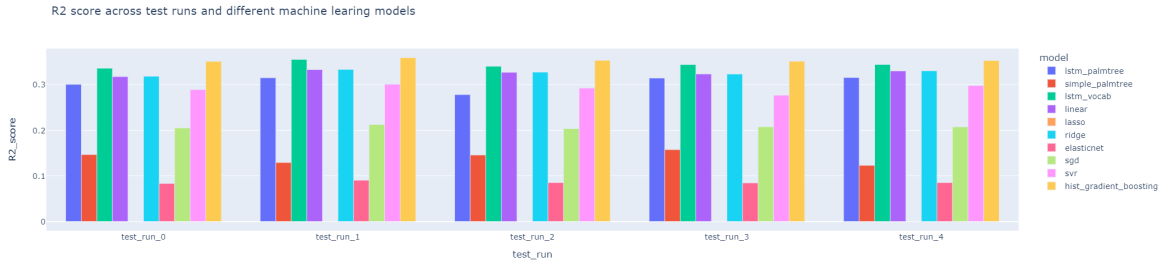


Figure 5.18: *R^2 for the implemented models for the entire dataset across different train/test runs*

surements for the entire dataset is around 0.7. When assessing the models based on RMSE, the Custom LSTM model,the Histogram-Gradient model and the Linear and Ridge Regression pipelines achieved the best results, with an RMSE of approximately 0.55. For R^2 scores, the Histogram-Gradient Boost model outperformed other models, closely followed by the Custom LSTM model. It is worth noting that the best R^2 scores were around 0.35, suggesting that the models moderately explain the relationship between input features and output.

Based on the assessment of all the evaluation metrics, we conclude that the Custom LSTM model is the most effective, followed by the Histogram-Gradient Boosting, SVR, Linear and Ridge Regression pipelines. The PalmTree LSTM model provided relatively good results, while the Simple PalmTree Model, Lasso Regression, ElasticNet, and SGD approaches performed poorly compared to the rest of the models.

Figure 5.19 presents the distributions of the true energy labels and the Custom LSTM Model energy predictions for the third train/test sets run, providing a detailed illustration of the model's performance. Notably, the model struggles to predict values that are very close to 0, while being able to capture some values that appear rarely (values around $3 * 61 \mu J$).

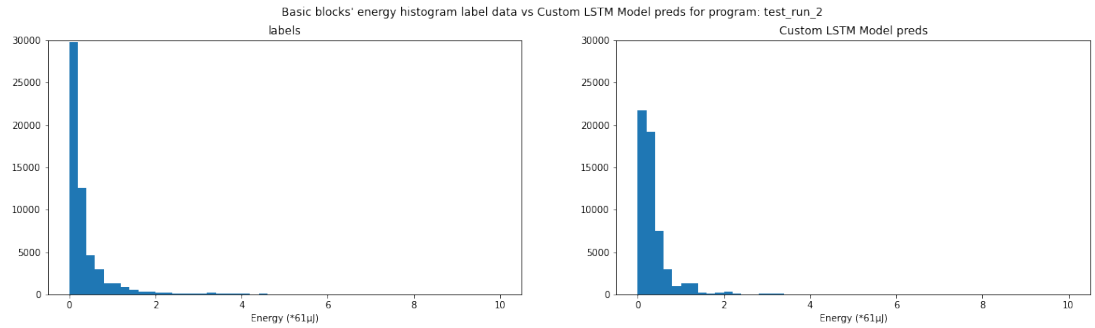


Figure 5.19: *Custom LSTM Model predictions distribution compared with labels distribution*

5.5.2 Scenario 2

The evaluation process for scenario 2 closely mirrors the methodology that was employed for scenario 1, with the main difference being that the test sets in scenario 2

contain 15% of the data, with the remaining 85% being divided into train and validation sets. This diversion from the previous scenario was motivated by the limited amount of data available for scenario 2, requiring a higher percentage of data for the evaluation of model performance. Similarly to scenario 1, we present the MAE, RMSE, and R^2 results for each test case in Figures 5.20, 5.21, and 5.22, respectively.

Figure 5.20: *MAE for the implemented models for the unique basic block dataset across different train/test runs*

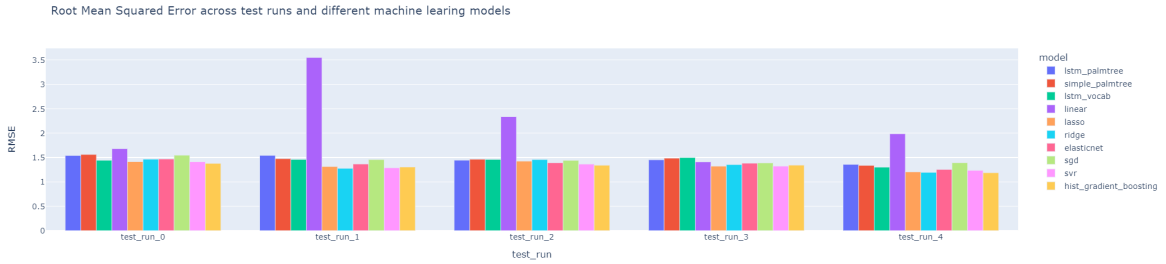


Figure 5.21: *RMSE for the implemented models for the unique basic block dataset across different train/test runs*

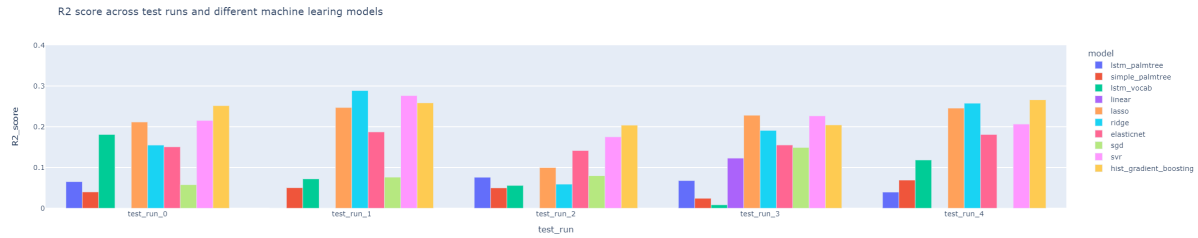


Figure 5.22: *R^2 for the implemented models for the unique basic block dataset across different train/test runs*

The results for the evaluation of the dataset that contains only the unique basic blocks with their median energy as labels, revealed considerable variability across different samples of train and test sets. This variability was anticipated, given the limited amount of data available for training and evaluating data-driven approaches. While assigning median measurements as representative of each basic block was proposed as a solution to cope with the multiple labels, such an approach may not accurately reflect the energy consumption of the basic blocks.

Examining the combined metric results, the SVR pipeline outperformed the other

machine learning approaches, achieving an MAE of approximately 0.75, with a standard deviation for the new dataset fluctuating around 1.5. Figure 5.23 depicts the distribution of labels for the small dataset, along with the corresponding distribution of predictions generated by the SVR pipeline. The SVR approach exhibits moderate performance, with labels appearing more sparse when compared to those in scenario 1.

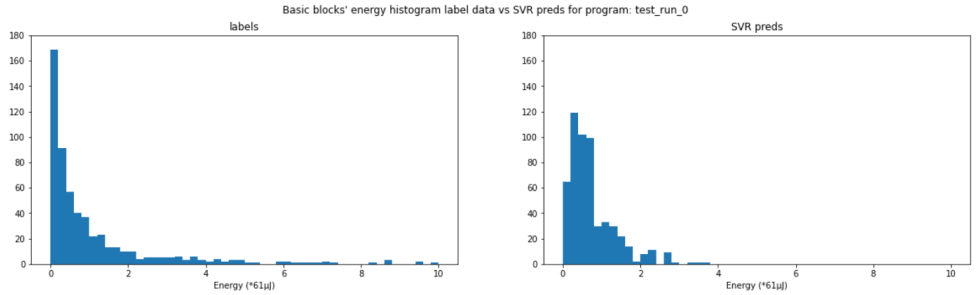


Figure 5.23: *SVR Pipeline predictions distribution compared with labels distribution for the dataset of the unique basic blocks*

5.5.3 Scenario 3

The performance of the machine learning models developed for the entire dataset was further evaluated across various benchmark programs, as outlined in section 5.2. The evaluation process yielded performance metric results for each benchmark, as illustrated in Figures 5.24-5.26. It should be noted, however, that the results exhibit a considerable degree of variance, owing to the unbalanced number of data points and label distribution across the benchmarks. Hence, it is difficult to draw a definitive conclusion based on the obtained results.

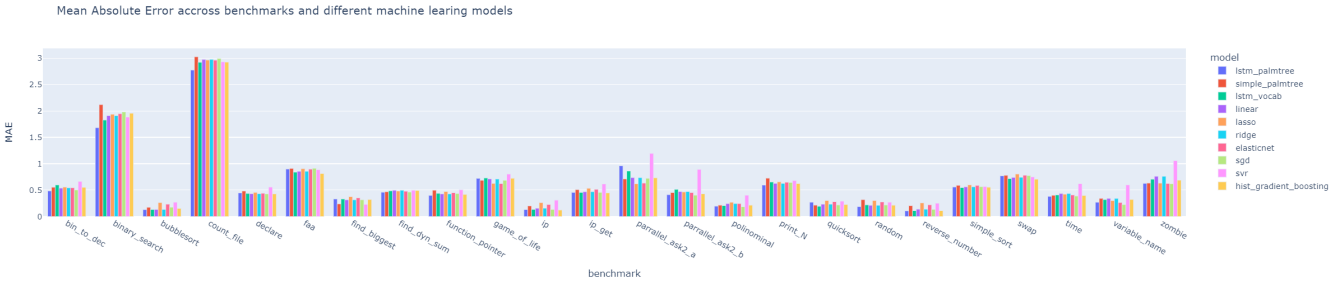


Figure 5.24: *MAE for the implemented models for the entire dataset across the benchmark programs*

5.5.4 Final Results

To provide a concise summary of the machine learning evaluation, the first scenario is deemed the most appropriate approach for identifying the best model for the task of basic block energy consumption prediction. Despite the dataset's design flaws, where multiple labels are assigned to the same input, it is still considered superior to training

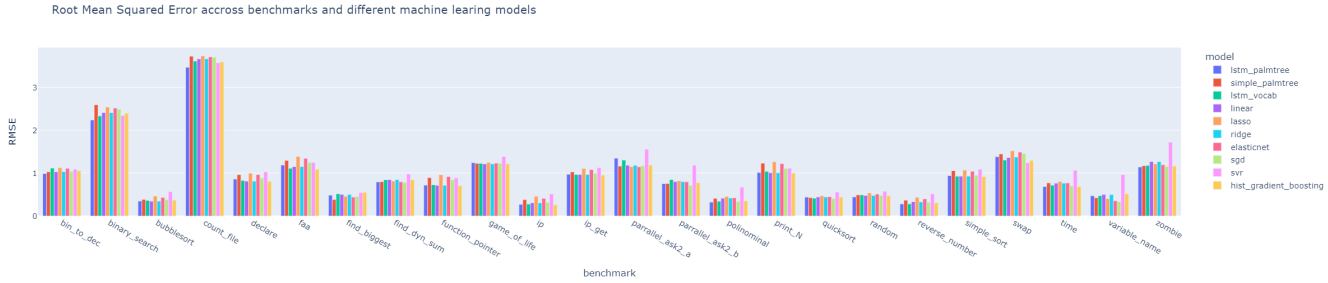


Figure 5.25: *RMSE for the implemented models for the entire dataset across the benchmark programs*

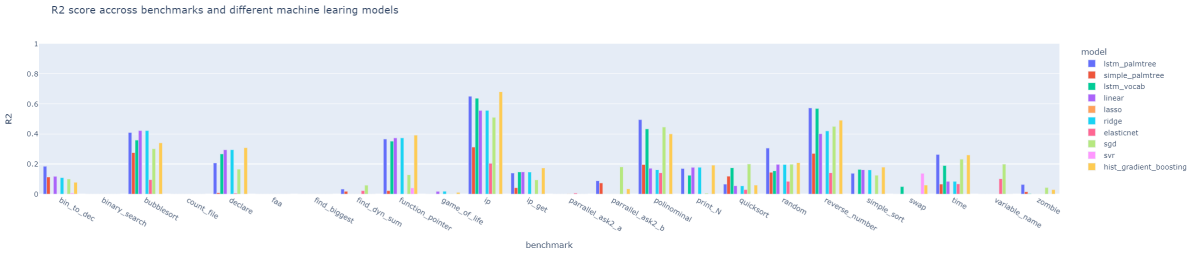


Figure 5.26: *R^2 for the implemented models for the entire dataset across the benchmark programs*

and evaluating data-driven approaches with only 3,828 data points. Additionally, evaluating based on scenario 3 is not ideal for basic block energy prediction since it focuses on specific program cases rather than the general case.

As the results of the first scenario were consistent across various train-test splits, the final results are presented based on one such split. Figures 5.27-5.29 provide a high-resolution comparison of the machine learning approaches for the different evaluation metrics, while Table 5.14 details the results across the models. Furthermore, the distribution of the best-performing approaches' predictions is presented alongside the label distribution in Figure 5.30.

Upon analyzing the final results, it can be inferred that the machine learning models perform relatively well in predicting the basic block energy consumption, given the small standard deviation of 0.7 in the labels. However, it should be noted that the varying energy measurements for the same basic block have a significant impact on the performance of the models and we strongly believe that considering the preceding basic block sequence as context would significantly improve the results of the LSTM models. Among the different model approaches, the Custom LSTM Model is expected to outperform the PalmTree-based input models due to its ability to tailor the embeddings for the specific energy consumption task. Furthermore, the SVR approach proves to be a well-performing alternative, especially in cases where the input is characterized by many dimensions, as SVMs are adept at handling such scenarios. Linear and Ridge Regression pipelines perform at a great level as well. In addition, Gradient Boosting also shows promising results, as it is able to capture the relation between the input features and the output values' variation. In conclusion, these models should be considered as potential candidates for basic block energy consumption prediction.

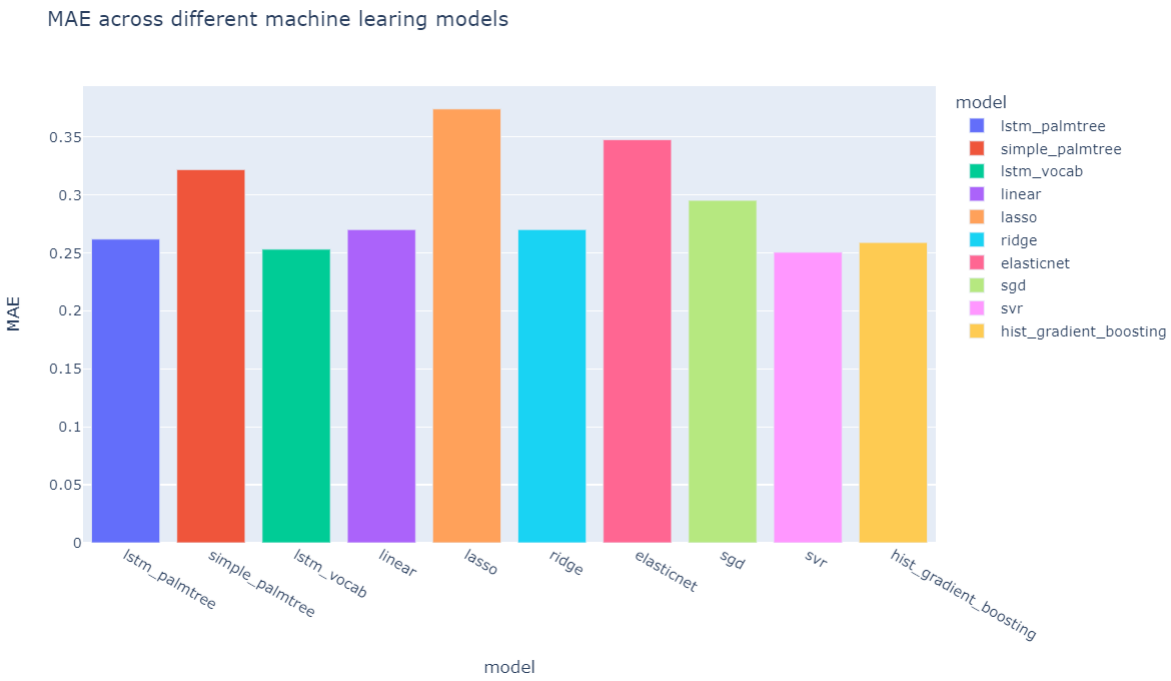


Figure 5.27: *MAE for the implemented models for the entire dataset*

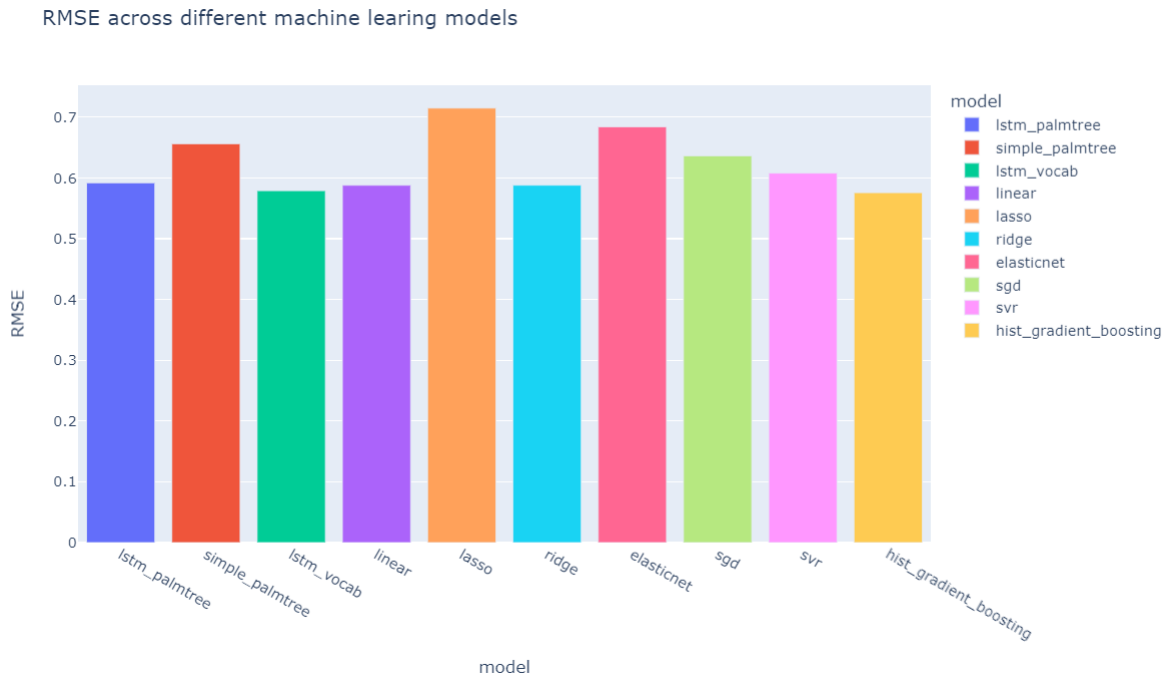


Figure 5.28: *RMSE for the implemented models for the entire dataset*

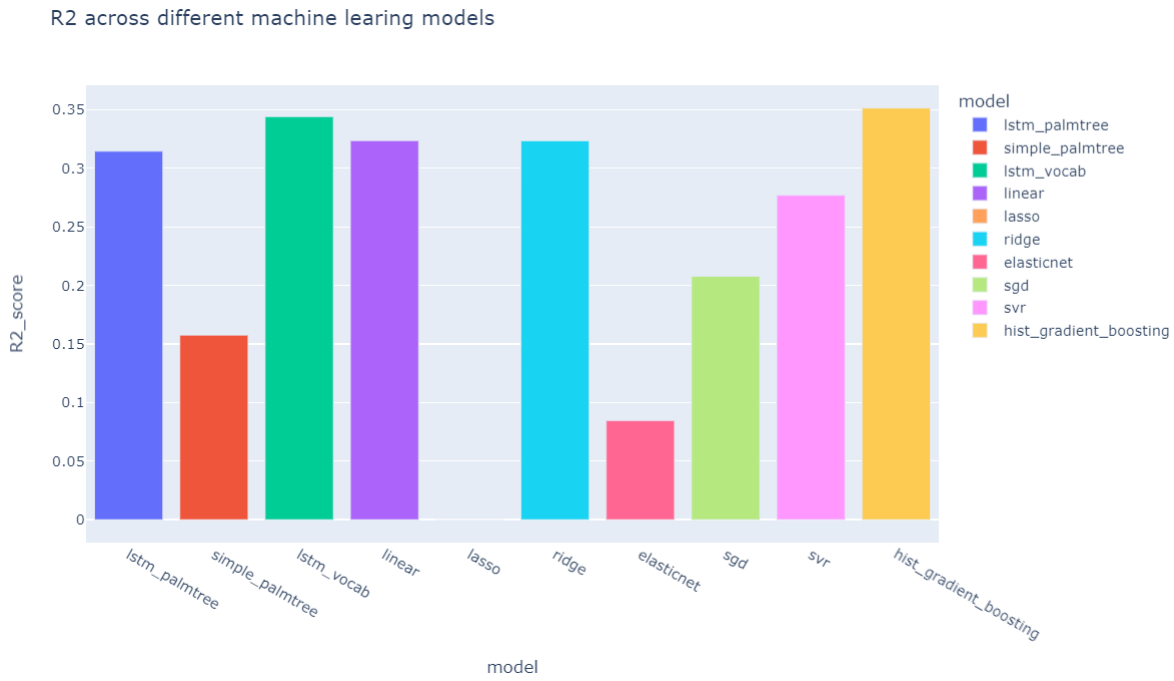


Figure 5.29: *R^2 for the implemented models for the entire dataset*

Final Evaluation Results			
Machine Learning Approach	MAE	RMSE	R2
PalmTree LSTM	0.26	0.59	0.31
Simple LSTM	0.32	0.65	0.15
Custom LSTM	**0.25**	**0.57**	0.34
Linear Regression	0.27	0.58	0.32
Lasso Regression	0.37	0.71	0.0
Ridge Regression	0.27	0.58	0.32
ElasticNet Regression	0.34	0.68	0.08
SGD	0.29	0.63	0.2
SVR	**0.25**	0.6	0.27
Hist-Gradient Boosting	0.26	**0.57**	**0.35**

Table 5.14: *Final results for the evaluation metrics for the models implemented for the entire dataset*

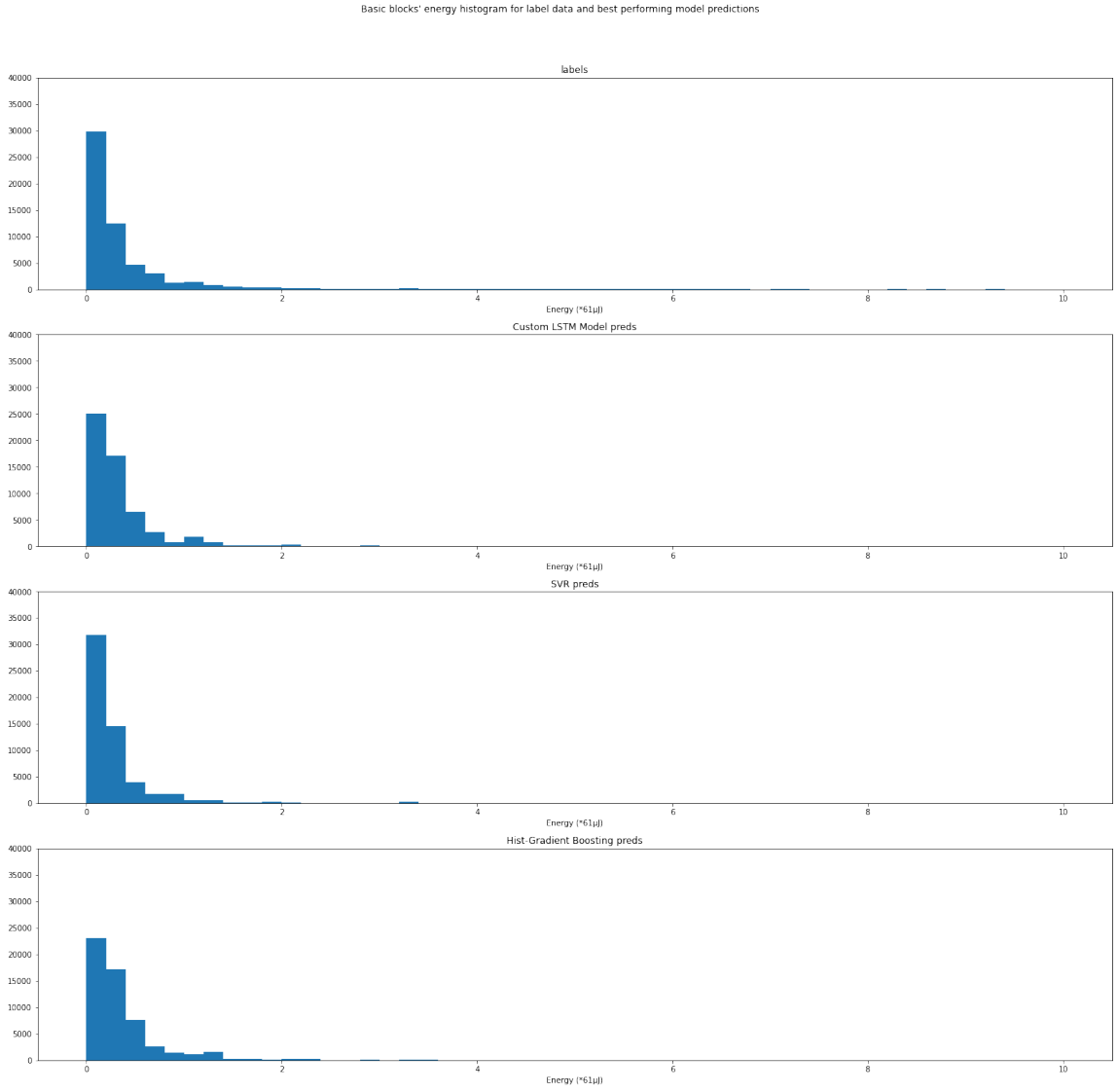


Figure 5.30: *Distribution of labels along with the distribution of the predictions of the Custom LSTM, SVR and Histogram-Gradient Boosting models*

Chapter 6

Conclusion & Future Work

In conclusion, the task of basic block energy consumption prediction constitutes a complex objective and holds great potential for energy optimization in a variety of scenarios, such as scheduling and compiling. Machine learning techniques have emerged as a powerful tool for basic block energy profiling. However, to achieve this, basic blocks must first be represented in a vector space using text representation techniques.

In this work, several distinct machine learning models were developed and implemented that broadly can be categorized into: traditional machine learning regressors, deep neural network architectures. The model training was based on a basic block energy measurement dataset, that was developed using the RAPL tool and consisted the work of a preceding thesis. This dataset contained multiple labels for the same basic blocks, which greatly affected the performance of the models. However, the best performing models managed to capture the basic block energy consumption ranges, achieving 0.25 MAE with the standard deviation of the labels being 0.7.

Future research directions in basic block energy profiling should explore the use of the sequence of basic blocks preceding the basic block, since multiple energy measurements for the same basic block reveal that the energy consumption is not solely dependent on the basic block itself. To facilitate this approach, a dataset containing the basic block energy along with its preceding sequence should be developed.

Moreover, the effect of the hardware platform on the basic block energy consumption should be investigated, in order to evaluate if the same tool for energy profiling can be used for multiple systems. It is also important to examine the way that the state of the system affects the basic block energy consumption, as we noticed that different executions of the same benchmark produce different results.

In addition, Transformer architectures can be implemented to boost the performance of energy prediction using machine learning. Adapting the PalmTree pretrained BERT model to the energy prediction task can prove fruitful. Furthermore, building a deep neural network architecture using hierarchical LSTM architecture similar to [2] to predict basic block energy consumption could also yield promising results.

Ultimately, the goal of basic block energy profiling is to enable energy-aware scheduling and compilation, selecting the most energy-efficient direction. Developing compilers and systems that prioritize energy efficiency will resolve the research challenges in the

basic block energy profiling domain.

The code of the systems implemented in this research is available in `https://github.com/Thodorissio/basic-block-energy-prediction`.

Bibliography

[1] Marcus Hähnel, Björn Döbel, Marcus Völp και Hermann Härtig. *Measuring energy consumption for short code paths using RAPL*. *ACM SIGMETRICS Performance Evaluation Review*, 40(3):13–17, 2012.

[2] Charith Mendis, Alex Renda, Saman Amarasinghe και Michael Carbin. *Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks*. *International Conference on machine learning*, σελίδες 4505–4515. PMLR, 2019.

[3] Young Jun Lee, Sang Hoon Choi, Chulwoo Kim, Seung Ho Lim και Ki Woong Park. *Learning binary code with deep learning to detect software weakness*. *KSII the 9th international conference on internet (ICONI) 2017 symposium*, 2017.

[4] Fei Zuo, Xiaopeng Li, Patrick Young, Lannan Luo, Qiang Zeng και Zhexin Zhang. *Neural machine translation inspired binary code similarity comparison beyond function pairs*. *arXiv preprint arXiv:1808.04706*, 2018.

[5] Steven HH Ding, Benjamin CM Fung και Philippe Charland. *Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization*. *2019 IEEE Symposium on Security and Privacy (SP)*, σελίδες 472–489. IEEE, 2019.

[6] Xuezixiang Li, Yu Qu και Heng Yin. *Palmtree: Learning an assembly language model for instruction embedding*. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, σελίδες 3236–3251, 2021.

[7] Alex J Smola και Bernhard Schölkopf. *A tutorial on support vector regression*. *Statistics and computing*, 14:199–222, 2004.

[8] Stylianos Gavriel. *Stock Market Prediction using Long Short-Term Memory*. B.S. thesis, University of Twente, 2021.

[9] Yong Yu, Xiaosheng Si, Changhua Hu και Jianxun Zhang. *A review of recurrent neural networks: LSTM cells and network architectures*. *Neural computation*, 31(7):1235–1270, 2019.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever και Ruslan Salakhutdinov. *Dropout: a simple way to prevent neural networks from overfitting*. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[11] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta και Masanori Koyama. *Optuna: A next-generation hyperparameter optimization framework*. *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, σελίδες 2623–2631, 2019.

[12] Lev Mukhanov, Pavlos Petoumenos, Zheng Wang, Nikos Parasyris, Dimitrios S Nikolopoulos, Bronis R De Supinski και Hugh Leather. *Alea: A fine-grained energy profiling tool*. *ACM Transactions on Architecture and Code Optimization (TACO)*, 14(1):1–25, 2017.

[13] Gerard Salton και Christopher Buckley. *Term-weighting approaches in automatic text retrieval*. *Information processing & management*, 24(5):513–523, 1988.

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado και Jeff Dean. *Distributed representations of words and phrases and their compositionality*. *Advances in neural information processing systems*, 26, 2013.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser και Illia Polosukhin. *Attention is all you need*. *Advances in neural information processing systems*, 30, 2017.

[16] Jacob Devlin, Ming Wei Chang, Kenton Lee και Kristina Toutanova. *Bert: Pre-training of deep bidirectional transformers for language understanding*. *arXiv preprint arXiv:1810.04805*, 2018.

[17] Douglas C Montgomery, Elizabeth A Peck και G Geoffrey Vining. *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[18] Robert Tibshirani. *Regression shrinkage and selection via the lasso*. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[19] Arthur E Hoerl και Robert W Kennard. *Ridge regression: Biased estimation for nonorthogonal problems*. *Technometrics*, 12(1):55–67, 1970.

[20] Jerome Friedman, Trevor Hastie και Rob Tibshirani. *Regularization paths for generalized linear models via coordinate descent*. *Journal of statistical software*, 33(1):1, 2010.

[21] Léon Bottou. *Stochastic gradient descent tricks*. *Neural Networks: Tricks of the Trade: Second Edition*, σελίδες 421–436, 2012.

[22] Harris Drucker, Christopher J Burges, Linda Kaufman, Alex Smola και Vladimir Vapnik. *Support vector regression machines*. *Advances in neural information processing systems*, 9, 1996.

[23] Sepp Hochreiter και Jürgen Schmidhuber. *Long short-term memory*. *Neural computation*, 9(8):1735–1780, 1997.

[24] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K Nurminen και Zhonghong Ou. *RAPL in Action: Experiences in Using RAPL for Power measurements*. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 3(2):1–26, 2018.

[25] Nils Reimers και Iryna Gurevych. *Sentence-bert: Sentence embeddings using siamese bert-networks*. *arXiv preprint arXiv:1908.10084*, 2019.

[26] Wenbo Guo, Dongliang Mu, Xinyu Xing, Min Du και Dawn Song. *DEEPVSA: Facilitating Value-set Analysis with Deep Learning for Postmortem Program Analysis. USENIX Security Symposium*, σελίδες 1787–1804, 2019.

[27] Tomas Mikolov, Kai Chen, Greg Corrado και Jeffrey Dean. *Efficient estimation of word representations in vector space*. *arXiv preprint arXiv:1301.3781*, 2013.

[28] Corinna Cortes και Vladimir Vapnik. *Support-vector networks*. *Machine learning*, 20:273–297, 1995.

[29] Jerome H Friedman. *Greedy function approximation: a gradient boosting machine*. *Annals of statistics*, σελίδες 1189–1232, 2001.

[30] Jason Brownlee. *Machine learning mastery with Python: understand your data, create accurate models, and work projects end-to-end*. Machine Learning Mastery, 2016.

[31] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[32] Yoshua Bengio, Patrice Simard και Paolo Frasconi. *Learning long-term dependencies with gradient descent is difficult*. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[33] Razvan Pascanu, Tomas Mikolov και Yoshua Bengio. *On the difficulty of training recurrent neural networks*. *International conference on machine learning*, σελίδες 1310–1318. Pmlr, 2013.

[34] Ning Qian. *On the momentum term in gradient descent learning algorithms*. *Neural networks*, 12(1):145–151, 1999.

[35] Tijmen Tieleman, Geoffrey Hinton και others. *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[36] Diederik P Kingma και Jimmy Ba. *Adam: A method for stochastic optimization*. *arXiv preprint arXiv:1412.6980*, 2014.

[37] Kaichao You, Mingsheng Long, Jianmin Wang και Michael I Jordan. *How does learning rate decay help modern neural networks? arXiv preprint arXiv:1908.01878*, 2019.

[38] Samuel L Smith, Pieter Jan Kindermans, Chris Ying και Quoc V Le. *Don't decay the learning rate, increase the batch size*. *arXiv preprint arXiv:1711.00489*, 2017.

[39] Felix A Gers, Jürgen Schmidhuber και Fred Cummins. *Learning to forget: Continual prediction with LSTM*. *Neural computation*, 12(10):2451–2471, 2000.

[40] James Bergstra, Rémi Bardenet, Yoshua Bengio και Balázs Kégl. *Algorithms for hyper-parameter optimization*. *Advances in neural information processing systems*, 24, 2011.

[41] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht και Ameet Talwalkar. *A system for massively parallel hyperparameter tuning*. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot και E. Duchesnay. *Scikit-learn: Machine Learning in Python*. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[43] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt και Gaël Varoquaux. *API design for machine learning software: experiences from the scikit-learn project*. *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, σελίδες 108–122, 2013.

[44] Yann LeCun, Léon Bottou, Genevieve B Orr και Klaus Robert Müller. *Efficient backprop*. *Neural networks: Tricks of the trade*, σελίδες 9–50. Springer, 2002.

[45] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga και Adam Lerer. *Automatic differentiation in PyTorch*. *NIPS-W*, 2017.