Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Έγκαιρη Προανάκτηση Δεδομένων στην Κρυφή Μνήμη με Χρήση Τεχνικών Μηχανικής Μάθησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Στυλιαράς

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
Αθήνα, Σεπτέμβριος 2022

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

Έγκαιρη Προανάκτηση Δεδομένων στην Κρυφή Μνήμη με Χρήση Τεχνικών Μηχανικής Μάθησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Στυλιαράς

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28η Σεπτεμβρίου, 2022.

....	....	....
Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2022

(Υπογραφή)

....

Δημητριος Στυλιαρας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

στην οικογένεια μου

Περίληψη

Η συνεχής ανάγκη για βελτίωση στην αρχιτεκτονική των υπολογιστών έχει οδηγήσει στην εφαρμογή διαφορετικών μηχανισμών για τη βελτίωση της απόδοσης. Η προανάκτηση είναι η διαδικασία φόρτωσης δεδομένων σε μνήμες υψηλότερης ταχύτητας από χαμηλότερες, προτού αυτές χρειαστούν, προκειμένου να μειωθεί ο αριθμός των κύκλων που ένας επεξεργαστής μένει σε αδράνεια. Υπάρχουν διαφορετικοί τύποι μηχανισμών προανάκτησης, αλλά οι περισσότεροι από αυτούς συγκεντρώνουν και αναλύουν τις πιο πρόσφατες προσβάσεις στη μνήμη μαζί με μεταδεδομένα, για να προβλέψουν μελλοντικές διευθύνσεις με διαφορετικούς αλγόριθμους. Ένας μηχανισμός προανάκτησης σχεδιάζεται σύμφωνα με τους δεδομένους πόρους και την πολυπλοκότητα των μοτίβων πρόσβασης στη μνήμη που προσπαθεί να προβλέψει. Ειδικοί μηχανισμοί προανάκτησης είναι δαπανηροί, αλλά μπορούν να αναγνωρίσουν προηγμένα μοτίβα και να βελτιώσουν απαιτητικές εφαρμογές. Οι τεχνικές μηχανικής μάθησης και τα νευρωνικά δίκτυα, παρέχουν λύσεις πρόβλεψης σε πολλά διαφορετικά πεδία και η χρήση τους υπόσχεται βελτιώσεις και στην αρχιτεκτονική υπολογιστών, καθώς τα μοντέλα νευρωνικών δικτύων μέσω εκπαίδευσης μπορούν να προσαρμοστούν σε διαφορετικά μοτίβα πρόσβασης για κάθε εφαρμογή. Στην παρούσα διπλωματική εργασία εστιάζουμε στην προανάκτηση δεδομένων, και εξετάζουμε διάφορους μηχανισμούς προανάκτησης (με ή χωρίς Μηχανική Μάθηση) και εφαρμόζουμε ένα μοντέλο Επαναλαμβανόμενου Νευρωνικού Δικτύου Μακράς Βραχυπρόθεσμης Μνήμης (LSTM) που συνδυάζουμε μαζί με άλλους prefetchers για να λειτουργεί ταυτόχρονα. Το μοντέλο μας αντιμετωπίζει την πρόβλεψη διευθύνσεων μνήμης ως πρόβλημα ταξινόμησης της διαφοράς (δέλτα) διευθύνσεων μνήμης διαφορετικών μετρητών προγραμμάτων στην κρυφή μνήμη τελευταίου επιπέδου. Ένας σημαντικός παράγοντας που λαμβάνουμε υπόψη στην εκπαίδευση μοντέλων είναι ο χρονισμός προανάκτησης, όπως προέκυψε από τον πειραματισμό του ίδιου μοντέλου(LSTM) με μοναδική διαφορά την απόσταση των προβλέψεων. Αντί να στοχεύουμε στην πρόβλεψη των αμέσως επόμενων προσβάσεων μνήμης, στοχεύουμε σε πιο μελλοντικές προβλέψεις υπολογίζοντας τα δέλτα μεταξύ συγκεκριμένου βήματος που προκύπτουν από την ανάλυσή μας. Τα αποτελέσματα που παρουσιάζουμε είναι ενθαρρυντικά για εφαρμογή νευρωνικών δικτύων σε prefetchers που είτε λειτουργούν αυτόνομα είτε ακόμα καλύτερα λειτουργούν συμπληρωματικά με άλλους prefetchers χωρίς μηχανική μάθηση, με τον συδνυασμό του μοντέλου μας και τον Irregular Stream Buffer να παρουσιάζει την καλύτερη και πιο συνεπή επίδοση στο σύνολο των προσομοιώσεων.

Λέξεις Κλειδιά — Κρυφή Μνήμη, Προανάκτηση, Κρυφή Μνήμη Τελευταίου Επιπέδου, Δέλτα, Δείκτης Προγράμματος, Νευρωνικά Δίκτυα, Μακρά Βραχυπρόθεσμη Μνήμη, Μηχανική μάθηση, Επαναλαμβανόμενα Νευρωνικά Δίκτυα

Abstract

The constant need for improvement in computer architecture has led to the implementation of different mechanisms to improve performance. Prefetching is the process of loading data to higher speed memories from lower ones before they are needed in order to reduce the number of cycles a processor stays idle. There are different types of prefetchers, but most of them gather and analyze recent memory accesses along with metadata to extrapolate and predict future addresses with different algorithms. A prefetcher is designed in accordance with the given resources and the complexity of the memory access patterns it tries to predict. Specific or more complex prefetchers are costly, but they can prefetch advanced patterns and applications. Machine Learning Techniques, Neural Networks specifically, provide solutions to different fields in recent years and seem promising for computer architecture. Neural Network Models through training can be adapted to different access patterns for each application. In the current thesis, we examine other commonly used prefetchers (with or without Machine Learning) and implement a Recurrent Long Short Term Memory Neural Network model that we combine with other prefetchers to work simultaneously. Our LSTM model treats memory address prediction as a classification problem of memory address deltas of different Program Counters on the Last Level Cache. An important factor of our model is timeliness by calculating the deltas between addresses with specific distances derived from our analysis with the same LSTM model being tested considering prediction of the next and prediction of an address further in the future. The results we present are promising for Neural Network applications in prefetchers on their own or even better working complementary to other non Machine Learning ones.

Keywords — Cache, Prefetching, Last Level Cache, Delta, Program Counter, Neural Networks, Long Short Term Memory Cells, Machine Learning, Recurrent Neural Networks

Ευχαριστίες

Στην παρούσα διπλωματική εργασία παρουσιάζεται το έργο που πραγματοποίησα στο Εργαστήριο Υπολογιστικών Συστημάτων(CSLab) για την ολοκλήρωση των σπουδών μου στην Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών. Θα ήθελα να ευχαριστήσω:

Τον καθηγητή κ. Νεκτάριο Κοζύρη για την ευκαιρία που μου έδωσε για την εκπόνηση της διπλωματικής εργασίας υπό την επίβλεψη του, καθώς και για την διδασκαλία του σε μαθήματα των σπουδών μου με αποτέλεσμα την ανάπτυξη των γνώσεων και ενδιαφερόντων μου στο αντικείμενο.

Τον Δρ. Βασίλειο Καρακώστα για την εξαιρετική συνεργασία που είχαμε την περίοδο εκπόνησης της εργασίας, καθώς και για την καθοδήγηση, συμβουλές και διορθώσεις του προς την ολοκλήρωση του έργου όσο και για την επίλυση αποριών και εμποδίων που προέκυπταν.

Τους γονείς και αδερφό μου στήριξη που μου προσέφεραν όλα αυτά τα χρόνια, καθώς και την δυνατότητα που μου έδωσαν να ολοκληρώσω τις σπουδές μου.

Τέλος, θέλω να ευχαριστήσω τους φίλους μου που ήταν κοντά μου καθ’ όλη τη διάρκεια των σπουδών μου, για την αμέριστη συμπαράσταση τους και τις εμπειρίες που μοιραστήκαμε

Δημήτριος Στυλιαράς
Σεπτέμβριος 2022

Contents

List of Figures

List of Tables

Chapter 1

Εκτενής Περίληψη

1.1 Εισαγωγή

1.1.1 Θέμα Διπλωματικής

Η αύξηση των υπολογιστικών δυνατοτήτων έχει επιτρέψει στην εκτεταμένη χρήση και έρευνα της μηχανικής μάθησης σε μία πληθώρα εφαρμογών. Η χρήση τέτοιων τεχνικών στην αρχιτεκτονική υπολογιστών έχει μελετηθεί κυρίως τα τελευταία χρόνια [1, 2]. Υπάρχουν τμήματα αρχιτεκτονικής όπου η χρήση της μηχανικής μάθησης έχει παρουσιάσει υποσχόμενα αποτελέσματα και σε ορισμένες περιπτώσεις όπως στην πρόβλεψη διακλάδωσης προγραμμάτων [3], τα έχει ξεπεράσει.

Τα σύγχρονα συστήματα έχουν δομημένη ιεραρχία μνημών με κάθε επόμενο επίπεδο να έχει διαφορετικά χαρακτηριστικά ταχύτητας και μεγέθους από το προηγούμενο με σκοπό τα πιο συχνά χρησιμοποιούμενα δεδομένα να βρίσκονται σε υψηλότερα επίπεδα ταχύτερης μνήμης. Το πλησιέστερο στον επεξεργαστή και πιο γρήγορο επίπεδο είναι η κρυφή μνήμη, η οποία με τη σειρά της έχει ιεραρχία.

Στην κρυφή μνήμη όμως οι αστοχίες παραμένουν δημιουργώντας καθυστερήσεις για τον επεξεργαστή. Προς την αποφυγή αυτών έχει εισαχθεί ο μηχανισμός προανάκτησης δεδομένων, που προσπαθεί να προβλέψει τις διευθύνσεις μνήμης που θα χρειαστεί ο επεξεργαστής και να της προφορτώσει στην κρυφή μνήμη.

Συγκεκριμένα, η παρούσα διπλωματική μελετάει την εφαρμογή τεχνικών μηχανικής μάθησης και νευρωνικών δικτύων στην προφόρτωση δεδομένων στο τελευταίο επίπεδο κρυφής μνήμης. Το πρόβλημα της πρόβλεψης διευθύνσεων μνήμης είναι παρόμοιο με το πρόβλημα πρόβλεψης επόμενης λέξης της Επεξεργασίας Φυσικών Γλωσσών [4]. Επομένως η προσέγγιση είναι παρόμοια με χρήση Επαναλαμβανόμενων Νευρωνικών Δικτύων(Recurrent Neural Networks) [5], όπως τα Δίκτυα Μακράς Βραχύχρονης Μνήμνης (Long Short Term Memory Networks) [6].

1.1.2 Προσέγγιση και Συνεισφορές

Για την επίλυση του προβλήματος προανάκτησης εμπνευστήκαμε κυρίως από τις εργασίες [7, 8]. Στην εργασία μας το νευρωνικό δίκτυο αντιμετωπίζει το πρόβλημα πρόβλεψης ως κατηγοριοποίηση των διαφορών των διευθύνσεων μνήμης. Οι διαφορές αυτές υπολογίζονται μεταξύ διευθύνσεων που απέχουν συγκεκριμένο αριθμό εντολών φόρτωσης του ίδιου Δείκτη Προγράμματος (Program Counter). Ο μηχανισμός προανάκτησης λειτουργεί στο τελευταίο επίπεδο κρυφής μνήμης, όπου χρησιμοποιεί τις ροές δεδομένων που φτάνουν σε αυτό ανεξαρτήτως της ύπαρξης τους στην κρυφή μνήμη.

Αλληλουχίες των διαφορών διευθύνσεων μαζί με τους Δείκτες Προγράμματος χρησιμοποιούνται ως είσοδοι στο Νευρωνικό Δίκτυο, όπου εισάγονται σε επίπεδα ενσωμάτωσης (embeddings) τα οποία ύστερα γίνονται είσοδοι του Επαναλαμβανόμενου Νευρωνικού Δικτύου LSTM.

Το μοντέλο εκπαιδεύτηκε και αξιολογήθηκε στις σουίτες αναφοράς SPEC06, SPEC17, και GAP με χρήση του προσομοιωτή ChampSim. Τα αποτελέσματα είναι πολλά υποσχόμενα για την χρήση νευρωνικών δικτύων στο πρόβλημα της προανάκτησης, κυρίως σε συνδυασμό με άλλες παραδοσιακές τεχνικές.

1.2 Θεωρητικό Υπόβαθρο

1.2.1 Προανάκτηση Δεδομένων

Κρυφή μνήμη και Προανάκτηση

Η μεγάλη διαφορά ταχύτητας της κύριας μνήμης(RAM) και επεξεργαστών έχει οδηγήσει στην εύρεση τεχνικών γεφύρωσης του χάσματος προς αποφυγή καθυστερήσεων κύκλων στον επεξεργαστή καθώς αναμένει την λειτουργία της μνήμης. Η αρχιτεκτονική μνημών έχει οδηγηθεί σε ιεραρχική δομή εισάγοντας επιπλέον επίπεδα ταχύτερης, αλλά μικρότερης και ακριβότερης μνήμης εκμεταλλευόμενη την χρονική και χωρική τοπικότητα των προσβάσεων μνήμης. Στα σύγχρονα συστήματα έχει εισαχθεί η κρυφή μνήμη (cache) πολλών επιπέδων (συνήθως 3) που αποθηκεύει χρησιμοποιούμενα δεδομένα και εντολές σε μνήμη ταχύτητας κοντινής τάξης με τον επεξεργαστή.

Η χρήση κρυφών μνημών δεν μπορεί να εξαλείψει την εισαγωγή καθυστερήσεων στον επεξεργαστή όταν χρειάζεται να γίνει πρόσβαση σε δεδομένα. Προς την περαιτέρω μείωση καθυστερήσεων και αύξηση απόδοσης συστημάτων χρησιμοποιούνται μηχανισμοί πρόβλεψης και προανάκτησης διευθύνσεων στην κρυφή μνήμη, για άμεση χρήση τους από τον επεξεργαστή όταν τα χρειαστεί

Μετρικές για μηχανισμούς προανάκτησης

Οι μηχανισμοί προανάκτησης αξιολογούνται λαμβάνοντας υπό όψιν συγκεκριμένες μετρικές. Αυτές είναι οι παρακάτω

- **Ακρίβεια**: Το ποσοστό των χρήσιμων προφορτώσεων προς το σύνολο όλων των άχρηστων και χρήσιμων.
- **Κάλυψη**: Το ποσοστό των χρήσιμων προφορτώσεων προς των αριθμό των αστοχιών μνήμης χωρίς την χρήση τέτοιου μηχανισμού
- **Χρονική Συνέπεια**: Η χρονική τοποθέτηση προανάκτησης σε σύγκριση με την χρήση της συγκεκριμένης διεύθυνσης.

Τέλος η αξιολόγηση ενός μηχανισμού μετριέται από την επίπτωση του στον αριθμό εντολών ανά κύκλο(IPC). Ο αριθμός αυτός επηρεάζεται από πολλούς παράγοντες, αλλά για την αξιολόγηση μηχανισμού προανάκτησης κρατούνται όλες οι υπόλοιπες μεταβλητές σταθερές.

Χαρακτηριστικά προανάκτησης

Τα χαρακτηριστικά προανάκτησης ενός μηχανισμού είναι τα δεδομένα προανάκτησης, ο χρονισμός της προανάκτησης και το επίπεδο μνήμης.

Από τα σημαντικότερα χαρακτηριστικά που καθορίζουν την επιτυχία ενός μηχανισμού προανάκτησης είναι ο καθορισμός ποιών δεδομένων θα προφορτωθούν στην μνήμη. Αυτό πραγματοποιείται με την αναγνώριση μοτίβων πρόσβασης και προσδιορισμού των δεδομένων που προκαλούν αστοχίες κρυφής μνήμης.

Ο χρονισμός της προανάκτησης είναι καίριος στην αποτελεσματικότητα της, καθώς η προφόρτωση χρειάζεται χρόνο και η καθυστερημένη ολοκλήρωση της την αχρηστεύει ή χειρότερα πληγώνει την επίδοση του συστήματος, λόγω περιορισμένης κρυφής μνήμης. Συνήθως η έκδοση εντολής προανάκτησης εξαρτάται από γεγονότα του επεξεργαστή και προγράμματος όπως πρόσβαση σε μνήμη, αστοχία σε κρυφή μνήμη, διακλάδωση κτλ.

Ένας μηχανισμός προανάκτησης μπορεί να υλοποιηθεί σε διαφορετικά επίπεδα μνήμης, συνήθως

Μοτίβο	Αναδρομική Σχέση $\mathcal{R}$	Παράδειγμα	Σημειώσεις
Σταθερό	$A_n = A_{n-1}$	*ptr	
Δέλτα	$A_n = A_{n-1} + d$	διάσχιση πίνακα	d = stride
Δέλτα Δείκτη	$A_n = Ld(A_{n-1})$	next = current->next	Φόρτωση διεύθυνσης από το προηγούμενο
Έμμεσο Δέλτα	$A_n = Ld(B_{n-1} + d)$	*(M[i])	
Έμμεσο ευ-ρετήριο	$A_n = Ld(B_{n-1+c} + d)$	M[N[i]]	c = βασική διεύθυνση του M, d=stride

Table 1.1: Κατηγοριοποίηση Μοτίβων Πρόσβασης Μνήμης [9]

Μηχανική Κατηγοριοποίηση	Αντίστοιχο μοτίβο
Σταθερό	Σταθερό
Πρόσθεση	Δέλτα
Πρόσθεση, Πολλαπλασιασμός	Σύνθετο
Φόρτωση, Πρόσθεση	Συνδεδεμένη λίστα, έμμεσο ευρετήριο
Φόρτωση, Πρόσθεση, Πολλαπλασιασμός	Πολλαπλασιασμός

Table 1.2: Λειτουργίες μηχανής και μοτίβα [9]

μεταξύ της κύριας μνήμης(RAM) και της κρυφής ή μεταξύ επιπέδων της κρυφής μνήμης. Επίσης μπορεί ανάλογα με το επίπεδο να λειτουργεί με εικονικές ή φυσικές διευθύνσεις μνήμης.

Μία διαφορετική πτυχή ενός μηχανισμού προανάκτησης κρυφής μνήμης είναι η τοποθέτηση των προβλεπόμενων δεδομένων. Υπάρχουν δύο περιπτώσεις τοποθέτησης τους, απευθείας στην κρυφή μνήμη ή σε μια βοηθητική δομή μνήμης που απομονώνει τις προφορτώσεις από την κρυφή μνήμη με σκοπό την μείωση συγκρούσεων κρυφής μνήμης.

Τέλος, πολύ σημαντικό χαρακτηριστικό προανάκτησης είναι το μοντέλο αναγνώρισης μοτίβων μνήμης με σκοπό την πρόβλεψη. Εκτενής ανάλυση των μοτίβων γίνεται στην εργασία Understanding Memory Access Patterns [9]. Συνοπτικά παρουσιάζονται στους παρακάτω πίνακες Table 1.1 και Table 1.2

Προσεγγίσεις προανάκτησης

Υπάρχουν δύο κύριες κατηγορίες προανάκτησης, η προανάκτησης εντολών και η προανάκτησης δεδομένων.

Ακόμα οι μηχανισμοί προανάκτησης έχουν δύο σημαντικούς τρόπους λειτουργίας, την προανάκτησης μέσω λογισμικού και την προανάκτηση μέσω υλικού.

Προανάκτησης Λογισμικού. Οι μηχανισμοί προανάκτησης λογισμικού επιτρέπουν σε προγραμματιστή ή μεταγλωττιστή να εκτελέσει προανάκτησης εισάγοντας αντίστοιχες εντολές στο σύνολο εντολών είτε μέσω λογισμικά προγραμματίσιμων καταχωρητών. Παραδείγματα λογισμικών μηχανισμών προανάκτησης είναι τα εξής [10] [11] [12].

Προανάκτησης Υλικού. Η προανάκτηση υλικού πραγματοποιείται από αφιερωμένο μηχανισμό προανάκτησης στον επεξεργαστή που παρακολουθεί τις προσβάσεις μνήμης ή εντολές, προβλέπει μελλοντικές προσβάσεις και εκτελεί την προφόρτωση. Υπάρχουν πολλοί διαφορετικοί μηχανισμοί ανάλογα την περίσταση, το κόστος και την εφαρμογή. Οι βασικότεροι είναι:

- Η ακολουθιακή προανάκτηση είναι η πρόβλεψη διευθύνσεων ως μέρος ακολουθίας όπως με πρόσθεση σταθερού βήματος στην τρέχουσα διεύθυνση. Ακόμα υπάρχουν άλλοι ακολουθιακοί μηχανισμοί προανάκτησης με μεγαλύτερη προσαρμοστικότητα[13] [14] [15]
- Προανάκτηση βασισμένη σε ιστορικό: Ο πιο διαδεδομένος μηχανισμός είναι η προανάκτησης βασισμένη σε ιστορικό, όπου αποθηκεύονται ιστορικό μεταδεδομένων προσβάσεων μνήμης ή και ιστορικό αστοχιών. Υπάρχουν πολλές παραλλαγές είτε εμπνευσμένες [16] [17] από TAGE [18] πρόβλεψη διακλαδώσεων, είτε μέσω συσχέτισης διευθύνσεων με Markov αλυσίδες [19], είτε με Απομονωτή Παγκόσμιας Ιστορίας[20]
- Προανάκτηση μέσω Προκαταβολικής Εκτέλεσης: Σε πολυπύρηνα ή πολυνηματικά συστήματα ή και όταν υπάρχει αδρανής πυρήνας μπορούν να υπολογιστούν διευθύνσεις για προανάκτηση. Αυτό μπορεί να πραγματοποιηθεί με ξεχωριστή μηχανή [21] που δημιουργεί γραφήματα εξάρτησης είτε με εκτέλεση μελλοντικών εντολών από διαφορετικά νήματα [22], είτε με ακριβή εκτέλεση επόμενων εντολών [23].

Συνδυαστική Προανάκτηση. Ο συνδυασμός χρήσης λογισμικού και υλικού για προανάκτηση σε αρμονία μπορεί να εκμεταλλευτεί τα πλεονεκτήματα και των δύο μεθόδων, αλλά χρειάζεται προσοχή στην εξισορρόπηση της χρήσης πόρων. Υπάρχει η μέθοδος παράλληλης εκτέλεσης σε πολυπύρηνα συστήματα [24] με ικανοποιητικά αποτελέσματα.

Προκλήσεις στην Προανάκτηση

Μόλυνση Κρυφής μνήμης και κοινή χρήση πόρων. Ο τρόπος λειτουργίας μηχανισμού προανάκτησης επηρεάζει την αποτελεσματικότητα του καθώς μπορεί να οδηγήσει και σε απώλεια απόδοσης. Η επιθετική ή πιο άστοχη προανάκτηση μπορεί να μολύνει την κρυφή μνήμη καθώς λόγω περιορισμένου μεγέθους οι άστοχες προφορτώσεις οδηγούν σε αντικατάσταση χρήσιμων δεδομένων της κρυφής μνήμης. Η χρήση απομονωτών προανάκτησης είναι τρόπος αντιμετώπισης αυτού του προβλήματος, αλλά ο ανταγωνισμός για το περιορισμένο εύρος ζώνης της μνήμης παραμένει και χρειάζεται προσοχή.

Προβλήματα απόδοσης. Η διαφορετικότητα των συστημάτων, κρυφών μνημών και εφαρμογών δυσκολεύει τον συγχρονισμό της προανάκτησης, όπως και η εκτέλεση εντολών εκτός σειράς

Παράγοντες Υλοποίησης. Κάθε μηχανισμός προανάκτησης έχει διαφορετικές ανάγκες σε μνήμη και υπολογιστική ισχύ για τον υπολογισμό των μοτίβων πρόσβασης μνήμης. Αυτές οι ανάγκες μπορεί να καλυφθούν με επιπλέον υλικό αφιερωμένο στην προανάκτηση με αυξημένο κόστος ή να ανταγωνίζεται τις εφαρμογές για την χρήση κοινών πόρων.

1.2.2 Μηχανική Μάθηση

Η Μηχανική Μάθηση είναι πεδίο της Τεχνητής Νοημοσύνης και μελετά αλγόριθμους που βελτιώνονται μέσω εμπειρίας με χρήση δεδομένων για κάποιο συγκεκριμένο σκοπό. Ένα σύστημα Μηχανικής Μάθησης μοντελοποιείται από μια απλή συνάρτηση $Y = f(X, \theta)$, με X την είσοδο και θ τις παραμέτρους. Μέσω εκπαίδευσης αξιολογεί το αποτέλεσμα της συνάρτησης και το χρησιμοποιεί για επαναπροσδιορισμό των παραμέτρων. Ένα σύστημα Μηχανικής Μάθησης αποτελείται από τα εξής:

- **Μοντέλο**: Το εκπαιδευόμενο παραμετρικό σύστημα
- **Σύνολο δεδομένων**: Δεδομένα για εκπαίδευση και αξιολόγηση
- **Συνάρτηση απώλειας ή απόδοσης**: Η συνάρτηση μετρικής για αξιολόγηση του μοντέλου
- **Αλγόριθμο βελτιστοποίησης**: Αλγόριθμος επαναπροσδιορισμού παραμέτρων

Οι κατηγορίες μοντέλων Μηχανικής Μάθησης βάσει των τύπο του συνόλου δεδομένων είναι τρεις.

Επιβλεπόμενη Μάθηση: Το σύνολο δεδομένων είναι οργανωμένο σε ζευγάρια εισόδου(χαρακτηριστικών)-εξόδου με σκοπό την χρήση της πραγματικής τιμής εξόδου για την εκπαίδευση. Υπάρχουν δύο κατηγορίες Επιβλεπόμενης Μάθησης: η Παλινδρόμηση(Regression), δηλαδή η πρόβλεψη συνεχών μεταβλητών σε συγκεκριμένο εύρος και η Ταξινόμηση(Classification), δηλαδή η αντιστοίχηση της εξόδου σε κατηγορίες, τάξεις ή ακεραίους.

Μη Επιβλεπόμενη Μάθηση: Ο στόχος της Μη Επιβλεπόμενης Μάθησης είναι ο εντοπισμός μοτίβων και συσχετίσεων στα δεδομένα με σκοπό την δημιουργία συστάδων, ομάδων, μείωση διάστασης δεδομένων ή δημιουργία νέων αναπαραστάσεων.

Ενισχυτική Μάθηση: Η Ενισχυτική Μάθηση δεν χρησιμοποιεί συγκεκριμένο σύνολο δεδομένων για εκπαίδευση, αλλά ανταμοιβές για την μάθηση πράκτορα σε συγκεκριμένο περιβάλλον. Το σύστημα τροφοδοτείται με σύνολο επιτρεπόμενων πράξεων, κανόνων αλληλεπίδρασης και συναρτήσεις ανταμοιβής, που χρησιμοποιεί για εξερεύνηση των πιθανών ενεργειών σε ένα περιβάλλον με μέθοδο παρόμοια στην δοκιμή και σφάλμα.

1.2.3 Νευρωνικά Δίκτυα

Τα τεχνητά Νευρωνικά Δίκτυα είναι σημαντικό κομμάτι Μηχανική Μάθησης μοντελοποιώντας μαθηματικά την επεξεργασία πληροφορίας όπως παρόμοια βιολογικά συστήματα.

Αντίληπτρο

Το Αντίληπτρο(Perceptron) παρομοιάζει την λειτουργία νευρώνα και είναι το θεμελιώδες στοιχείο για όλα τα νευρωνικά δίκτυα.

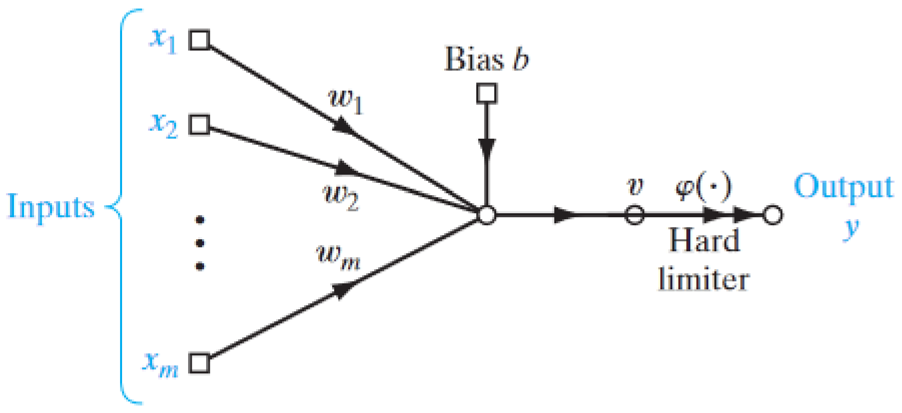


Figure 1.1: Αντίληπτρο [25]

Πολυεπίπεδο Αντίληπτρο

Το πολυεπίπεδο αντίληπτρο είναι ένα νευρωνικό δίκτυο τροφοδοσίας αποτελείται από πολλά επίπεδα αντιλήπτρων μεταξύ τους πλήρως συνδεδεμένα.

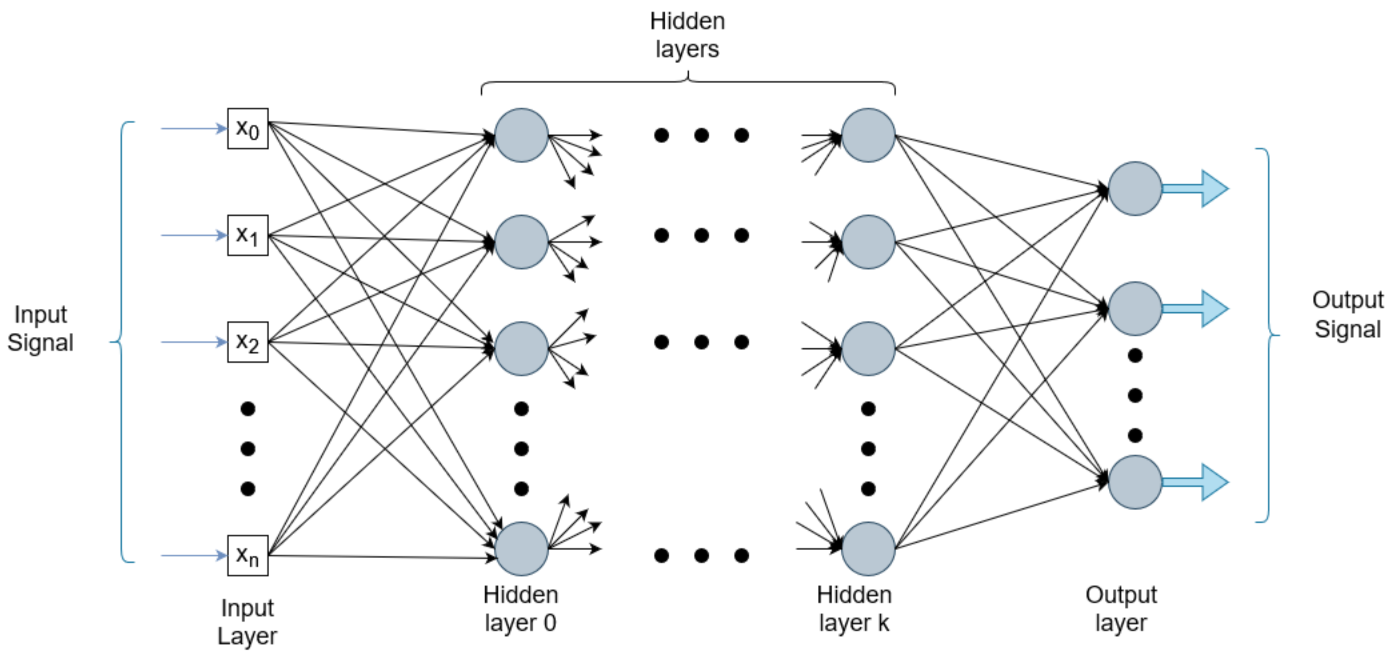


Figure 1.2: Πολυεπίπεδο Αντίληπτρο

Αλγόριθμος Εκπαίδευσης

Η εκπαίδευση ενός μοντέλου είναι διαδικασία βελτιστοποίησης, όπου με την έξοδο του υπολογίζεται η συνάρτηση απώλειας και βάσει αυτής προσαρμόζονται οι παράμετροι του με σκοπό το καλύτερο αποτέλεσμα. Για την προσαρμογή των παραμέτρων χρησιμοποιούνται δυο αλγόριθμοι. Ο αλγόριθμος Οπισθοδιάδοσης(Backward Propagation) υπολογίζει τις μερικές παραγώγους του λάθους προς κάθε βάρους για επιμερισμό της συμβολής κάθε κόμβου και ο επαναληπτικός αλγόριθμος ενημερώνει τις παραμέτρους(βάρη) του συστήματος χρησιμοποιώντας το αποτέλεσμα οπισθοδιάδοσης.

1.2.4 Επαναλαμβανόμενα Νευρωνικά Δίκτυα

Τα επαναλαμβανόμενα Νευρωνικά Δίκτυα(Recurrent είναι κατηγορία νευρωνικών για επεξεργασία σειριακών δεδομένων, είτε χρονικών είτε τοπικών. Πέρα από τα εκάστοτε δεδομένα εισόδου συσσωρεύει την πληροφορία του παρελθόντος σε εσωτερική κατάσταση για υπολογισμό της νέας.

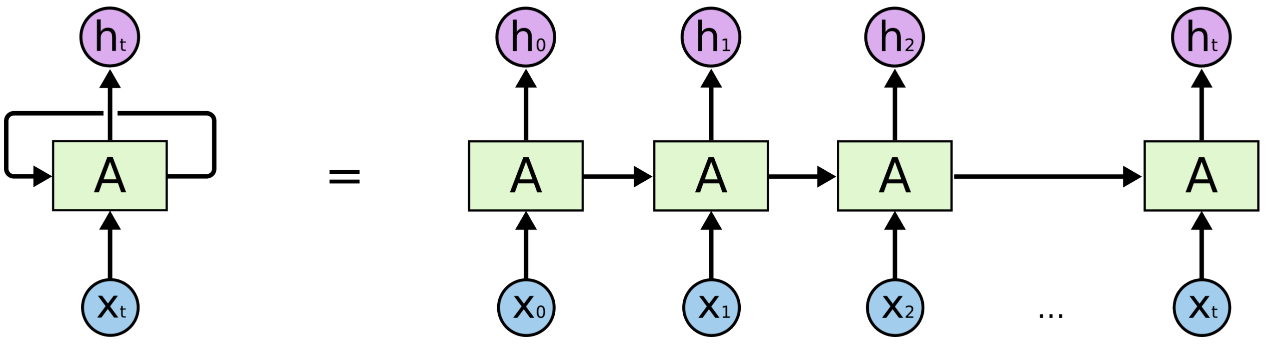


Figure 1.3: Ξεδιπλωμένο Επαναλαμβανόμενο νευρωνικό

Νευρωνικά Δίκτυα Μακράς Βαρχυχρόνια Μνήμη

Τα Νευρωνικά Μακράς Βραχυχρόνιας Μνήμης(LSTM) αποτελούνται από τρεις πύλες, την πύλη λήθης, την ενημέρωσης και της εξόδου. Τα πλεονεκτήματα του είναι η αντιμετώπιση του προβλήματος εξαφανιζόμενων κλίσεων(Vanishing Gradient) και η διατήρηση εξαρτήσεων σε μεγάλες αποστάσεις ακολουθιών. Η κυψέλη του παρουσιάζεται στο Figure 1.4

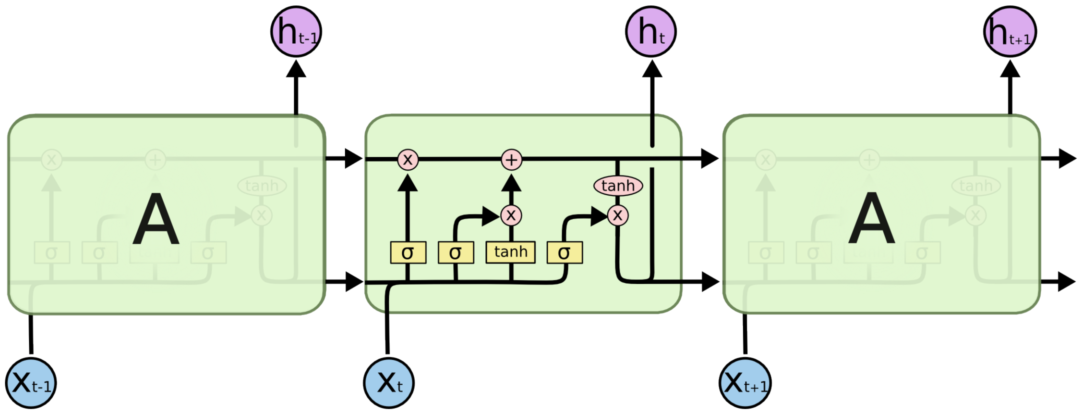


Figure 1.4: LSTM κυψέλη

Αναπαράσταση Κατηγοριών

Υπάρχουν δύο διαδεδομένοι τρόποι κωδικοποίησης της κατηγορίας ως χαρακτηριστικό για νευρωνικά δίκτυα. Το One-hot Encoding είναι η κωδικοποίηση της κατηγορία σε διάνυσμα διάστασης όσης ο αριθμός των κατηγοριών, όπου τα στοιχεία του είναι 0 εκτός από την θέση που αντιστοιχεί στην εκάστοτε κατηγορία. Οι ενσωματώσεις (Embeddings) κωδικοποιούν της κατηγορίες σε εκπαιδευόμενο διάνυσμα πραγματικών τιμών αναπαριστώντας σχέσεις και αποστάσεις μεταξύ κατηγοριών. Κύρια χρήση τους είναι στα προβλήματα Φυσικής Γλώσσας [26].

1.3 Σχετική Βιβλιογραφία

Best Offset Prefetcher

Ο μηχανισμός προανάκτησης Best Offset(BOP)[27] είναι από τους πιο διαδεδομένους μηχανισμούς προανάκτησης. Πραγματοποιεί προβλέψεις ίδιας σελίδας όταν εκτελούνται εντολές φόρτωσης.

Ο BOP δυναμικά προσαρμόζεται στις εφαρμογές ορίζοντας απόσταση προανάκτησης δοκιμάζοντας συγκεκριμένες τιμές στο πρόσφατο ιστορικό με σκοπό την αξιολόγηση της πρόβλεψης της βάσει της χρησιμότητας της. Η λειτουργία του παρουσιάζεται στο διάγραμμα Figure 1.5

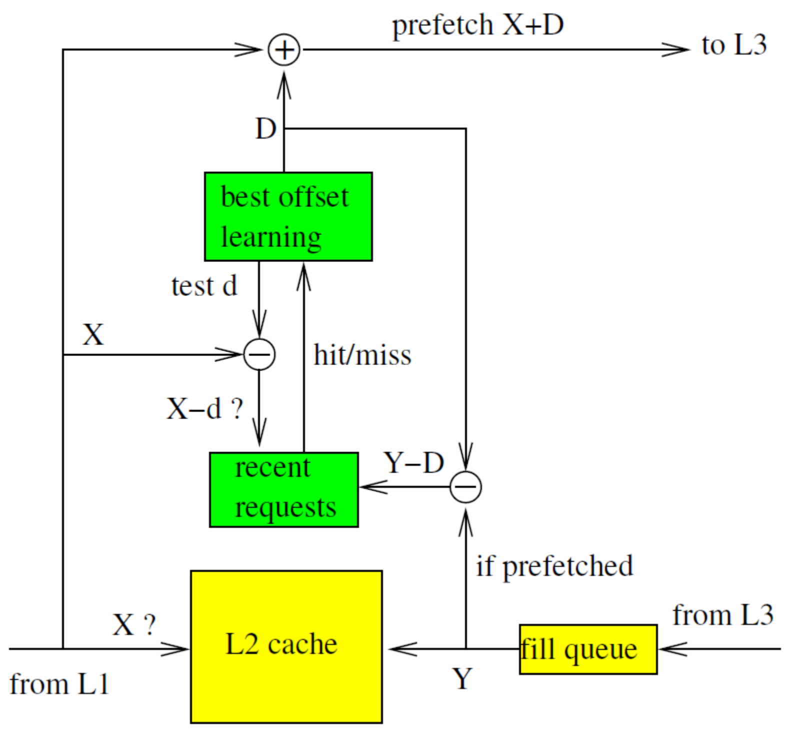


Figure 1.5: Σχηματική όψη του μηχανισμού Best-Offset Prefetcher

Irregular Stream Buffer

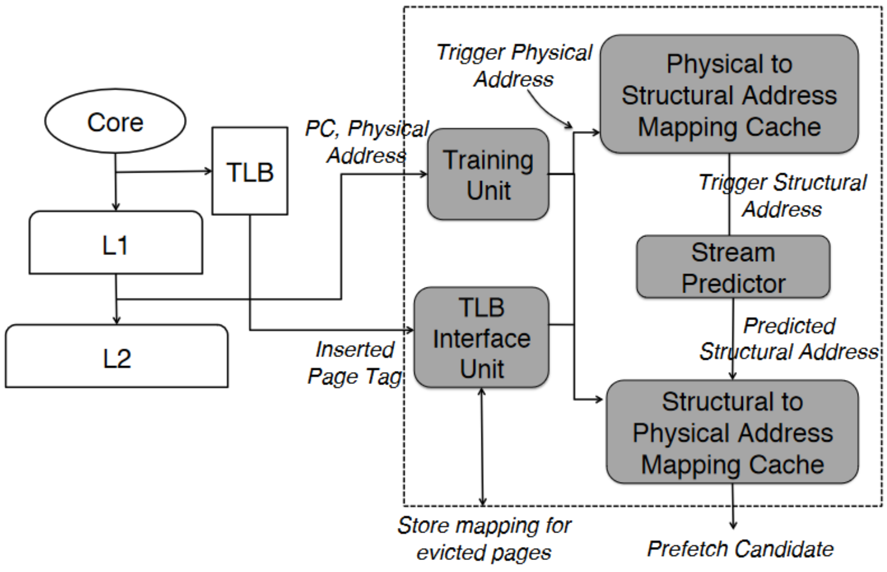

Core
TLB
L1
L2
PC, Physical Address
Training Unit
TLB Interface Unit
Inserted Page Tag
Trigger Physical Address
Physical to Structural Address Mapping Cache
Trigger Structural Address
Stream Predictor
Predicted Structural Address
Structural to Physical Address Mapping Cache
Store mapping for evicted pages
Prefetch Candidate

Figure 1.6: Irregular Stream Buffer

O Irregular Stream Buffer[28] (σχήμα Figure 1.6) είναι μηχανισμός προανάκτησης που βασίζεται στην χρονική συσχέτιση ακανόνιστων ακολουθιών αναφορών μνήμης. Μαθαίνει στις ροές Μετρητών Προγράμματος συσχετίσεις συνεχόμενων φυσικών διευθύνσεων χρησιμοποιώντας πίνακες αντιστοίχησης για χαρτογράφηση των τιμών.

Blue

O BLUE είναι μηχανισμός αφοσιωμένος στον χρονισμό των προφορτώσεων, που βασίστηκε στον Berti[29] και τον entangling prefetcher[30] επεκτείνοντας σε πρόβλεψη μεταξύ σελίδων. O Berti παρουσιάζεται στο Figure 1.7 O Berti προσπαθεί ελέγχοντας τις τελευταίες αιτήσεις προανάκτησης να εντοπίσει τον χρόνο που η προανάκτηση θα ήταν επιτυχής για την αποφυγή της αστοχίας και να τον βαθμολογήσει το χρονικό δέλτα της προανάκτησης(berti delta). Χρησιμοποιεί το υψηλότερα βαθμολογημένο δέλτα για να κάνει την πρόβλεψη από τους πίνακες ιστορικού. Στον BLUE εισάγεται ο πίνακας σελίδων και προσβάσεων σε άλλες σελίδες για πρόβλεψη σε περιπτώσεις εκτός ορίων σελίδας. O προηγούμενος μηχανισμός δουλεύει συμπληρωματικά με τον IP-Based entangling prefetcher.

Επόμενη Γραμμή

Η προανάκτηση επόμενης Γραμμής είναι ο απλούστερος μηχανισμός προφορτώνοντας τις επόμενες διευθύνσεις σε κάθε πρόσβαση.

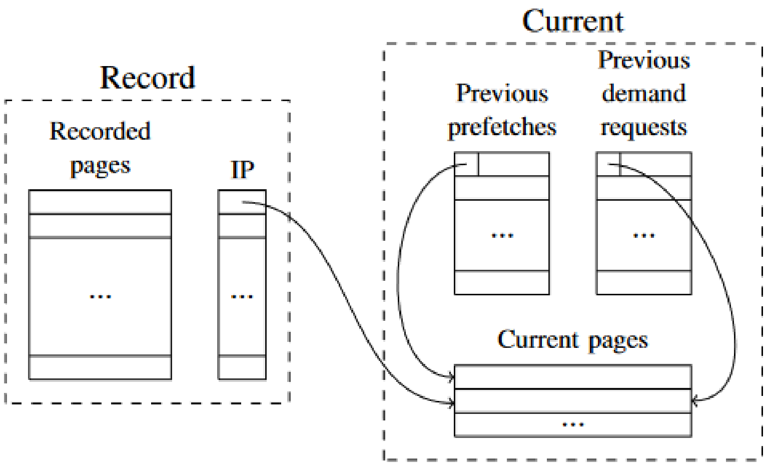


Figure 1.7: Berti prefetcher

Μηχανικής Μάθησης

Μαθαίνοντας Μοτίβα πρόσβασης Μνήμης

Η εργασία Learning Memory Access Patterns[8] είναι η βασική έμπνευση αυτής της διπλωματικής. Υλοποιούνται δυο μοντέλα νευρωνικών δικτύων LSTM για την πρόβλεψη δέλτα διευθύνσεων ως κατηγορίες. Το πρώτο μοντέλο Figure 1.8 χρησιμοποιεί ενσωματώσεις για κωδικοποίηση των συχνότερων Δέλτα και Δεικτών Προγραμμάτων για είσοδο στο LSTM. Η έξοδος του μοντέλου είναι τα 10 πιθανότερα Δέλτα σε κάθε διεύθυνση.

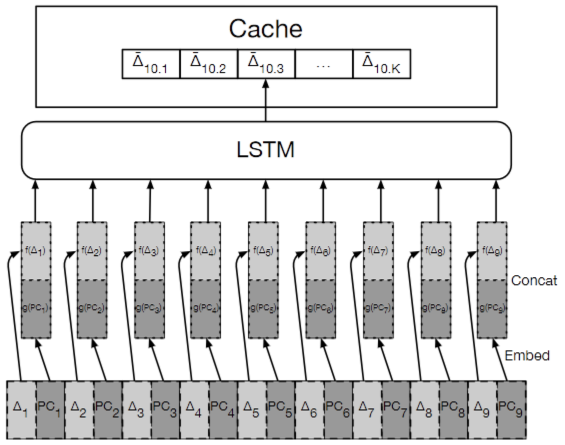


Figure 1.8: Σχηματική όψη του embedding LSTM μοντέλου

Το δεύτερο μοντέλο Figure 1.9 εκμεταλλεύεται την τοπικότητα του χώρου διευθύνσεων και Δεικτών Προγραμμάτων για συσταδοποίηση των διευθύνσεων και χρήση διαφορετικών LSTM για κάθε συστάδα.

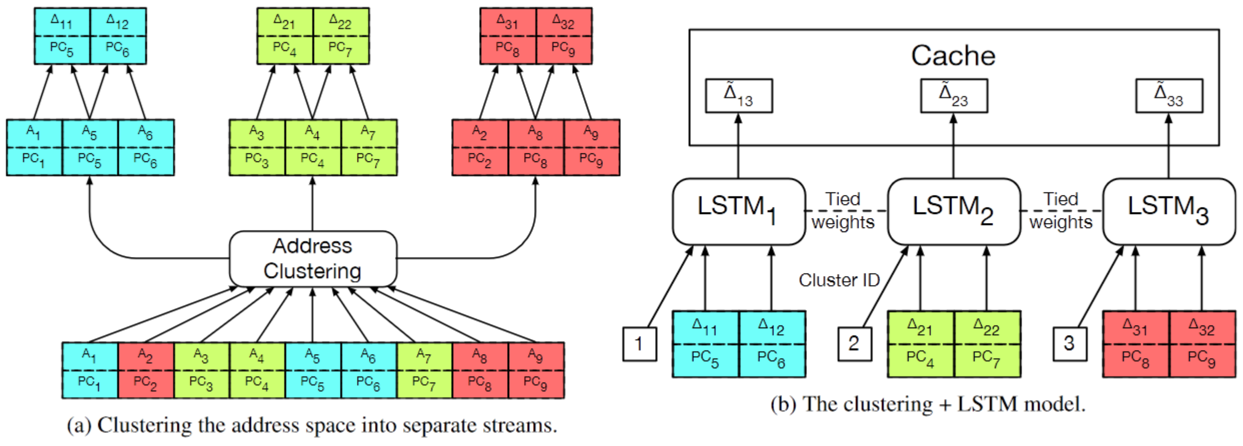

Address Clustering
Cache
LSTM1
LSTM2
LSTM3
Tied weights
Cluster ID
(a) Clustering the address space into separate streams.
(b) The clustering + LSTM model.

Figure 1.9: Σχηματική όψη της clustering LSTM προεπεξεργασίας δεδομένων(a) και του μοντέλου(b)

TransFetch

Το μοντέλο TransFetch[31] είναι μοντέρνα προσέγγιση στο πρόβλημα προανάκτησης χρησιμοποιώντας Transformers και Μηχανισμούς προσοχής[32] για μοντέλο πρόβλεψης. Επιπλέον εισάγει τον κατακερματισμό διευθύνσεων για μείωση του προβλήματος έκρηξης του αριθμού κλάσεων που υπάρχει στην πρόβλεψη διευθύνσεων. Η λειτουργία του εμφανίζεται στο Figure 1.10.

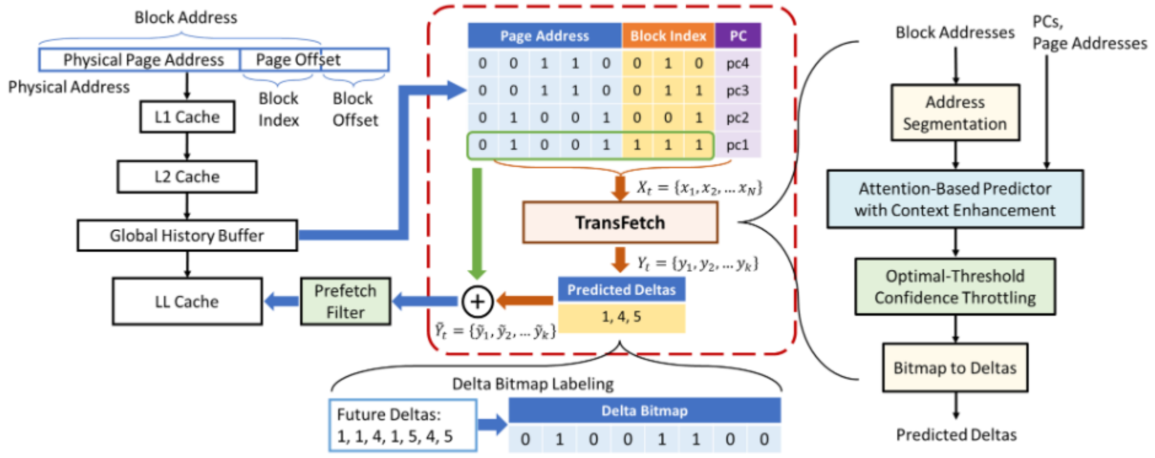


Figure 1.10: Αρχιτεκτονική του TransFetch

Voyager

Το μοντέλο Voyager[33] εισάγει ιεραχική δομή νευρωνικών δικτύων για την πρόβλεψη συσχετίσεων ξεχωριστά σε διευθύνσεις σελίδων και διευθύνσεις block. Χρησιμοποιεί διαφορετικά embeddings για Δείκτες Προγράμματος, διευθύνσεις σελίδων και μπλοκ συνδυάζοντας τα αποτελέσματα των 2 τελευταίων σε ένα επιπλέον embedding και τα αποτελέσματα αυτά τα χρησιμοποιεί ως εισόδους σε διαφορετικά LSTM.Το μοντέλο φαίνεται στο Figure 1.11.

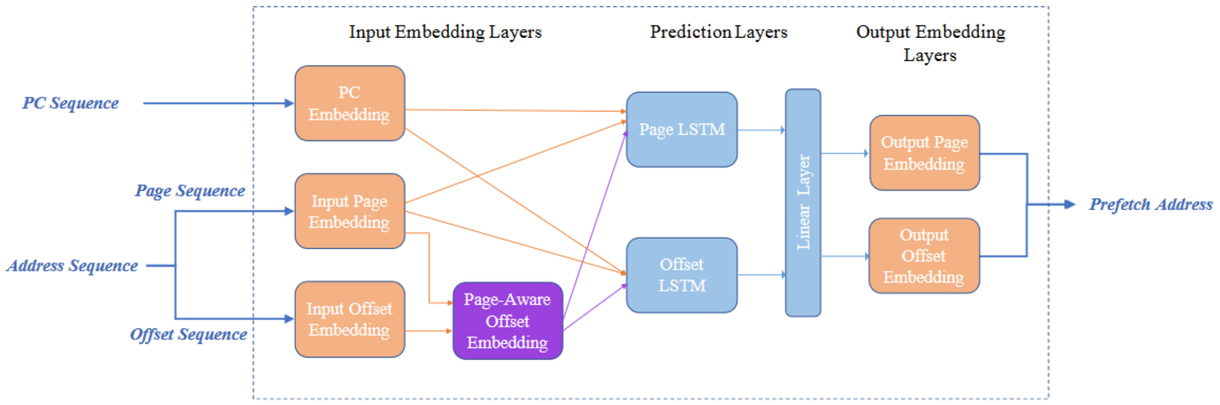


Figure 1.11: Αρχιτεκτονική Voyager

Pythia

Το μοντέλο Pythia[34] είναι διαφορετικό καθώς χρησιμοποιεί Ενισχυτική Μάθηση για προανάκτηση. Το διάνυσμα κατάστασης προκύπτει από την ροή του προγράμματος(Δείκτης Προγράμματος) και την ροή δεδομένων(Διευθύνσεις μνημών) και βάσει αυτού το μοντέλο επιλέγει μια πράξη από ένα σύνολο μετατοπίσεων στα όρια σελίδων. Το αποτέλεσμα αυτής της πράξης αξιολογείται και η πράξη επιβραβεύεται ανάλογα. Ο μηχανισμός επιβράβευσης και η λειτουργία του μοντέλου συνοπτικά περιγράφονται στο Figure 1.12

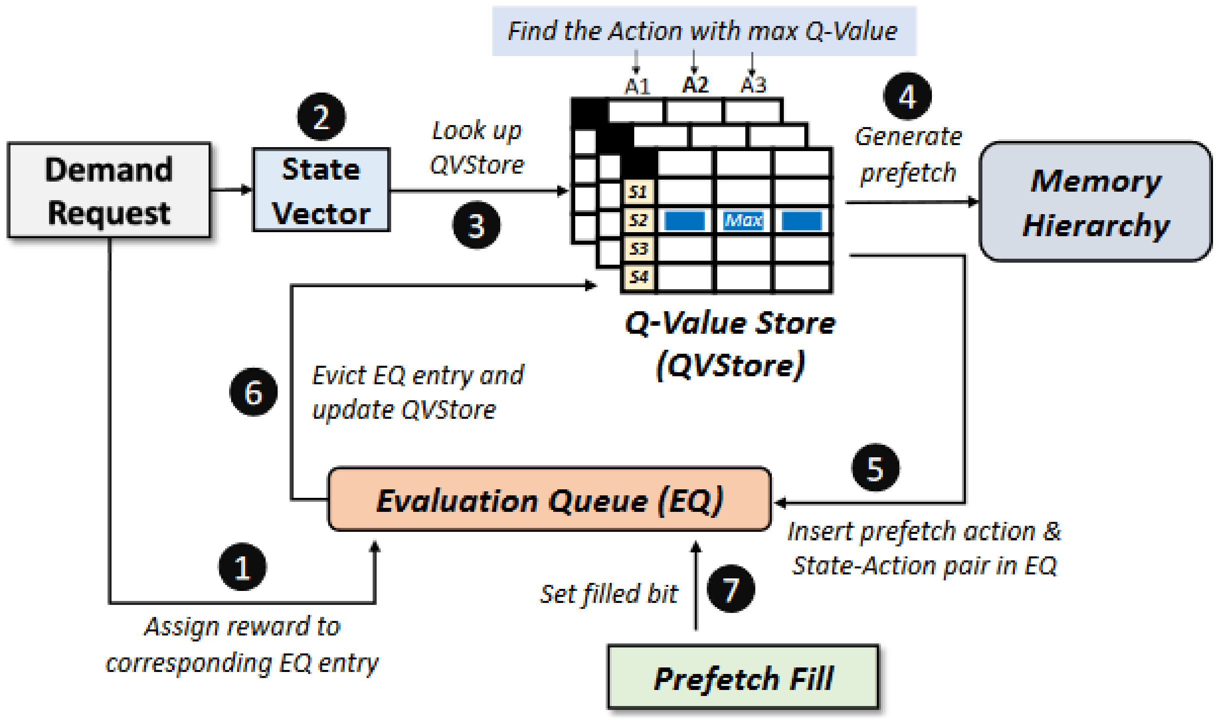


Figure 1.12: Σφαιρική όψη του μοντέλου Pythia

1.4 Ανάλυση, Σχεδίαση και Υλοποίηση

1.4.1 Μαντική Προανάκτηση

Για τον προσδιορισμό της καλύτερης απόστασης προανάκτησης υλοποιήθηκε μοντέλο μαντικής προανάκτησης, που χρησιμοποιούσε τις μελλοντικές διευθύνσεις για προανάκτηση σε διαφορετικές αποστάσεις. Αφού πραγματοποιήθηκε η μαντική προανάκτηση για συνολική απόσταση σε μία εφαρμογή, παρατηρήθηκε ο αυξημένος αριθμός διαφορετικών δέλτα και η δυσκολία πρόβλεψης. Γι αυτό τροποποιήθηκε ο μηχανισμός για εντοπισμό απόστασης μεταξύ ίδιων Δεικτών Προγραμμάτων. Τα αποτελέσματα βρίσκονται στο Section 1.5.1.

1.4.2 Αριθμός Δέλτα

Χρησιμοποιώντας το βήμα της μαντικής προανάκτησης ενιαίου ιστορικού και ανά Δείκτη προγράμματος εντοπίσαμε τον αριθμό διαφορετικών Δέλτα των διευθύνσεων καθώς και το ποσοστό των προσβάσεων που καλύπτουν οι συχνότερες τιμές.

	Global Delta Calculation			Per PC Delta Calculation		
Suite	**Step**	**No of Deltas**	**Filter Coverage %**	**Step**	**No of Deltas**	**Filter Coverage %**
SPEC06	13.48	3415580.48	71.32	7.29	482760.86	87.31
SPEC17	-	-	-	4.51	2351659.70	80.51
GAP	24.26	3386725.16	62.40	8.05	724414.00	86.39

Table 1.3: Στατιστικά των Δέλτα βέλτιστων βημάτων

1.4.3 Μοντέλο Προανάκτησης

Στην παρούσα εργασία το μοντέλο προανάκτησης που χρησιμοποιήθηκε βασίστηκε στο Learning Memory Access Patterns[8] και Understanding Memory Access Patterns [7]. Το πρόβλημα αντιμετωπίζεται ως πρόβλεψη χρονικής σειράς κατηγοριών. Χρησιμοποιούμε ιστορικό τελευταίων N προσβάσεων Δεικτών Προγραμμάτων και Διευθύνσεων Μνήμης εντολών φόρτωσης για την πρόβλεψη επόμενων διευθύνσεων. Προς αποφυγή του μεγάλου αριθμού διαφορετικών διευθύνσεων χρησιμοποιήθηκε η διαφορά διευθύνσεων απόστασης k μεταξύ ίδιων Δεικτών Προγράμματος ως χαρακτηριστικό και στόχος πρόβλεψης.

$$\Delta_N = Addr_{N+k} - Addr_N$$

Ο αριθμός διαφορετικών Δέλτα παραμένει μεγάλος, για αυτό χρησιμοποιήθηκε φίλτρο το πολύ 30000 συχνότερων Δέλτα και η κωδικοποίηση με embedding.

1.4.4 Υλοποίηση Μοντέλου Πρόβλεψης

Pytorch

Το Pytorch[35] είναι βιβλιοθήκη της Python για υλοποίηση μοντέλων μηχανικής μάθησης σε υλικούς επιταχυντές όπως οι κάρτες γραφικών. Προσφέρει ευκολία χρήσης, μεγάλη λειτουργικότητα και πολλές δυνατότητες πειραματισμού.

Υλοποίηση Μοντέλου

Το Νευρωνικό μας Δίκτυο αντιμετωπίζει το πρόβλημα παρόμοια με την Πρόβλεψη Επόμενης Λέξης[4] στη Επεξεργασία Φυσικών Γλωσσών. Χρησιμοποιούμε LSTM μοντέλο αφού έχουν κωδικοποιηθεί τα δεδομένα μας με embedding για να μειωθεί η διάσταση εισόδου από 30000 σε μικρό διάνυσμα πραγματικών τιμών που αναπαριστά και συσχέτιση μεταξύ των κατηγοριών. Το αποτέλεσμα των embedding Δέλτα και Δεικτών προγράμματος εισάγονται σε ένα μοντέλο LSTM για κατηγοριοποίηση, με την έξοδο του LSTM τα τροφοδοτείται σε ένα Γραμμικό

Επίπεδο διάστασης του μεγέθους λεξιλογίου για προσδιορισμό της κατηγορίας. Αφού εφαρμοστεί η συνάρτηση softmax και μετατραπούν οι τιμές του επιπέδου σε πιθανότητες εφαρμόζουμε ένα ήπιο φίλτρο απόρριψης πιθανοτήτων μικρότερων του 0.1 και παράγουμε τα 2 πιο πιθανά αποτελέσματα. Το μοντέλο παρουσιάζεται στο Figure 1.13

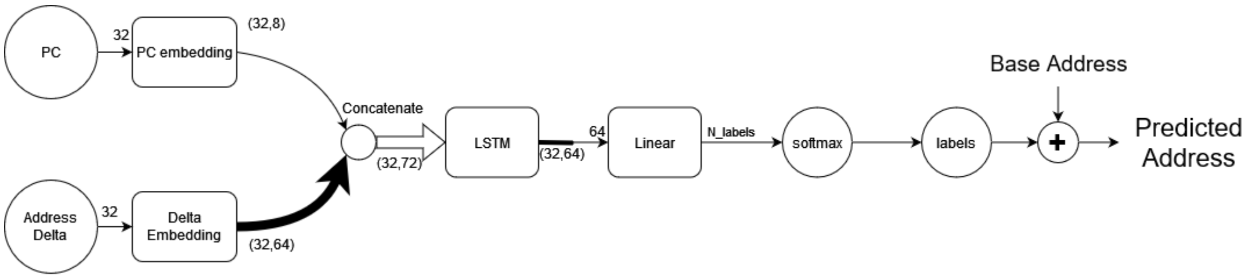


Figure 1.13: Πλήρες Νευρωνικό Μοντέλο για προανάκτηση

Εκπαίδευση Μοντέλου

Για την εκπαίδευση του μοντέλου χρησιμοποιούμε LSTM,χωρίς κατάσταση, γιατί το διαφορετικό προφίλ πρόσβασης σε διαφορετικές φάσεις είναι αρκετά διαφοροποιήσιμο. Η εκπαίδευση πραγματοποιείται σε τυχαία δειγματοληπτούμενα δεδομένα απ' όλη τη διάρκεια του προγράμματος, έχοντας έτσι αρκετά μικρότερο αλλά πλήρως αντιπροσωπευτικό δείγμα δεδομένων για εκαπίδευση(περίπου 6.5% του συνολικού μεγέθους). Το μέγεθος δέσμης δεδομένων εκπαίδευσης πέρα από την επίδραση στην απόδοση επηρεάζει και την επίδοση του μοντέλου λόγω της μη διατήρησης κατάστασης μεταξύ δεσμών. Το μοντέλο εκπαιδεύτηκε με τον ADAM optimizer και αρχικό ρυθμό μάθησης=0.001.

Παράμετροι Μοντέλου

Παράμετρος	Τιμή
Μέγεθος Ενσωμάτωσης Δέλτα	64
Μέγεθος Ενσωμάτωσης Δεικτών Προγράμματος	8
Επίπεδα LSTM	2
Μέγεθος LSTM	64
Μήκος ακολουθίας	32
Μέγεθος Γραμμικού Επιπέδου	30000

Table 1.4: Παράμετροι Μοντέλου

Οι παράμετροι του μοντέλου προέκυψαν από εκτέλεση συγκεκριμένων προγραμμάτων με διαφορετικούς συνδυασμού παραμέτρων και την αξιολόγηση της ακρίβειας και απόδοσης τους.

1.5 Αξιολόγηση

1.5.1 Μεθοδολογία

ChampSim Προσομοιωτής

Για την προσομοίωση και αξιολόγηση των μηχανισμών προανάκτσης χρησιμοποιήθηκε μια τροποποιημένη έκδοσης του ChampSim[36]. Ο ChampSim είναι προσομοιωτής με ίχνη εκτέλεσης για αξιολόγηση αρχιτεκτονικών επιλογών όπως η προανάκτηση μνήμης ή πρόβλεψη διακλάδωσης. Τα ίχνη που λαμβάνει παράγονται από το PIN εργαλείο [37] από εκτελέσεις σε πραγματικά συστήματα, έχοντας την δυνατότητα να προσομοίωση νέα επανεκτέλεση των εφαρμογών.

Στην τροποποιημένη έκδοση χρησιμοποιούμε υλοποιήσεις μοντέλων μηχανισμών προανάκτησης και μοντέλο εισαγωγής ιχνών από αρχεία για την εισαγωγή αποτελεσμάτων των μοντέλων μηχανικής μάθησης.

Μετροπρογράμματα

Για την αξιολόγηση χρησιμοποιήθηκαν 3 σουίτες μετροπρογραμμάτων που χρησιμοποιούνται σε διαγωνισμούς και αξιολόγηση ερευνητικού έργου μοντελοποιώντας πραγματικές εφαρμογές. Αυτές είναι οι Standard Performance Evaluation Corporation 2006 (SPEC06)[38] και 2017(SPEC17)[39] με προγράμματα αριθμητικής ακεραίων και κινητής αριθμητικής. Η τρίτη σουίτα που χρησιμοποιήθηκε είναι η GAP[40] που περιλαμβάνει εφαρμογές επεξεργασίας γράφων.

Μαντική Προανάκτηση

Τα συνολικά αποτελέσματα παρουσιάζονται στον πίνακα Table 1.5

Διατάξεις Μηχανισμών προανάκτησης

Οι διατάξεις προανάκτησης που προσομοιώθηκαν είναι οι εξής:

- Χωρίς μηχανισμό(no)
- Προανάκτηση επόμενης γραμμής (next_line)

Σουίτα	Ενιαία Απόσταση			Ανά Δείκτη Προγράμματος		
	Βήμα	**IPC**	**IPC Βελτίωση**	**Βήμα**	**IPC**	**IPC βελτίωση**
SPEC06	13.48	0.7289969748	73.86	7.29	0.7261998886	73.26
SPEC17	-	-	-	4.51	0.7453825265	45.72
GAP	24.26	0.3360217918	71.11	8.05	0.2759496478	40.32

Table 1.5: Μέσες Τιμές για Μαντική Προανάκτηση ανά Σουίτα

- Best Offset Prefetcher(bo)
- Irregular Stream Buffer(sisb)
- TransFetch
- Our LSTM model (LSTM)

Υλοποιήσαμε και αξιολογήσαμε συνεργατικές διατάξεις κάποιων μηχανισμών προανάκτησης:

- Irregular Stream Buffer με Best Offset Prefetcher(sisb_bo)
- Το μοντέλο μας με προανάκτηση επόμενησ γραμμής (LSTM_next)
- Το μοντέλο μας με Best Offset Prefetcher(LSTM_bo)
- Το μοντέλο μας με Irregular Stream Buffer(LSTM_sisb)

Όλες οι διατάξεις είχαν βαθμό 2 και αξιλογήθηκαν κυρίως στην επίδοση τους στις μετρικές IPC και βελτίωση IPC. Οι υπόλοιπες μετρικές που παρουσιάζονται βοήθησαν στον σχεδιασμό και την κατανόηση των αποτελεσμάτων.

1.5.2 Αποτελέσματα

Απόσταση Προανάκτησης

Η χρήση απόστασης πρόβλεψης αναλύθηκε και στο μοντέλο εκτελώντας το με βήμα 1(LSTM_oneStep) και το ιδανικό βήμα(LSTM_model) και συγκρίνοντας τα αποτελέσματα των εκτελέσεων στο Figure 1.14

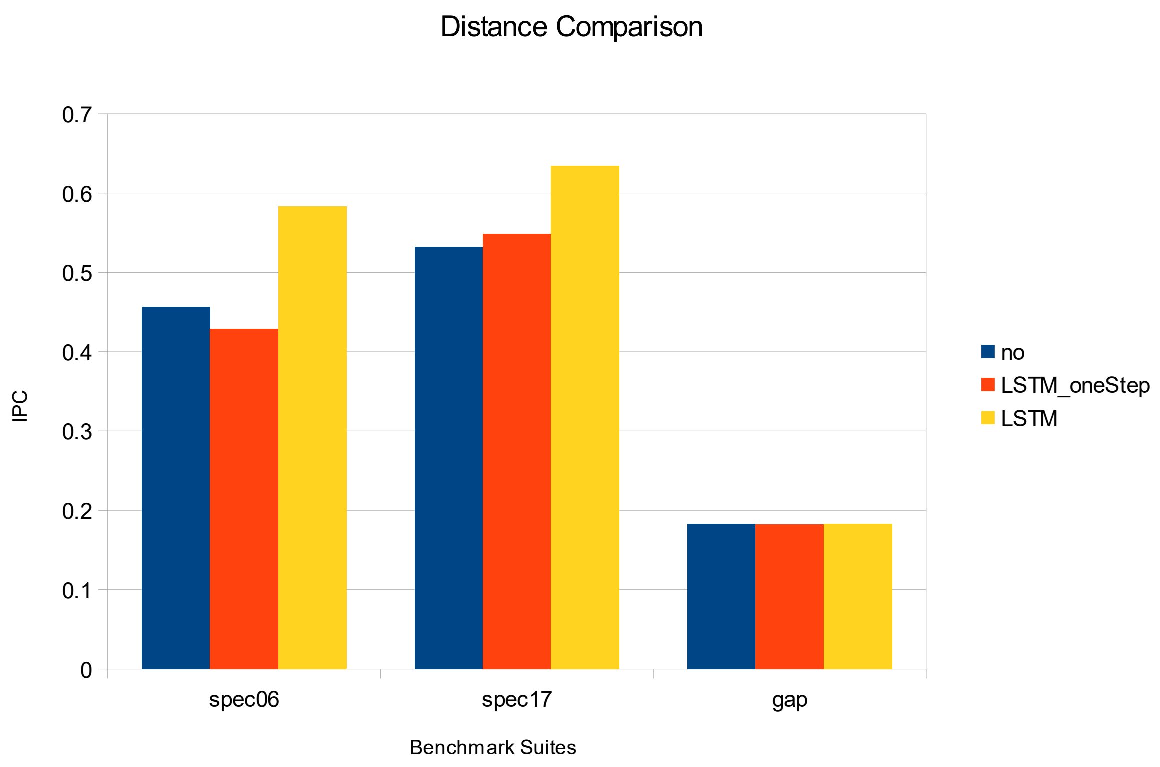


Figure 1.14: IPC Σύγκριση του μοντέλου Βήματος 1, Ιδανικού Βήματος και χωρίς προανάκτηση

Η προανάκτηση με ιδανικό βήμα είναι απαραίτητη για βελτίωση της επίδοσης προανάκτησης κυρίως στα SPEC06, SPEC17. Στο GAP παρουσιάζει λιγότερο πλήγμα στην επίδοση μηδενίζοντας το σχεδόν.

Αποτελέσματα Μοντέλων

Ερευνήσαμε διάφορα μοντέλα και πειραματιστήκαμε με σκοπό την βελτίωση της επίδοσης τους. Τα μοντέλα μη Μηχανικής Μάθησης ήταν τα εξής: Προανάκτηση Επόμενης Γραμμής(next_line), ο Best Offset Prefetcher(bo) και ο Irregular Stream Buffer(sisb), όπως και ο συνδυασμός των δύο τελευταίων. Το μοντέλο μηχανικής μάθησης που προσομοιώσαμε ήταν το TransFetch με μειωμένο βαθμό προανάκτησης σε 2 όπως και σε όλους τους μηχανισμού και το δικό μας νευρωνικό δίκτυο(LSTM). Τέλος το μοντέλο μας συνδυάστηκε ιεραρχικά και συμπληρωματικά με το μοντέλο Επόμενης Γραμμής(LSTM_next_line), το Best Offset Prefetcher(LSTM_bo) και τον Irregular Stream Buffer(LSTM_sisb).

Για κάθε σουίτα μετροπρογραμμάτων παρουσιάζουμε τα αποτελέσματα όλων των μηχανισμών προανάκτησης.

IPC

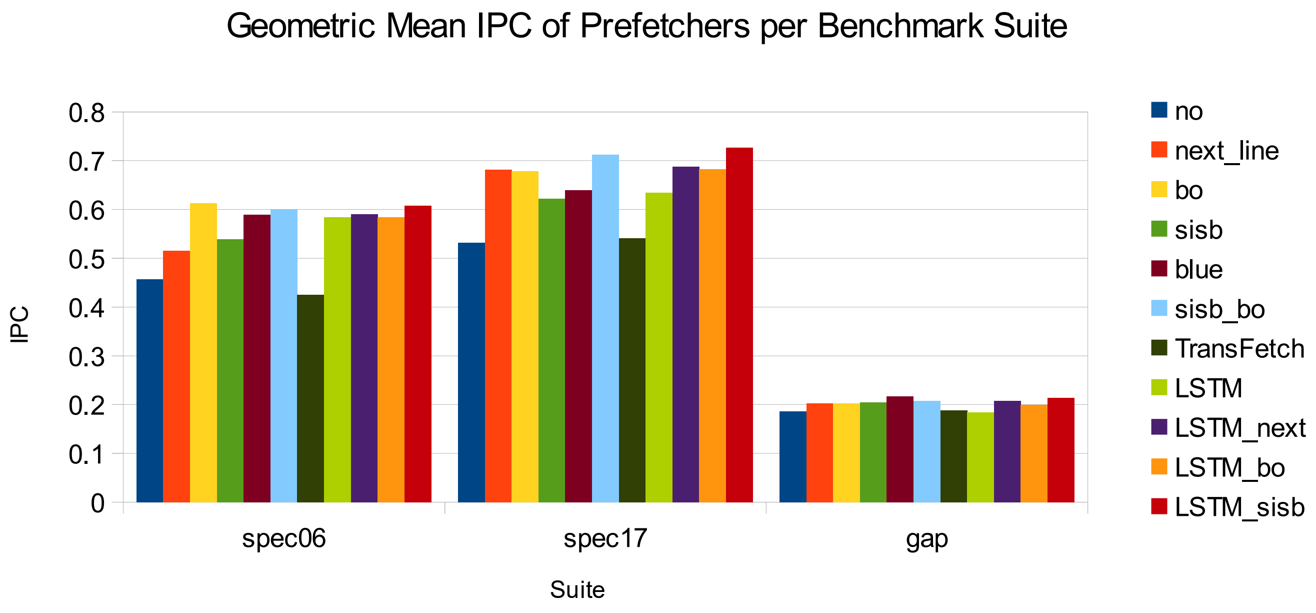


Figure 1.15: Γεωμετρικός Μέσος IPC μοντέλων ανά σουίτα

IPC βελτίωση

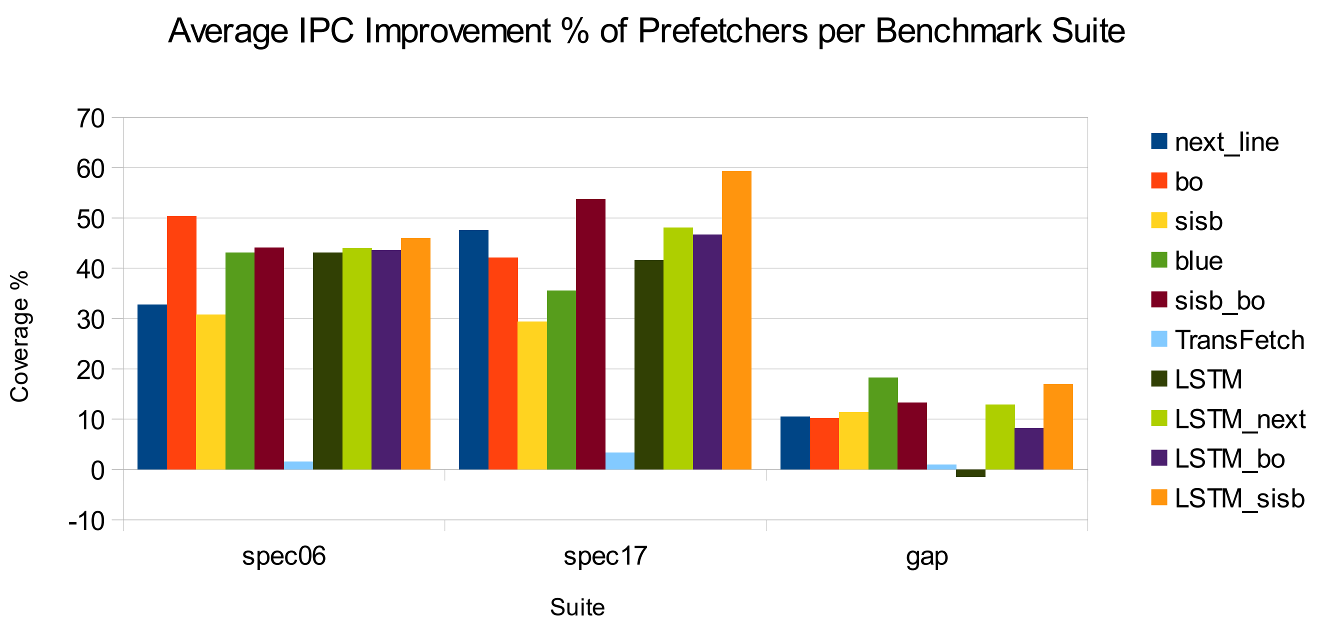


Figure 1.16: Μέσος όρος βελτίωσης IPC μοντέλων ανά σουίτα

Από τα διαγράμματα Figure 1.15, Figure 1.16 παρατηρούμε πως τα καλύτερα μοντέλα είναι ο Best Offset Prefetcher για SPEC06, LSTM_sisb για SPEC17 και blue για το GAP. Όμως το μοντέλο μας σε συνδυασμό με τον ISB έχει υψηλή επίδοση και στις 3 σουίτες καθώς κατατάσσεται δεύτερο στις δύο άλλες σουίτες. Ικανοποιητική επίδοση έχει και ο συνδυασμός ISB με BOP χωρίς να υστερεί πολύ από τον καλύτερο. Παρακάτω εντοπίζουμε τα αποτελέσματα άλλων μετρικών των prefetcher που προέκυψαν από την προσομοίωση.

Ακρίβεια

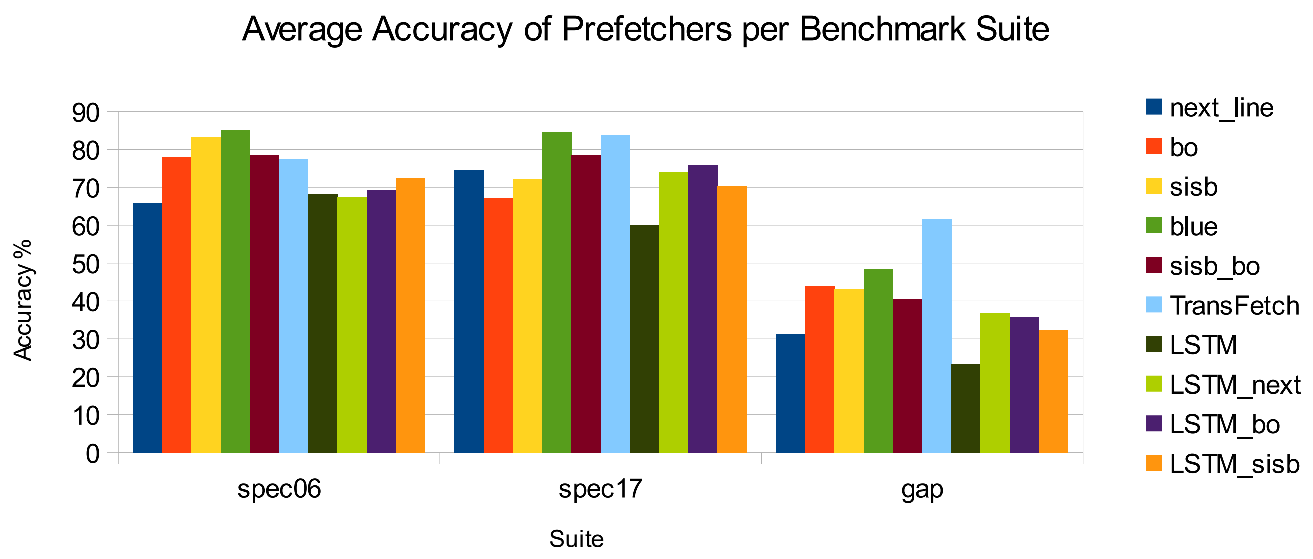


Figure 1.17: Μέσος όρος ακρίβειας % μοντέλων ανά σουίτα

Κάλυψη

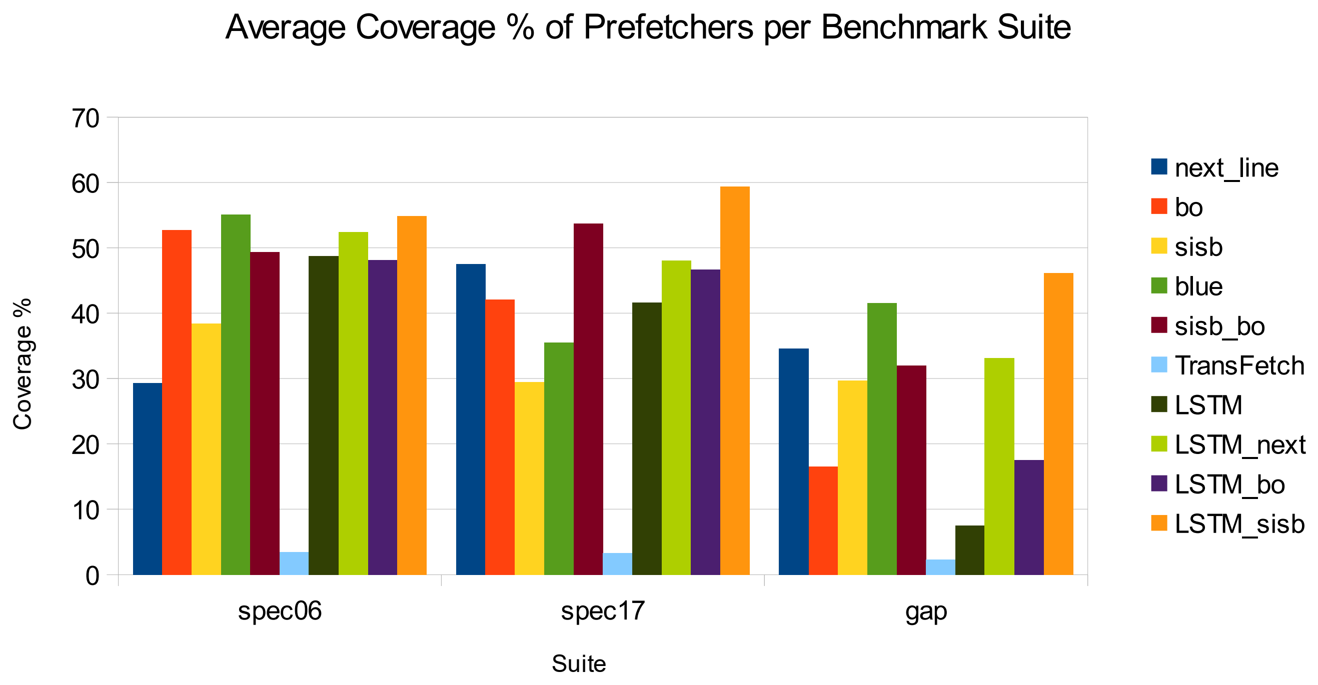


Figure 1.18: Μέσος όρος Κάλυψη % μοντέλων ανά σουίτα

Αστοχίες ανά χιλιάδα εντολών

Οι αστοχίες ανά χιλιάδα εντολών(MPKI) παρουσιάζεται στο Figure 1.19

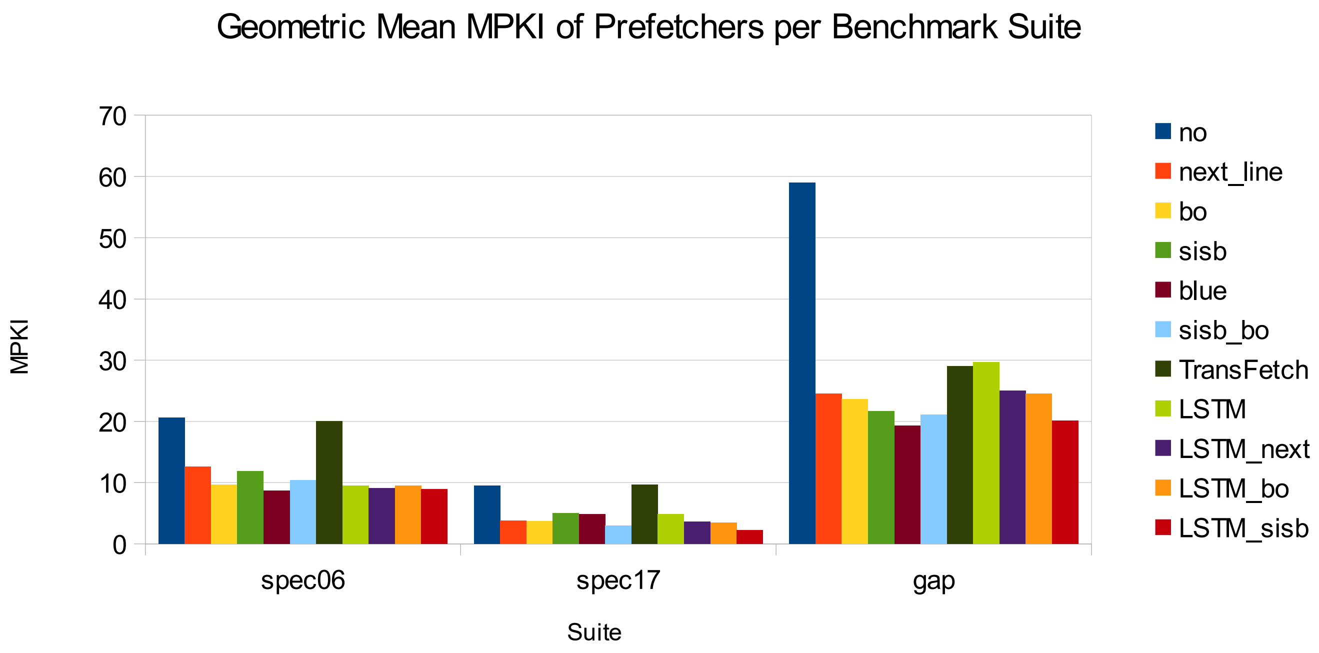


Figure 1.19: Γεωμετρικός Μέσος MPKI μοντέλων ανά σουίτα

1.6 Σύνοψη και μελλοντικές επεκτάσεις

Η χρήση Νευρωνικών Δικτύων σε μηχανισμό προανάκτησης δεδομένων σε προσομοιώσεις και ελεγχόμενο περιβάλλον δείχνει υποσχόμενα αποτελέσματα για την υλοποίηση πραγματικών συστημάτων. Το μοντέλο (LSTM) μας έχει παρόμοια επίδοση με τον Irregular Stream Buffer όταν χρησιμοποιείται μόνο του έχοντας σε κάποια μετροπρογράμματα καλύτερη επίδοση και από τον Best Offset. Η βέλτιστη χρήση του όμως παρουσιάζεται όταν λειτουργεί συμπληρωματικά με άλλους μηχανισμούς, όπως επόμενης γραμμής, BOP, ISB. Μάλιστα το μοντέλο με τον ISB έχει σταθερά από τις καλύτερες επιδόσεις σε όλες τις περιπτώσεις, ενώ σε ένα μεγάλο μέρος έχει το μεγαλύτερο κέρδος σε επίδοση.

Το μοντέλο μπορεί να επεκταθεί και να μελετηθεί περισσότερο προς διαφορετικές υλοποιήσεις σε διαφορετικά περιβάλλοντα προσομοιώσεων. Η μετατροπή του σε μοντέλο σύγχρονης εκπαίδευσης και εκτός προσομοίωσης ιχνών μπορεί να προσφέρει καλύτερα αποτελέσματα έχοντας άμεση απόκριση για τον προσδιορισμό λαθών και χρονισμού καθώς οι συνθήκες κρυφής μνήμης αλλάζουν. Επίσης αυτό το βήμα πιθανόν να οδηγήσει σε πιο ρεαλιστική υλοποίηση πραγματικών μοντέλων.

Η μέθοδος προσδιορισμού του συνόλου δεδομένων εκπαίδευσης έγινε με τυχαίο τρόπο δειγματοληψίας. Μέθοδοι για εκπαίδευση από την αρχή της εκτέλεσης ενός προγράμματος ή ανάλυση για τον τρόπο δειγματοληψίας είναι χρήσιμοι για βελτίωση των τελικών μοντέλων που χρησιμοποιούνται.

Επιπλέον η επεξεργασία των χαρακτηριστικών που αξιοποιεί το μοντέλο μας έχει περιθώρια βελτίωσης κυρίως στον τρόπο αναπαράστασης των διευθύνσεων ή και στην αλλαγή των χαρακτηριστικών με προσθήκη επιπλέον όπως η βασική διεύθυνση ή η σελίδα που εντοπίζεται.

Τέλος, εφαρμογή του μοντέλου σε εικονικές διευθύνσεις μπορεί να προσφέρει ευκολότερη πρόβλεψη, αλλά και καλύτερη επίδοση στην προανάκτηση.

Chapter 2

Introduction

Recently Machine Learning has been extensively developed with the increase in processing power, allowing it to be used in a plethora of applications. The increasing efficiency and speed of Machine Learning Techniques with the ability to identify complex patterns has led to applications in new fields. Advances in computer architecture seem to slow down and not follow Moore's Law, so researchers experiment with new and different techniques to overcome the slowing of advancement. Consequently, there has been progress in the applications of Machine Learning techniques in computer architecture to enhance and support newer and more efficient solutions. Machine Learning has been used in computer architecture to improve the performance of mechanisms. In branch prediction Machine Learning, more specifically a perceptron[3] model, shows one of the better results when compared to other methods. Other works in branch predictors were inspired and improved performance further by aiming hard to predict branches[41] [2] [42] [43].

Machine Learning Techniques are also used in dynamic scheduling [44] and resource management [45] in architecture, usually Reinforcement Learning. Another area of resource management where ML is used is system memory modules, e.g. memory controllers [46] [1], or Input output communication for memory [47].

2.1 Subject of the thesis

The current thesis studies the application of Machine Learning approaches and Neural Networks in computer architecture and more specifically, we focus on the data cache prefetching at the level of the Last-level Cache (LLC) or L3 cache. One of the main obstacles to computer performance is the processor-memory performance gap that leads to idle time in a

CPU waiting for the needed data or instructions from the main memory. Prefetching is a mechanism that limits the above-mentioned problem by predicting and loading data to faster memories earlier than needed.

2.2 Motivation and other work

Traditional prefetching techniques are done with different approaches based on the available resources and use cases. A prefetcher can be as simple as a next-line calculation and prefetch or complex algorithms with increasing needs in processing and memory. The most referenced and used prefetchers are a step prefetcher, correlation methods such as Markov chains[19], Global History Buffer [20], Best Offset prefetcher [27] and others. Each approach has its own benefits and drawbacks in performance, efficiency, and hardware, thus making it unrealistic to have a universal solution for every application.

There have been several Machine Learning models in data prefetching, with varying approaches. The main comparison of the data prefetching problem is the next word prediction problem in Natural Language Processing. Therefore the use of Long Short Term Memory Recurrent Neural Networks in the issue has prevailed treating it as a classification problem [7] [8]. There have been other approaches to the issue that show promising results that are described in the thesis in Section 4.2

2.3 Approach and contributions

The current thesis is based on previous works in the subject such as Learning Memory Access Patterns [8] and Understanding memory access patterns for prefetching [7].

We treated the prefetching problem as a classification problem of memory address deltas. We define Delta as the distance between the current memory address and a future one with a constant distance. The addresses are organized by their Program Counters(PCs) and differentiated only when they are accessed by the same PC. The distance of address deltas is carefully and after an extensive analysis picked to overcome the latency of memory accesses and produce the most useful prefetches, thus improving performance.

Our prefetcher operates in the Last Level Cache(LLC), in a three cache layer system, where only addresses that arrive at the LLC are used for prediction. Our model tries to predict memory addresses even with across page predictions.

Our Machine Learning Model uses the memory address deltas and PCs as inputs feeding them to embeddings and the results to a single Long Short Term Memory [6] model. Finally, the results of the LSTM are processed through a Linear layer where a softmax function is applied to produce probabilities of the results, where the most probable results, if they exceed a fixed probability threshold, are finally predicted.

Our model is trained offline with produced traces of three large benchmark suites such as the SPEC CPU 2006 and 2017 and finally the GAP suite. After training, we produce prefetch traces that are fed to the ChampSim simulator where each application is run again with a prefetcher. The results of the simulator are processed and used for the evaluation of prefetchers, mainly the Instructions Per Cycle metric and statistics of misses and prefetched addresses.

The results show that machine learning models can compete with state-of-the-art prefetchers and with further research surpass them or work complementary to them. There are benchmarks and workloads where our model outperforms Best Offset and ISB prefetcher, whereas on others our model working simultaneously with the next line or Best Offset produces better results. So further work on the subject with different experimentation in the neural network model or the level of memory is recommended.

2.4 Thesis Overview

This thesis is split into chapters. In Chapter 3 we explain the theoretical background of the subject. There is the Section 3.1 where the function, characteristics, and types of prefetchers are explained. In the Section 3.2, we describe how Machine Learning and Neural Networks work and are used for different applications. In Chapter 4 we present related work done on the subject of prefetching with and without Machine Learning. In Chapter 5 there is an analysis of the work done in this thesis and the design choices of the model. In Chapter 6 we describe the setups and tools we used. Additionally, we present the results of our work that guided us in the design and the final comparison of prefetchers with the benchmarks that were described before. Finally, Chapter 7 concludes the work of this thesis with information about future work.

Chapter 3

Background

In this chapter, we explain the theoretical background needed to understand our work. The first section describes the theoretical background needed to understand prefetching in the cache memory. The second section of the chapter is focused on Machine Learning Techniques and the knowledge need for the implemented model.

3.1 Prefetching

3.1.1 Cache and Prefetching

The RAM and CPU performance gap is one of the main bottlenecks of modern computing systems. Computation speeds are getting faster and quicker than memory speeds deteriorating the problem. This difference in speeds causes data starvation of the CPU, while it stalls waiting for data to be fetched. For memory-intensive computations more than half of the CPU cycles can be lost waiting.

Since it is not possible to have memory at the required speeds with enough space, computer architectures have started implementing a hierarchical model of memory. Specifically, there is a layer of memory between RAM and CPU which is a lot smaller than RAM but its speed matches the clock speed of a CPU. This type of memory is called cache memory. Cache Memory has only a small fraction of RAM's capacity it takes advantage of the two types of locality to give the illusion of faster total memory speed.

- **Temporal locality**: When there is a reference of a memory location then it will likely be referenced again in the near future

- **Spatial locality**: When there is a reference of a memory location then nearby memory locations will likely be referenced in the near future

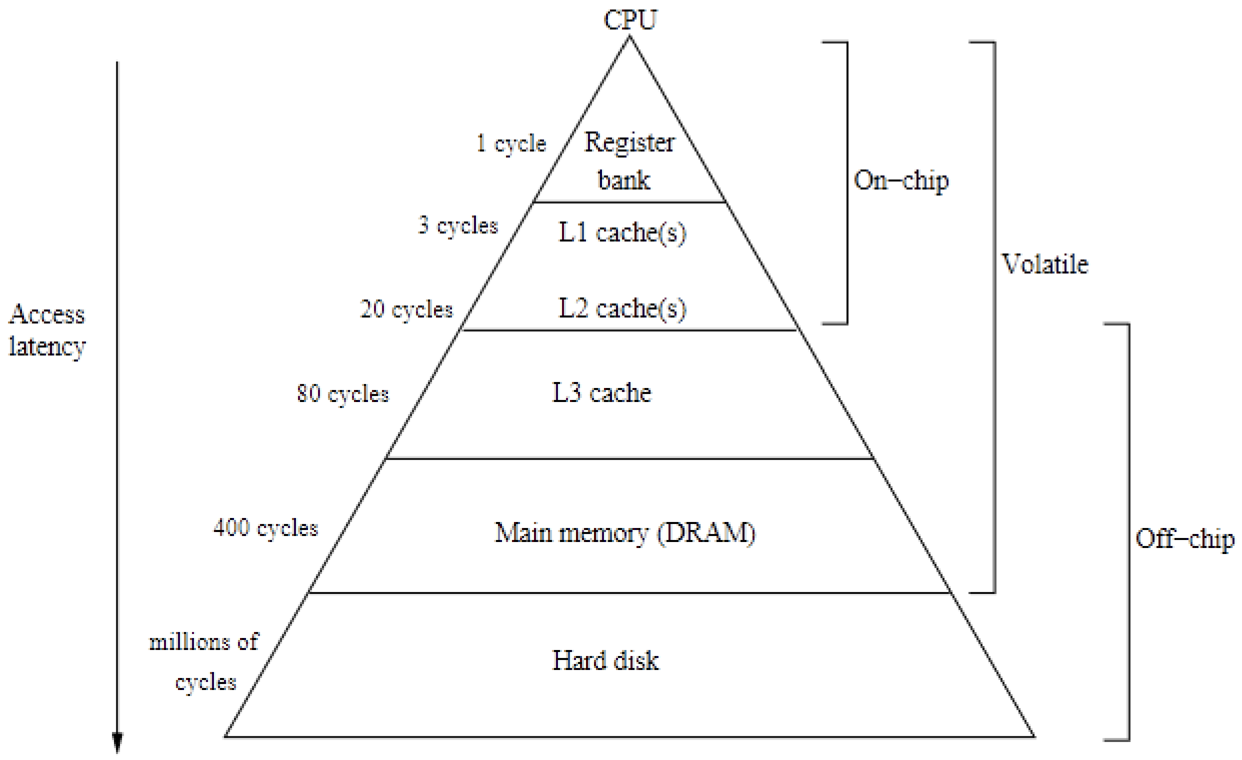


Figure 3.1: Standard Memory Hierarchy [48].

In modern systems, there is more than one type of cache memory split in a hierarchical structure. Each layer of cache has different characteristics of speed, capacity, density, power consumption, and cost. As shown in Figure 3.1, usually there are 3 cache levels with each level away from the processor having higher latency and increasing capacity. The effect of that hierarchy benefits from the locality.

Even though cache reduces memory latency, by delivering some of the data faster to the registers, CPU still stalls when an address is not there. It is very common, especially in memory-intensive workloads with a lot of data. Cache prefetching is a technique used to counter that problem by predicting and fetching memory blocks from slower main memory to cache before they are needed by the CPU to reduce idle time.

3.1.2 Prefetcher Terminology

A prefetcher has some basic characteristics independent of implementation.

- **Degree**: The number of addresses that are being prefetched on a single prefetch command.

- **Distance**: How many instructions in the future from the prediction will the prefetched address be used.

After a prefetcher predicts and fetches an address (makes a prefetch), the prefetch can be categorized based on the result and usefulness. So we have the following types of prefetch:

- **Useful**: The prefetched address in the cache is used and reduces the miss rate of the cache.
- **Useless**: The prefetched address is evicted from the cache before being used.
- **Wrong**: The prefetched address is not used.
- **Harmful**: The prefetch induces a cache miss by replacing useful data.
- **Timely**: The prefetch is completed before the prefetched address is requested by the program.
- **Canceled**: The prefetch is not issued because the address has already been requested by the program.

3.1.3 Metrics for prefetchers

A prefetcher is evaluated by the following metrics:

- **Accuracy**: The percentage of useful prefetches over the sum of total prefetches, i.e., useful and useless prefetches:

  $$Accuracy = \frac{useful\ prefetches}{useful\ prefetches\ +\ useless\ prefetches}$$

- **Coverage**: The percentage of useful prefetches over the total number of misses without prefetcher:

  $$Coverage = \frac{useful\ prefetches}{total\ misses\ without\ prefetcher}$$

- **Timeliness**: How early a block is prefetched to when it is used. Timeliness can be measured by the number of consecutive load instructions between the issue of a prefetch and the usage. Its impact is shown indirectly in the performance

The impact of a prefetcher can also be measured by the Instructions per Cycle(IPC). This metric is influenced by many factors, but when all the other parameters are kept constant the improvement of the IPC is a useful metric. It shows the impactful result of the prefetcher, which is the speedup of an application.

3.1.4 Prefetching characteristics

During studying and designing prefetching techniques some main issues need to be resolved in order to understand the success of the prefetcher: Prefetch data, the timing of prefetches, and memory level.

Prefetch Data

As expected one of the most important factors in successful memory prefetching is predicting which data will be prefetched. Discovering data patterns and extrapolating which addresses will cause cache misses is the main focus of prefetched data to have them already loaded in the cache when needed.

The process of identifying these addresses needs to be accurate to cover the misses and improve performance. On the contrary, less accurate predictions pollute the cache with needless data inducing evictions of useful cache lines. Thus cache misses could hurt the performance of execution. Furthermore, useless data prefetches take up memory bus bandwidth increasing the time needed for completing other useful load operations.

Timing of prefetches

Besides the question of which data will be prefetched, the time in which the prefetch will be issued is critical and greatly impacts the usefulness of a prefetch. It takes time from the moment a prefetch is issued until the data have been loaded into the cache. For a prefetch to be useful the data need to have been loaded into the cache before they are requested, thus the time of a prefetch issue is critical. The necessary time between a prefetch issue and its completion can be calculated as the time of the prefetch calculation plus the duration of data transfer from the memory to the cache. Therefore a prefetch must be issued at least as long as the necessary time before the data is requested to avoid having a late prefetch.

Different methods regarding the timing of a prefetch have been proposed [49] [27]. A common practice is to define the prefetches to be dependent on other events of a program [49]. These events usually are memory accesses, cache misses, branches or jumps, etc. Another approach is to predict based on the PC trying to be some cycle ahead. The previous examples can be used in hardware and software prefetching. Specifically for software prefetching the decision of timing the prefetch issue is on the programmer or compiler side to add a prefetch command when needed. This freedom of choice needs more attention to produce accurate results and synchronize them if other threads or programs calculate the addresses.

Almost any implementation of a prefetcher follows a careful analysis of the timing of the accesses, misses, etc. The data gathered from the analysis are used to profile and determine the time frame of a prefetcher. Some prefetchers use timing as a feature to predict when a miss will happen. In this thesis, we take timeliness into heavy consideration for the designing and implementation of the machine learning model.

Operation Level

A prefetcher can be implemented on different layers in the memory hierarchy. It operates by monitoring a layer and tries to predict data that will be needed from the next farther level. The prevalent operation layers are between the main memory(RAM) and the cache hierarchy, trying to fetch from the storage and RAM accordingly because the order of latency of those mediums varies significantly. This causes the impact of the prefetcher to be the most notable.

The data are fetched to a level closer to the CPU to improve performance. The ordinary use case is at the cache level, so to design a prefetcher the details of the cache need to be considered. The policies of the cache, e.g., write-back policy, along with the layout in a multi-core system influence the impact of a prefetcher differently.

The memory addresses used on different levels of cache do not always represent the physical addresses of the main memory. Prefetchers operating on some levels use the virtual address space[1] for the predictions. We focus on the L3 or Last Level Cache of a system with the physical address stream.

Placement

Cache data prefetchers store the prefetched data mostly in two structures: in the cache itself or in a specific auxiliary data structure called the Prefetch cache, which works as a buffer. This happens because some high-intensity prefetchers or other prefetchers with low accuracy cause evictions and pollution in the main cache. The prefetched data stays in the buffer until the CPU asks them, which causes them to be moved into the main cache. If the predicted data is not used they do not cause eviction of blocks in the main cache, but there is an extra

[1]An address that corresponds to a location in virtual space and is translated by address mapping to a physical address when memory is accessed[50]

Pattern	Recurrence relation $\mathcal{R}$	Example	Notes
Constant	$A_n = A_{n-1}$	*ptr	
Delta	$A_n = A_{n-1} + d$	streaming, array traversal	d = stride
Pointer Delta	$A_n = Ld(A_{n-1})$	next = current->next	Load address from previous
Indirect Delta	$A_n = Ld(B_{n-1} + d)$	*(M[i])	
Indirect Index	$A_n = Ld(B_{n-1+c} + d)$	M[N[i]]	c = base address of M, d=stride

Table 3.1: Classification of Memory Access Patterns [9]

computation cost for data searching in the different cache or data transfer from the caches. For the purposes of this thesis, the prefetches issued are loaded on the cache itself

Prediction of Memory Access Patterns

The analysis of the memory access patterns has an extensive complexity as can be seen from the plethora of research works that contribute different perspectives on the subject [8] [7] [9]. Still, there is no universal solution and a deep understanding of the details of memory prediction, so prefetching has been an active and constantly adapting procedure. Since the adoption of a universal successful model is strenuous and almost impossible when considering the cost of it, prefetch designing is about balancing and finding the optimal result for different applications according to their access pattern.

A great analysis of this aspect is done in Classifying Memory Access Patterns for Prefetching [9]. It uses a recurrence relation $\mathcal{R}$ as $A_n = f(A_{n-1})$, where A is a memory address, n represents the n^{th} execution of a load instruction and $f(x)$ is an arbitrary function. In the Table 3.1 the main patterns are described.

Table 3.1 maps memory accesses to design patterns and data structures. However, they are not particularly useful for automated processing. It is a representative way for human understanding in order to create prefetch kernels that follow each behavior. These patterns can be binned into classes and matched to machine operations for the most common behaviors, as presented in Table 3.2

3.1.5 Main prefetching techniques

There are two types of prefetching based on the type of memory that is being fetched, Data prefetching and Instruction prefetching. Our model performs data prefetching in the LLC.

Prefetchers are classified by the level at which it operates and by the info it utilizes to make

Machine Classification	Corresponding Pattern
Constant	Constant
Add	Delta
Add, Multiply	Complex
Load, Add	Linked List, Indirect Index
Load, Add, Multiply	Multiply

Table 3.2: Machine Operations and Patterns [9]

the prefetch.

3.1.6 Approaches

There are two main approaches in the implementation of prefetchers based on the level of operation.

Software Prefetching

Software prefetching allows a programmer or compiler to make prefetch requests. The requests are implemented by inserting prefetch instructions in the instruction set or through software-configurable registers. Prefetch instructions are non-blocking load-to-cache instructions, that do not interfere with the processor or cause page faults. There are multiple examples of software prefetching [10],[11],[12]

The main idea is that by analyzing the code of a program and its data structures when compiling or programming, the memory access pattern can be predicted and fetched earlier by using compile-time information, thus reducing memory misses and stalls. The timing and scale of the inserted instructions are critical to the performance improvement of a software prefetcher and it varies for different programs. Mistimed prefetches can cause extra cache misses and hurt performance.

Software prefetching can be implemented in many different systems without any modifications to the hardware, but the footprint of the program is increased with the extra instructions.

Hardware Prefetching

Hardware prefetching is implemented by a dedicated hardware mechanism in the processor. It monitors memory access of data or instructions, predicts future accesses, and issues prefetches about them into the cache. The mechanism can differ greatly from model to model based on the requirements, cost, and application.

Sequential Prefetching

One of the first implementations of a hardware prefetcher is step prefetching. It exploits the spatial locality of memory prefetching by trying to load the next addresses$(A+k)$ by adding a constant step or stride (k) to the current address(A). A simple, cost- and performance-efficient approach for regular access patterns. There are prefetchers using adaptable steps for the prefetches [13] or even multiple at the same time trying to identify multiple streams of data [14] [15].

History-based prefetching

History-based prefetching is used in most state-of-the-art prefetchers mainly in hardware implementation. The mechanism used needs additional memory to store the history of memory access along with some metadata or only the history of cache misses depending on each model. It uses the history information to calculate a probable memory address that will be needed in the near future exploiting the temporal locality of memory patterns. Any additional hardware needed for the memory or calculation is added cost and power for the CPU and depends on the platform and application.

There are some approaches inspired by Tagged Geometric history length branch predictors(TAGE) [18] that use global history to make predictions [16, 17]. There are also some prefetchers that use the history to correlate the next accesses using a Markov chain model and the misses follow Markov transitions [19].

In most correlating prefetching techniques, issues with memory space and hardware are arising because ideally, they need a complete history. Besides the increased space there is a need for increased memory bandwidth to calculate from saved history. Hence, in many works, there are memory optimization techniques or limitations in the size of the history that can be saved.

The introduction of the Global History Buffer(GHB) [20] influenced most of the following models in the organization of history tracking in prefetching. It introduces a buffer that keeps only the n most recent in the order that they happened. Along with the buffer, an index table with pointers to the GHB is kept, with the keys being useful information such as PCs or missed addresses.

Another way to reduce the size of history data is by grouping different addresses in sets and keeping only information about those sets. An example of this is the following work [51] where different addresses are represented by smaller tags requiring less memory and smaller history to perform well.

Runahead execution

In multi-core or multi-threaded systems or in cases where a core is idle for a number of cycles, there is a way to take advantage of existing and unused hardware in order to calculate addresses for prefetching and improve performance. Runahead execution unblocks the instruction window blocked by long latency operations allowing an unused entity to execute far ahead in a program.

There are different techniques to implement runahead execution. In [21], a separate engine can run and generate dependence graphs without interfering with the main execution and state of a CPU by dynamically identifying and precomputing the instructions that determine the addresses accessed. Another approach [22] is to use different threads, in a multi-threaded system, to execute future load commands triggering cache misses far enough in advance of the main thread to avoid them. A more recent method is precise runahead execution[23], which uses a processor's unused resources in the issue queue and register file to speculatively execute instructions without needing to release the processor state while preserving dependencies among instructions.

Combined Prefetching

The analysis of software and hardware prefetchers shows that each approach has its own advantages and disadvantages. Hardware prefetchers usually are more adaptable with less overhead and faster response, but they require dedicated hardware mechanisms and can suffer on multiple access stream patterns. On the contrary, software prefetchers work better with different streams, as they add prefetch commands on each one, but they suffer from limited bandwidth and increased size of executed commands.

Due to the different natures of prefetchers, there have been attempts at simultaneous use of software and hardware prefetchers working in collaboration to satisfy the need in different and more demanding environments. It requires a delicate balance of both to function well without harming performance because issues such as cache pollution and limited bandwidth can be amplified.

Combined prefetchers [24] work better and are becoming more popular in multi-threaded systems, as parallel execution of more complex algorithms for prediction models and additional hardware with easier communication and timing between threads is more common.

3.1.7 Challenges in memory prefetching

Besides the previously defined characteristics of a prefetcher, there are further challenges that need to be overcome in designing an optimal prefetcher.

Cache pollution and resource sharing

Based on the operation of a prefetcher, e.g. the intensity of load operations, accuracy, etc, the mistakes or wrongly timed prefetches impact a system differently. The size of cache memory is limited and more load operations cause more evictions of previously loaded and useful data to lower level memory, thus decreasing the performance of the prefetcher. Ergo, the degree of a prefetcher, its accuracy, and its intensity impact the probability of useful data removal from a cache, creating new misses that would not appear otherwise. The introduction of prefetch buffers is a countermeasure to the cache pollution issue. A prefetch buffer isolates the loading operations of a prefetcher from the cache and only transfers the data when it is requested by the program. Hence useless prefetches do not cause unnecessary evictions of useful data. However, a larger number of useless loading operations needs larger bandwidth to complete them.

Efficiency issues

Depending on the implementation level of the cache, the latencies and delays in processing and executing prefetch requests vary, while being difficult to calculate for the timings of a prefetcher. In modern out-of-order processors, prefetching is also speculative when prefetching data that would be used only when following a specific path, causing in some cases more misses.

Implementation factors

Some of the above-mentioned prefetchers are easily implementable with small additional hardware or calculation cost, but they cover only simple memory access patterns and perform poorly in most cases. On the contrary, more advanced prefetchers require either more memory or calculation power to predict the addresses. If a prefetcher requires more memory it will either be additional hardware on-chip, increasing the power and cost, or it will use shared memory contesting for memory resources with other programs or elements of the processor/program using bandwidth for regular communication that the prefetcher needs. The cost of implementation differs on the level the prefetcher operates with the higher level being more costly and difficult.

3.2 Machine Learning and Neural Networks

3.2.1 Machine Learning

Machine Learning is one of the main research fields of Artificial Intelligence(AI). It studies the algorithms that improve automatically through experience using data to achieve a better result in a certain task or process. The increase in computational power has led to an explosion of Machine Learning in almost every field and application, from medicine, computer vision, language processing, word prediction, image classification, and more.

An ML system can be modeled as a parametric function $\mathbf{Y} = \mathbf{f}(\mathbf{X}, \theta)$, where $\mathbf{X}$ is the input as matrix and θ are the parameters of the model. This model is built by the training process on sample data. The training process calculates the function's output using the training data as input and measures the output's performance with some metrics, which it uses to adapt the parameters for better results.

So a Machine Learning system is compromised by the following parts:

- **Model**: the trainable parametric system with learnable parameters
- **Dataset**: the data used for training and evaluation. Training data are used in the training process to calculate the parameters, while test data are in the evaluation of the model's ability to calculate the correct output when used in generalized data not used in training. The quality and size of the dataset are critical to the generalized performance.
- **Performance or loss function**: a metric used to evaluate mathematically the output of a model to check the performance of the system in training and evaluation.
- **Optimization algorithm**: is the algorithm used to minimize a loss function or maximize a function based on the model parameters to achieve a better result for the model.

There are three broad categories of Machine Learning approaches based on the type of dataset used.

Supervised Learning

In Supervised Learning, the dataset includes the input data(features) with their desired outputs. So for a dataset D , the data are organized and include input-output, $D = \{(x_i, y_i)\, i = 1, ..., N\}$ where x_i is the input data, y_i the true output and N the total number of sample data. The goal is for the model to learn how the input X can be mapped to the desired output Y. In this type of ML the loss or performance function $L : Y \times Y \to \mathbb{R}_0^+$ is calculated by comparing the output of the model $\hat{y}_i = f(x_i)$ to the desired output y_i, so the loss is $L(\hat{y}_i, y_i)$.

Supervised Learning can be in turn classified into two categories:

- **Regression**: the Machine Learning approach of processing the features to calculate an output variable that is continuous within a specific range (usually a real numeric value). Examples of regression Supervised Learning are stock or asset price prediction.
- **Classification**: the Machine Learning approach of processing the features to label an output with a limited set of values. The values usually are categories, classes, or integers. When there are two possible values it is called binary classification, e.g. email filtering with spam and no spam emails.

Unsupervised Learning

In Unsupervised Learning systems the goal is to learn patterns and find correlations in the data. In contrast to supervised systems, they require simpler datasets that only include the input data (features), without the desired output. Through this process, the system can create clusters, and hidden groups or reduce the dimensionality of data by creating representations of a larger dataset.

Reinforcement Learning

In Reinforcement Learning the approach is different from the other methods because it does not have fixed datasets for training. It uses observed rewards (or punishment) to learn how an agent takes action in an environment. The system is fed with a set of permissible actions and rules for the interaction and a reward function to provide feedback for the selected actions. Reinforcement Learning focuses on a balance between exploration of an environment and current knowledge of the action. The training of such a system is similar to trial and error because when the system decides on an action it rewards itself when it is closer to the goal or punishes when it has the opposite result. An example of Reinforcement Learning is robot automated driving or a machine that learns to play chess. In both cases, the subject interacts with an environment, area, or chessboard accordingly, and chooses from a set of actions to achieve its goal of staying on a road or winning a chess match.

3.2.2 Neural Networks

Artificial Neural Networks (ANNs) are a significant part of Machine Learning and can be used in different contexts, such as Supervised and Unsupervised. ANNs are mathematical models of processing information based on similar biological systems. The model of Neural Networks (NNs) tries to loosely replicate the brain of humans and animals by copying the structure of biological neurons. The basic theory of Neural Networks started developing almost 80 years ago but had limited use until recently due to the computational complexity. Nowadays the use of Neural Networks has exploded and they are the main class of Machine Learning. They are used in many modern tasks with a lot of success like speech recognition, image analysis, automated trading systems, pattern recognition, and many more.

The Perceptron

A Neural Network, like its biological counterpart, is consisted of multiple neurons, which are connected in different patterns. An artificial neuron or node is the building block of all ANNs and is called a perceptron. On its own, a perceptron can be used for binary classification on linearly separable classes.

The perceptron model consists of two functions. First, the inputs of the perceptron are linearly combined with the inclusion of an external bias. Then a hard limiter, the signum function, is applied to the resulting sum of the first function resulting in an output of 1 or -1. So the inputs are explained mathematically. The input vector $x \in \mathbb{R}^n$ and the internal weights vector $w \in \mathbb{R}^n \forall i \in [1, ...n]$ are multiplied.

Multilayer Perceptron

The perceptron is the building block of more complex neural networks that can be used in a range of applications. Multilayer perceptron (MLP) is a feed-forward neural network that consists of different layers of perceptrons, the input layer, the hidden layers, and the output layer. Initially, the input layer receives the input signal and forwards it to the next layer, without processing or applying any function. The next layers are the hidden layers, where the main processing and computations are executed. Each hidden layer after processing the signal forwards it to the next hidden layer until the last one where it is forwarded to the

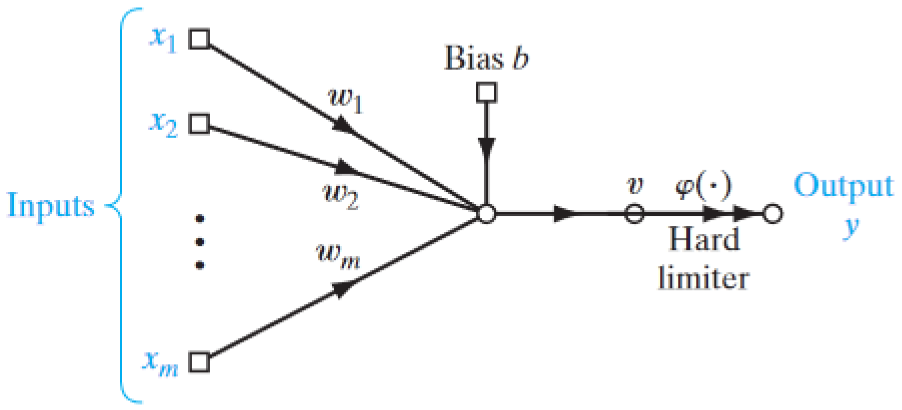


Figure 3.2: Perceptron [25]

output layer. Each hidden layer consists of an uncorrelated number of nodes, receiving the previous layer's output. Finally, the output layer operates on the intermediate representation of the hidden layers and transforms it for the final results depending on the task.

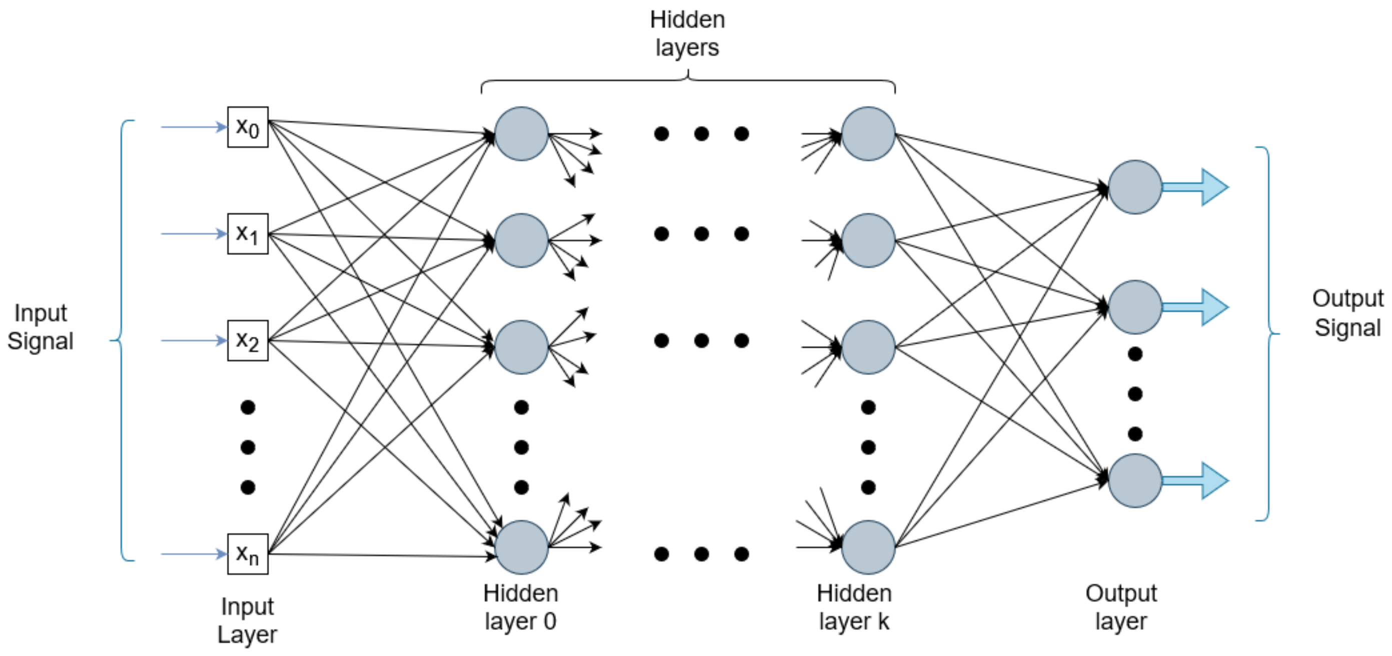


Figure 3.3: Graph of Multilayer Perceptron with k+1 hidden layers

A K+1 layer network with each $M_k, k = 0, 1, \ldots K$ node and input vector $x \in \mathbb{R}^D$. The input layer nodes forward the input to the first hidden layer. Each hidden layer(k) node process its input $z_i^{(k-1)}$ calculated by the previous one using the following equation:

$$a_j^{(k)} = \sum_{i=1}^{M_{k-1}} W_{ji}^{(k)} z_i^{(k-1)} + b_j^{(k)}$$

where $W_{ji}^{(k)}$ are the weights, $b_j^{(k)}$ the biases and $j = 1, 2, \dots M_k$ correspond to the node index in each layer. Then the output a, which are called activations, are given as input to the activation function h to calculate the layers output $z_j^{(k)} = h(a_j^{(k)}$. For the first hidden layer(1) $z_i^{(0)} = x_i$ and $M_0 = D$. The last layer's outputs are the network's final outputs.

Activation Functions

The activation functions are non-linear, partially differentiable functions that are one of the core elements of neural networks. They are applied to calculate the output of each node and finally to the output of a model. Their non-linearity decouples the linear transformations that are applied to each layer, giving the models the ability to solve non-linear problems. Also, the activation functions have a small range that normalizes the activations affecting the convergence of networks.

There are four main activation functions:

- Sigmoid function(σ): This is one of the first used activation functions. The output is in the (0,1) with lower values being transformed close to zero and bigger ones close to one, causing learning to be extremely slow or not happen. The almost negligible changes in output in those regions, with derivative values converging to 0, cause the above issue, which is called the Vanishing Gradient problem and is one of the most common ones that appear in learning. The function is calculated:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

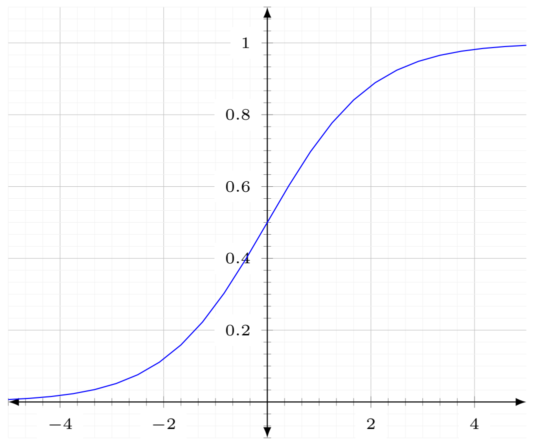


Figure 3.4: Graph of sigmoid function

- Hyperbolic tangent: This activation function limits the output to (-1,1). In comparison to the sigmoid, it keeps negative results strongly negative, differentiating them from positive ones. The derivative of the hyperbolic is steeper, producing more efficient and faster learning, but the Vanishing Gradients problem at the ends of the function persists, similarly to the sigmoid.

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

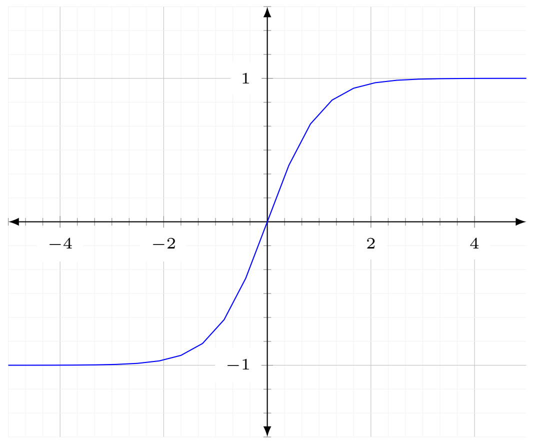


Figure 3.5: Graph of Hyperbolic tangent function

- Rectified Linear Unit - ReLU: ReLU is the most commonly used activation function in neural networks. Values below zero do not impact learning, because the output is 0, and values above zero are the input values. It has a smaller calculation load than the previous functions. The equation of ReLU is the following:

  $$f(x) = max(0, x)$$

  If the zero negative value region that does not impact learning is changed to a linear function with a small scaling value(a = 0.01). This modified version is called Leaky ReLU.

  $$f(x) = \left\{ \begin{array}{ll} ax, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{array} \right\}$$

- Softmax Function: The softmax function is a generalized version of the sigmoid function and is most commonly used when there is a non-binary classification. It regularizes the output values and distributes them to a [0,1] region with a total sum equaling 1 to represent the probability of each class.

  $$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=0}^{N} e^{z_j}}$$

  , where z_i is the i-th output matching the i-th class.

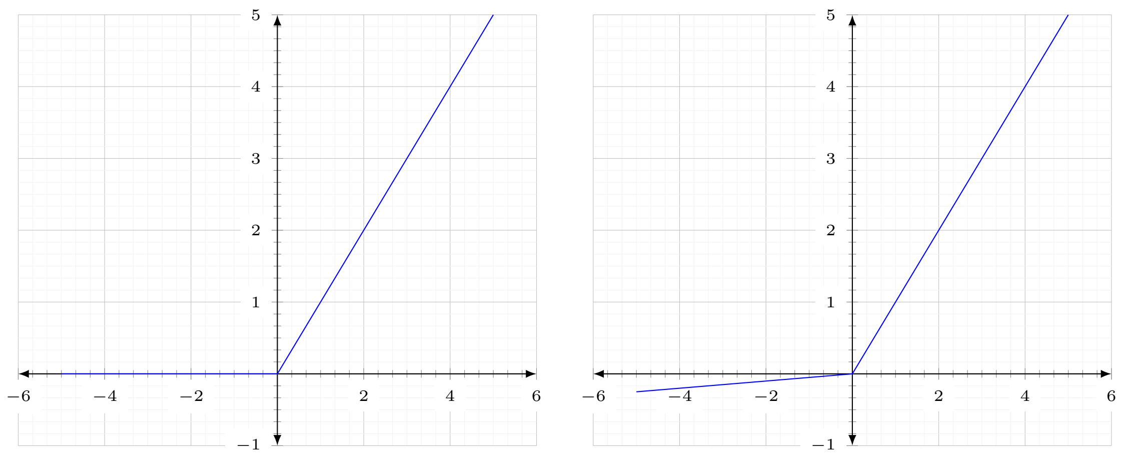


Figure 3.6: ReLU(a) and Leaky ReLU(b)

Loss Function

A loss function is the main metric for the results and performance of a model in a specific task. The loss function represents some cost because mathematically it calculates the distance of the true and predicted values in a suitable space. Depending on the task there are different functions to be used. For regression tasks the most common ones are Mean Absolute Error(MAE) and Mean Squared Error(MSE), which are calculated as follows with $x, y \in \mathbb{R}^N$:

$$MAE(y, x) = \frac{1}{N} \sum_{i=1}^{N} |y_i - x_i|$$

$$MSE(y, x) = \frac{1}{N} \sum_{i=1}^{N} |y_i - x_i|^2$$

In classification tasks the most commonly used is the CrossEntropy Loss function, which is calculated as:

$$L_{CE} = -H(y, p) = -\sum_{i=0}^{N} y_i log(p_{ij})$$

where N is the number of classes, y_i is the predicted value for the observation i and p is the posterior probability of i belonging to j class $p_{ij} = p(y_i = j|x)$.

Training Algorithms

The training process is fundamental in neural networks enabling them to learn and improve themselves by accumulating experience in a limited time. The goal of training is to correct the weights of various nodes of the network based on the output in order to improve performance. The selection of tools, methods, and data is critical for the effectiveness of the final model.

In essence, training is an optimization process, where the model outputs a value, calculating a cost, as described in the loss function subsection, and using it to correct itself and the predicted output. An analytical solution to the problem is extremely difficult and complex so an approximate solution with an iterative process is used. It is split into two parts.

- Firstly, the distribution of total error to each individual node. It represents the way each weight of a node contributes to the total error. This is calculated by the **Backward Propagation** algorithm.
- Secondly, an algorithm that uses the above information to update the weights of each node. This is calculated by **Iterative algorithms** such as gradient descent.

Backward Propagation

Backward Propagation or backpropagation is a method used for calculating the partial derivatives of the error for each of the weights, using the chain rule. To calculate the partial derivative of the cost $C(\widehat{y}, y)$, where $\widehat{y}$ is the network output and y is the ground truth corresponding to the network's output, the following equation is used:

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}}$$

The activation functions are partially differentiable and w_{ij} is the respective weight. As seen above, to calculate the derivative of cost to weight, it is needed to calculate the derivative of the cost with respect to the output of the model, the derivative of the output with respect to the activation and finally the derivative of activation to the subsequent layer weights. The backpropagation algorithm exploits the structure by calculating the derivatives efficiently from the last, output layer to the first moving backward with each step.

Iterative algorithms

The most common algorithm used for updating the weights is gradient descent. It is a first-order optimization algorithm for finding the local minima of a differentiable function. This method is based on the property of a function to decrease the fastest when going in the direction of the negative gradient of the function. So in neural networks, the weight matrices W are updated with the following rule:

$$W = W - \gamma \cdot \nabla_W E(W)$$

where γ is the learning rate, a hyperparameter that controls the pace of the descent, and $\nabla_W E(W)$, the gradient of the error with respect to the network weights calculated by the backpropagation. The total number of iterations in training is either constant or it repeats until there is convergence in a network. The learning rate can be constant in the entire training, but in some cases where the training data are highly dispersed the convergence can be very slow. Therefore, a modification in the above algorithms is commonly used, the Adam(adaptive moment estimation)[52]. It combines gradient descent with momentum and Root Mean Square Propagation(RMSP) [53]. Gradient descent with momentum is an extension of the previous algorithm that allows the search to build inertia in a direction in the search space by considering previous gradients. RMSP is an adaptive optimization algorithm to solve problems of momentum by taking an exponential average. It is given by $w_{t+1} = w_t - (\frac{a_t}{\sqrt{(u_t)}} + e) * \frac{\delta C}{\delta w_t}$, where $u_t = \beta * u_t + (1-\beta) * (\frac{\delta L}{\delta w_t})^2$, w_t, w_{t+1} weights at time t and t+1 respectively and a_t learning rate at time t, C the cost function and β the average parameter. Adam runs averages of gradients and second moments of gradients.

$$m_w^{(t+1)} = \beta_1 m_w^t + (1-\beta_1)\nabla_w C$$

$$u_w^{(t+1)} = \beta_2 u_w^t + (1-\beta_2)(\nabla_w C)^2$$

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-\beta_1^t}$$

$$\hat{u_w} = \frac{u_w^{(t+1)}}{1-\beta_2^t}$$

$$w^{(t+1)} = w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{u}_w} + \epsilon}$$

where ϵ is a small scalar and β_1, β_2 are the forgetting factors for the gradients and second moments of gradients.

The above algorithms can work in two different modes On-Line and Batch Learning. In On-Line Learning every sample of training data updates the weights of the network, in contrast to Batch Learning, where the training data are split into batches, each one used to calculate and update the weights.

3.2.3 Recurrent Neural Networks

A recurrent neural network is a class of deep neural networks for processing sequential data. The sequentiality of an RNN can be temporal or spatial based on the modeling of a problem. They are successfully used in time series prediction[5], machine translation [54], speech synthesis[55], or many other applications.

Unlike other neural network classes, an RNN depends not only on the current input but on other inputs before it. An internal state that works as the network's memory accumulates information from past inputs to influence the current one. Additionally, they have circular connections between higher- and lower-layer neurons and self-feedback connections with their state.

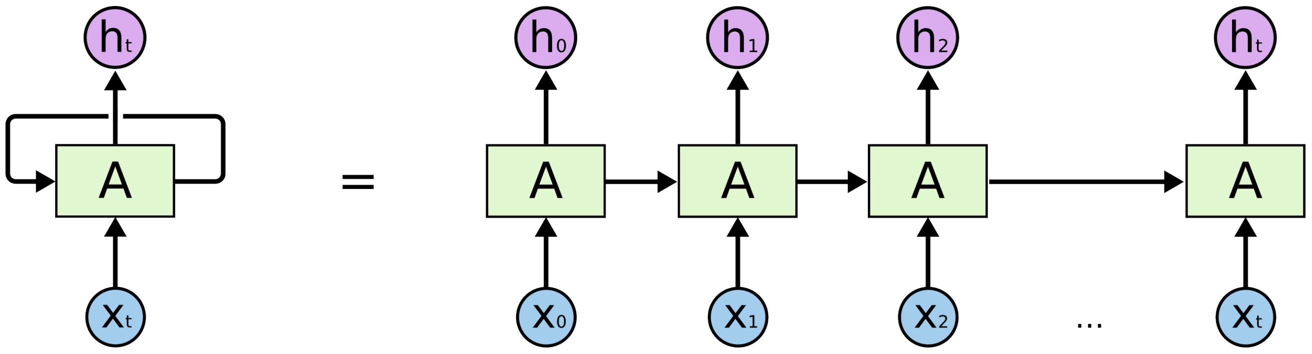


Figure 3.7: Unrolled recurrent neural network [56]

For a time point t the equations that describe an RNN are:

$$h_t = f_h(U_h h_{t-1} + W_h x_t + b_h)$$

$$y_t = f_y(W_y h_t + b_y)$$

,where f_h, f_y are non-linear activation function,h_t,h_{t-1} is the hidden state at time $t, t-1$ respectively, x_t is the input at time t, U_h,W_h,W_y are the weight matrices for the previous time step outputs, for the input and for the current hidden state and finally b_h, b_y are the biases. RNNs can be used with many layers to create deeper networks. An issue that appears with Vanilla RNNs is that they are affected significantly by the most recent samples in a sequence, essentially canceling the property of the infinite receptive field. Furthermore, longer sequences can lead to vanishing gradients, usually in deep architectures, that prevent the network from converging fast or entirely. Two different architectures solve these problems, the Long Short-Term Memory(LSTM) and Gated Recurrent Unit(GRU).

Long Short-Term Memory Neural Network

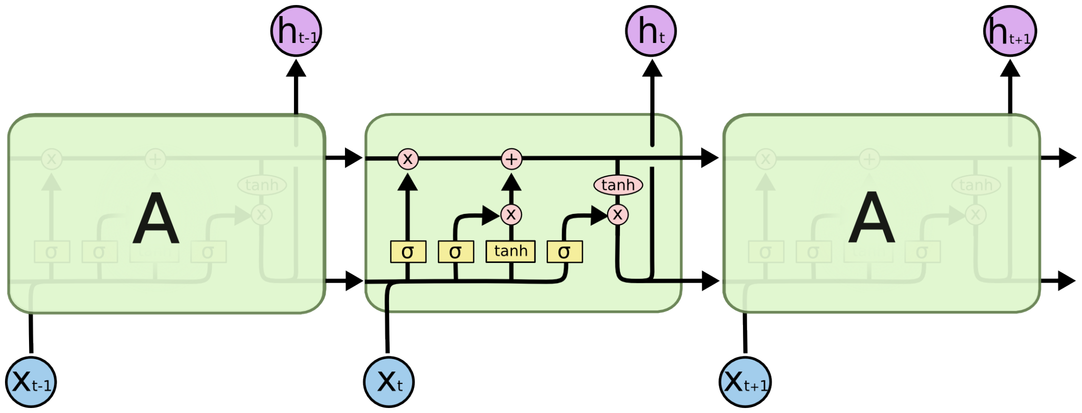
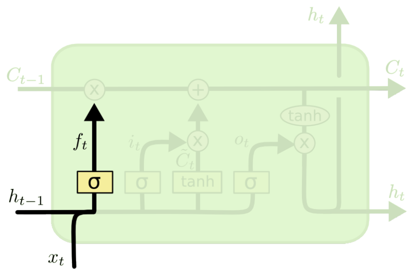


Figure 3.8: LSTM cell interacting with other layers [56]

Short-Term Memory networks were a solution to the Vanishing Gradient Problem of RNNs. LSTMs Figure 3.8 introduce different units in the basic RNN to contain information other than the normal flow of simple RNNs.

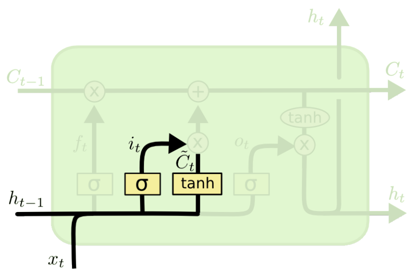
- A Forget gate (Figure 3.9) selects, based on information from the previous hidden state(h_{t-1}) and the input(x_t) by multiplying with a weight matrix(W_f), how the previous cell states(C_{t-1}) impact the result. It outputs a number between 0 and 1 for each cell state value.



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

Figure 3.9: LSTM forget gate[56]

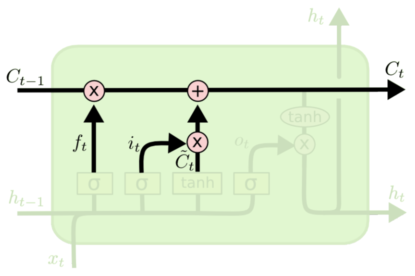
- An Update gate (Figure 3.10) decides what new information is going to be stored in the cell state. It combines two layers, a sigmoid layer(i_t), that decides the values to be updated, and a hyperbolic tangent layer, that creates new candidate values($\tilde{C}t$) to be added to the state.



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

Figure 3.10: LSTM update gate[56]

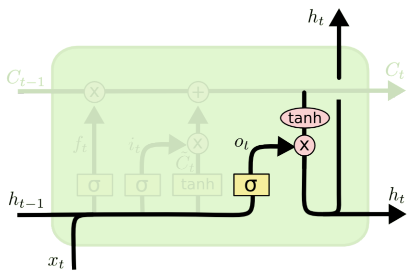
The new cell state(C_t) is calculated by combining the previous layers with the previous cell state (Figure 3.11).



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

Figure 3.11: LSTM cell state[56]

- Finally An Output gate (Figure 3.12) computes the current hidden state(h_t) with two layers. First, the previously calculated cell state(C_t) goes through a hyperbolic tangent layer and second the previous hidden state along with the current input goes through a sigmoid layer.



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

Figure 3.12: LSTM output gate[56]

Biderectional RNN

All Recursive Neural Networks can be modified by connecting two hidden layers of opposite directions to the same output. The output of the network is updated with past and future states to generate the output. The hidden state is a concatenation of the hidden states of each network.

3.2.4 Categorical Representation

A Machine Learning model performing classification needs a way to represent categories or classes as numbers to be fed into models as inputs. A simple way to represent categorical data is to match each unique category to an integer, thus needing as many values as the total number. Most Machine Learning models and especially neural networks work better when the inputs(or outputs) are real continuous values between 0 and 1.

One-hot Encoding

One-hot encoding is a simple solution to that issue. Each value is represented by a vector with size the number of different classes, that only has one value equal to 1 and the rest equal to zero. The index of the value that is equal to 1 is the index of the class that it represents. This approach works fine in cases where the total number of different classes is small, but it is inefficient with large vocabularies where almost all values are zero, while still needing large memory to store the values.

Embeddings

Embeddings are used to represent different words, or different categories, efficiently while encoding information and relations between them. An embedding is a mapping of categorical variables to a vector of continuous real(float) numbers. Instead of mapping the categories statically, embeddings are trainable adjusting their parameters during training similarly to other layers. They are mostly used in Natural Language Processing Tasks[26] because they can express similarities between two different words(or classes) and be represented by floating vectors of low dimensionality.

Embeddings are stored as a $|V| \times D$ matrix, where D is the dimensionality of the embeddings, and $|V|$ is the size of the vocabulary, such a word with index i has the embedding stored in the i-th row of the matrix. In essence, embeddings are a representation of semantics, encoding information relevant to each task

Chapter 4

Related work

The current work is inspired by other previous works and finally, it is compared with some of them

4.1 Non-Machine Learning Prefetchers

Best Offset Prefetcher The Best-Offset Hardware Prefetcher(BOP) [27] is a state-of-the-art hardware prefetcher that was proposed during the International Symposium on High-Performance Computer Architecture 2016. Best-Offset Hardware Prefetcher expands the core idea of offset prefetching by dynamically deciding the offset of a prefetch, whilst considering the factor of timeliness.

The BO Prefetcher is a hardware prefetcher that makes only same-page prefetches and only when a load command is issued. It works as presented in Figure 4.1.

BO-Prefetcher automatically and dynamically tries to adapt to the application behavior over time by setting the prefetch offset(D). Its algorithm tests several offsets to find the best prefetch offset. An offset(d) is considered to be a good one when an access of a memory line (X), there was a recent access for the line X-d. The access ideally should be before a duration that corresponds to the latency of completing a prefetch request. The BOP keeps a record table of the base address of completed prefetch requests, called the recent requests(RR) table. The base address is the address that triggered a prefetch request of the line X + D. It is calculated by the prefetched line when subtracting the prefetch offset D, so the base address is X. This table is used to check if an offset(d) is good by checking to see if the X-d line is in the RR table. The presence of that line means that if offset d was used to prefetch instead

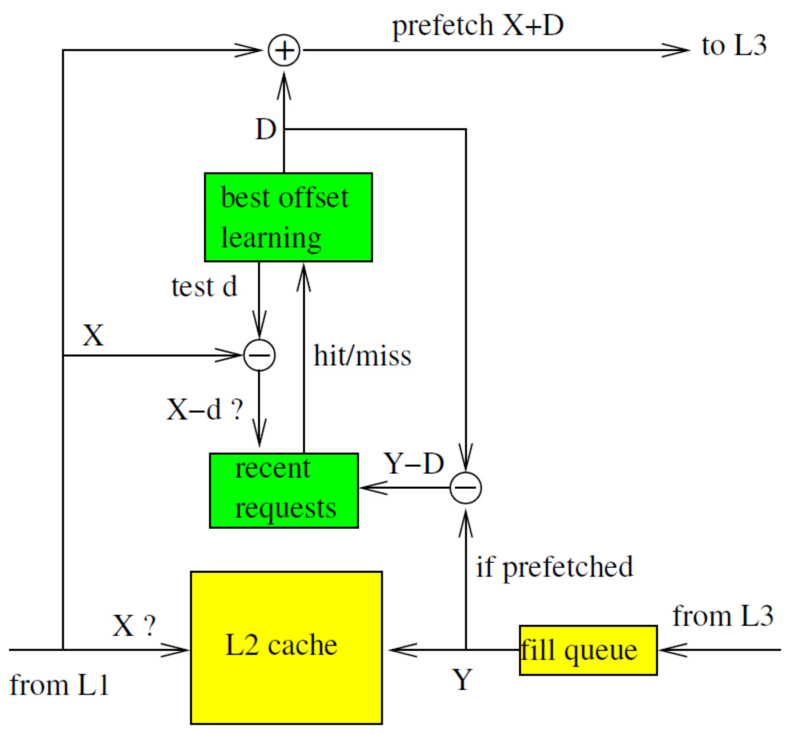


Figure 4.1: Schematic view of a Best-Offset prefetcher

of D the prefetch would have been timely, so the score of d is increased. The RR table is accessed through a simple hash function.

Additionally, the BO prefetcher stores a list of useful positive offsets and a score table with an entry for each listed offset. The BOP works only within the limits of a page so all offsets are limited by a maximum offset that exceeds the limits of a page. Even though offset is limited, the number of different offsets can be large so they need to be sampled in order to save space in the offset table and calculation time. The sampling is done by keeping only offsets between 1 and 256 whose prime factorization does not contain primes greater than 5.

The BO Prefetcher is considered an aggressive prefetching mechanism, that could lead to a significant number of useless prefetches when it encounters highly irregular access patterns, thus wasting processing power and memory bandwidth, which could harm performance. So it has a threshold of the minimum best offset score needed to issue a prefetch using it.

Irregular Stream Buffer

The Irregular Stream Buffer(ISB or sisb)[28] is a prefetcher that targets irregular sequences of temporally correlated memory references. It consists of three units: The training unit, the Address Mapping Caches, and the Stream Predictor. It is shown in Figure 4.2.

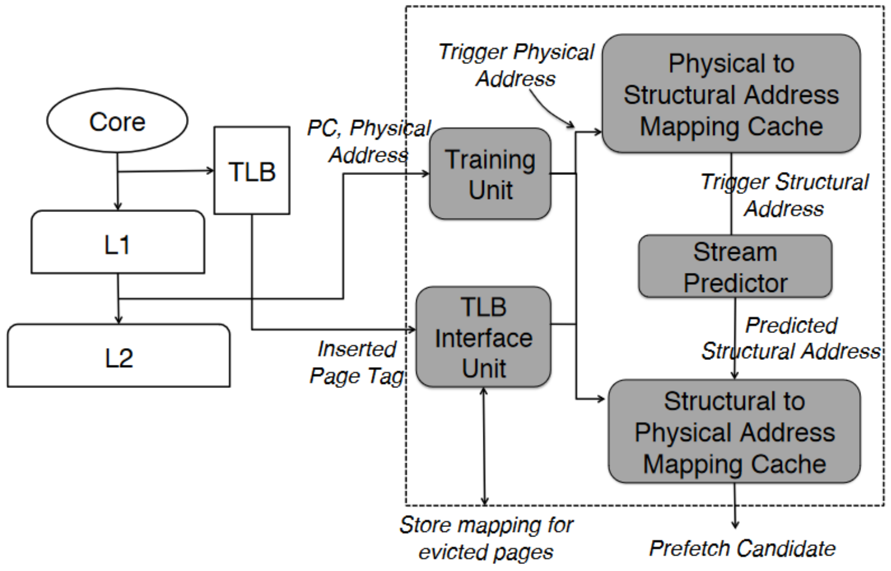

Core
TLB
L1
L2
PC, Physical Address
Training Unit
Inserted Page Tag
TLB Interface Unit
Trigger Physical Address
Physical to Structural Address Mapping Cache
Trigger Structural Address
Stream Predictor
Predicted Structural Address
Structural to Physical Address Mapping Cache
Store mapping for evicted pages
Prefetch Candidate

Figure 4.2: Irregular Stream Buffer

The training unit maintains the last observed address in each PC stream and learns pairs of correlated physical addresses to structural addresses. The Address Mapping Cache uses on-chip caches to store the Physical to Structural Address Mapping(PS-AMC) and the inverse structural to physical(SP-AMC) to enable temporal stream prediction with a single lookup of prefetch candidates. The stream Predictor stores with each entry the starting structural address of a stream along with a counter for the length of the observed stream and another counter for the look-ahead distance.

When an address pair is observed, based on the consecutiveness and previous encounters of those addresses in the PS-AMC, a confidence score is calculated or adjusted to represent the correlation. The ISB is triggered on every cache access to retrieve the structural address from the PS-AMC. Then the Stream Predictor uses the result to predict the next consecutive structural and finally, the SP-AMC translates the structural to physical to enable the prefetch.

Blue

The competition winner in ML-Based Prefetching competition[57] presented in ISCA 2021 is the BL∪E prefetcher [49]. It is a non-ML approach in an ML competition that won based on its performance outperforming other approaches. The main focus of the BLUE prefetcher is timeliness. It is based on the Berti prefetcher[29] with an extension to predict across pages with the entangling prefetcher[30] assisting as a secondary when needed.

Berti collects information about accessed pages, which cache lines were accessed, and the delta

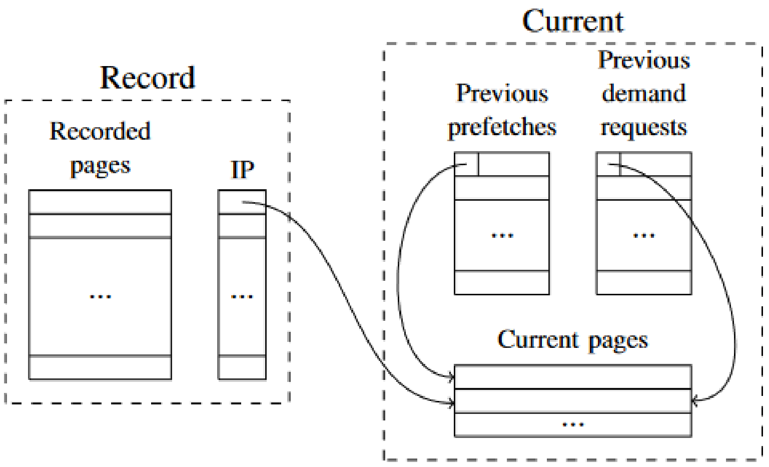


Figure 4.3: Berti prefetcher overview

that provides a timely prefetch for each page. It requires two structures to store previous demand requests and previous prefetch requests. It has two phases, a training phase, and an inference phase.

During training, on each potential miss, it tries to find a time where the prefetch of that address would be successful. When it is resolved the potential latency is calculated by checking when that request was made in case of a miss or in case of a hit it checks with the last prefetch requests. The latency is used to find the offsets and deltas that could serve as successful initiation points for a prefetch. The extension of Berti used in BLUE is the introduction of recording the deltas on a PC table and not per page. After a page is evicted from current pages the highest scored offset is used as the delta. During inference, the prefetcher uses the information kept in the PC table to find the Berti delta and the address offset from the tables. The Berti prefetcher is seen in Figure 4.3. In cases where Berti cannot produce confident results, when pages are poorly accessed, a stride prefetcher is used. Another optimization that is applied in Berti is the introduction of record pages and the prefetched accesses of other pages when Berti produces a result outside the page boundaries. Additionally, the IP-Based entangling data prefetcher is producing prefetches. It correlates the previously accessed cache lines before the latency required to fetch with the current one. So it needs an address to have been accessed previously. BLUE uses the prefetchers mentioned above complementary with one another. First, the entangling prefetcher and if it fails Berti. Finally, if both do not produce two prefetches the next line prefetcher is used,

Next Line

Next Line is one of the simplest and first prefetchers that have been implemented. It simply prefetches the next address from the currently loaded one. It can be extended to prefetch the next two addresses. This way it benefits from the simplest form of spatial locality.

4.2 Machine Learning Prefetchers

Learning Memory Access Patterns

The paper Learning Memory Access Patterns [8] is the core basis of this thesis. The use of Machine Learning in data prefetching is explored with two different neural network models. In both models, the prediction of the address is treated as a classification and not a regression problem. The number of possible classes, meaning memory lines, is vast, thus making it extremely difficult to predict each line if we consider them separate in our vocabulary. Even when using only the cache lines the number is pretty high, so to counter it it tries to predict deltas, $\Delta_N = Addr_{N+1} - Addr_N$ of consecutive addresses.

The first model (Figure 4.4) consists of a single extensive network that manages the prediction for the entirety of a running application. While considering the entirety of an application the vocabulary size of deltas is still very large. Hence only the 50000 most used deltas that appear more than 10 times are considered in this model. The corresponding PC_N and Δ_N are used as an input at each time step N. Each is individually embedded and then concatenated and fed as input to a two-layer LSTM. The LSTM is used for classification and produces the 10 highest-probability deltas for prefetching.

This model does not offer a practical approach to the prefetching problem. The large vocabulary used, is accompanied by increased computational complexity and storage. Additionally, the vocabulary used is limited in order to have an accurate LSTM but completely disregards rarely occurring deltas. This causes the model to have decreased coverage.

The second model (Figure 4.5) utilizes the locality of the address space and uses different models for each region. At first, the address space is clustered using the k-means algorithm [58] into 6 different regions. Afterward, the deltas are computed within each cluster, thus reducing the set of different deltas in the vocabulary when compared to the first model and eliminating the need for an embedding layer in the model.

Each different LSTM modeling a cluster independently has tied weights with the others to reduce the size of the complete model. Consequently, a cluster ID is provided as an additional feature, which gives a different set of biases. Even though the set of different deltas is reduced and can be normalized the model treats it as a classification problem because regression is

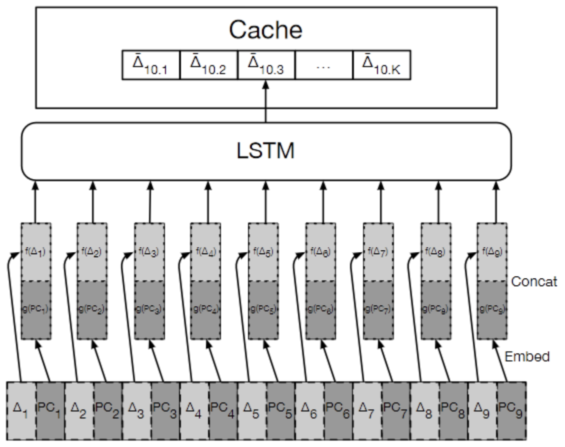


Figure 4.4: Schematic view of the embedding LSTM model

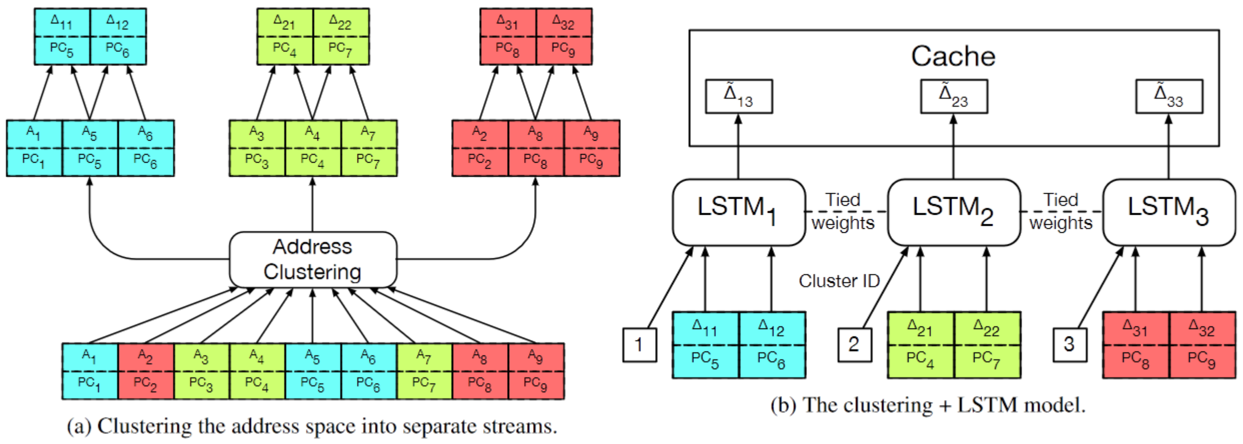

Address Clustering
Cache
LSTM$_1$
LSTM$_2$
LSTM$_3$
Tied weights
Cluster ID
(a) Clustering the address space into separate streams.
(b) The clustering + LSTM model.

Figure 4.5: Schematic view of the clustering LSTM data preprocessing(a) and model(b)

still inaccurate.

Nevertheless, some of the issues of embeddings and the large order of deltas are countered this way. The trade-off is the introduction of the preprocessing step to cluster the address space and the modeling of only the local context since there is no prefetching between different regions of address space.

TransFetch

The TransFetch [31] model is a novel approach to prefetching with Machine Learning Techniques. It tries to solve the problems that appeared in other prefetchers, the class explosion problem, the increased memory demands of tokenization, and finally the increased overhead of ML models that hinder the efficiency of a real-world application of those techniques. TransFetch uses the modern solutions of Transformers and Attention mechanisms[32] while

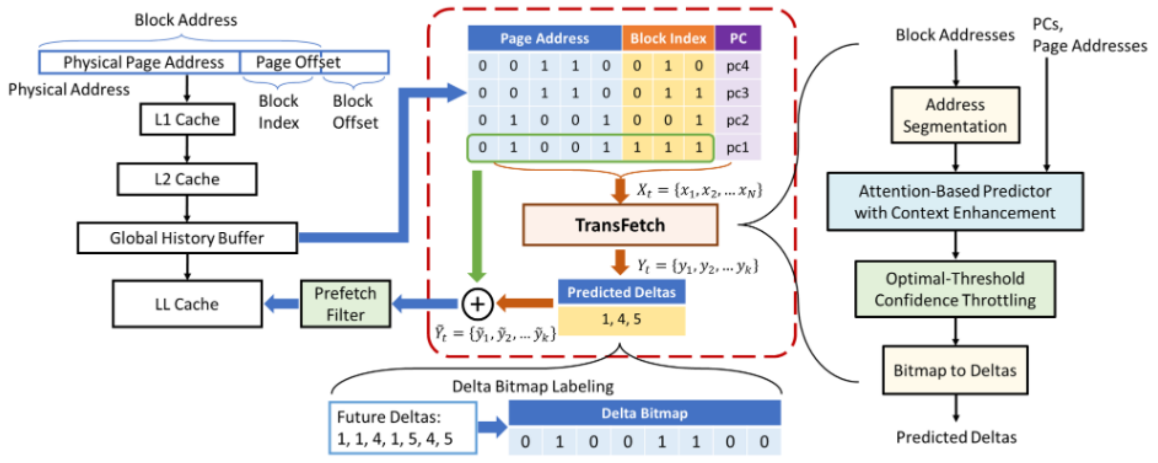


Figure 4.6: Overall Architecture of TransFetch

changing the representation of memory addresses to be more efficient without needing additional space. To solve the class explosion problem it uses address segmentation, where a block address with p-bit page address and c is split to $S = \lceil \frac{p+c}{s} \rceil$ segments. This results in an address being sufficiently represented as an integer vector in dimension S. The TransFetch has variable-degree prefetching by using delta bitmap labeling to cover inter-page patterns and an adaptation in the probability of outputs of the neural model to find an optimal threshold for throttling inference. The main model of TransFetch uses a history of the segmented addresses as an input, which after flattening it they are fed through an embedding layer. After adding position embeddings the resulting vectors are fed into a Transformer layer. Moreover hashed PC values and page distances are used as context for the Transformer.

Voyager Voyager [33] is a novel machine learning approach in data prefetching. It introduces a hierarchical neural structure that tries to learn temporal correlations among addresses. The model as seen in Figure 4.7 is described below. The inputs of the prefetcher are sequences of PCs and memory addresses, which are split into page addresses and offsets. Each of the previous values is fed to the first layer of embeddings, where the outputs of the address ones are used in another embedding layer to produce offset representations that are page-aware. The results of the PC, page and page-aware offset embeddings are combined and fed to two separate LSTM models to predict the output page and offset embeddings. Finally, the outputs are processed through a Linear layer with a softmax function to produce a probabilistic result of the predicted pages and offsets. The most probable pair is chosen for prefetching. The training process of the model is done online and continuously as the program runs.

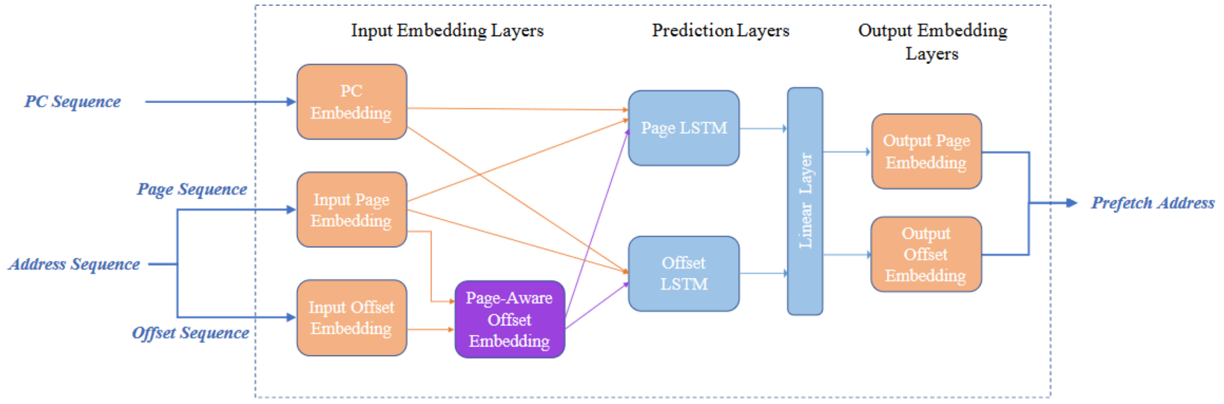


Figure 4.7: Voyager's Model Architecture

Pythia

Pythia[34] is a different Machine Learning approach to memory prefetching, because of the use of Reinforcement Learning for prediction, while giving the ability to customize the features and information it uses for predictions based on each system.

A Reinforcement Learning problem needs to formulate the state spate the actions and the reward scheme.

The state is a k-dimensional vector of features based on two components, the program flow, and the data flow. The control flow has information like the Program Counter of load instructions or branch instructions before a load instruction, with a denotation of the origin of the information from past or current requests. The data flow component is made up of information like cache line address, physical page number, page offset, and cache line delta and history. From all the information Pythia can learn any number of features, however, k features are selected given the limited storage.

The mathematical formulation of the state is the following

$$S \equiv \{\phi_S^1, \phi_S^2, \dots, \phi_S^k\}$$

The action that is selected is the prefetch offset from a set of candidate offsets within the limits of a page. The reward structure defines five levels of rewards based on the result of the prefetch. The five categories are accurate and timely, accurate but late, loss of coverage(when the address corresponds to a different physical page), inaccurate, and no-prefetch. These reward values are easily customized and adjusted based on the profile of the workloads, but there is also provided an automated method for configuration.

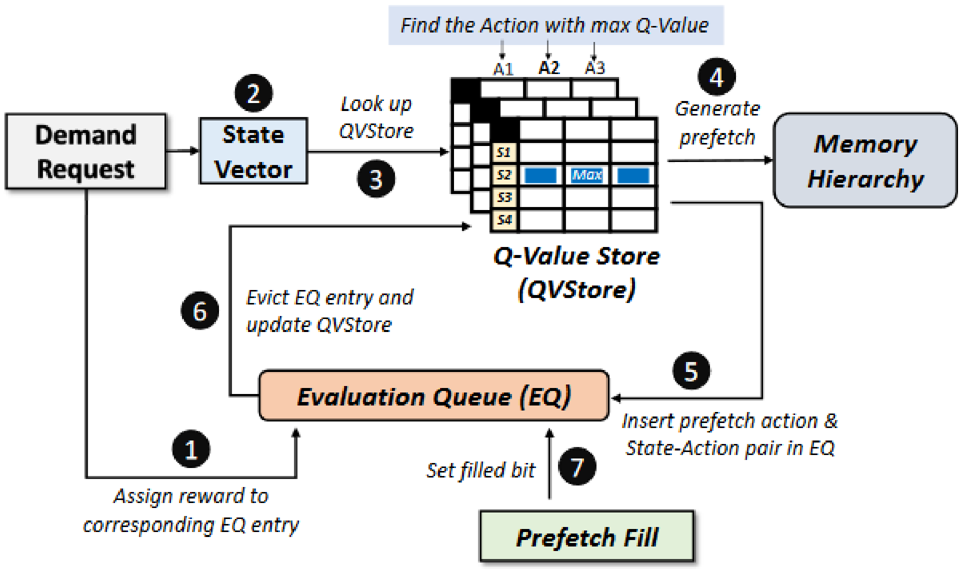


Figure 4.8: Overview of Pythia

The algorithm of Pythia for prefetching follows the Reinforcement learning algorithms of Q-learning. The data structures it requires are two the Q-Value Store, for the state-action pairs and the Evaluation Queue(EQ) to maintain the first-in-first-out list of recently-taken actions information about taken action, prefetch address, and filled into the cache. For a new demand request, Pythia checks the EQ for the address to reward the prefetch action for the EQ entry if it exists. Afterward, it searches the QVStore to find the action with the highest Q-value for the current request state and generates the prefetch request for the selected action, which it saves in the EQ. Additionally, it rewards the action accordingly if it is a no-prefetch or loss of coverage. Finally for every prefetch fill in the cache Pythia sets the EQ filled bit to indicate the presence in the cache for timely of late classification.

Chapter 5

Analysis, Design, and Implementation

In this chapter, the analysis, the design choices, and the implementation of the proposed prefetching mechanism are described. Additionally, the chapter includes an analysis of the experimentation process, frameworks, and tools that were used.

5.1 Oracle Prefetching

In order to find a better distance for calculating and predicting deltas, we implemented an oracle prefetcher. The oracle prefetcher has a priori knowledge of the memory address that will be needed by the application, so it issues prefetch of always correct addresses. The purpose of the oracle prefetcher is to experiment with the prefetching distance and find the maximum possible performance gain of an ideal prefetcher. Through this analysis, we discovered the optimal distance for prefetching. The results are shown in Section 6.1.3. During the implementation, we found that a prefetching model would be better implemented by calculating the distance of the same Program Counters for prefetching. Thus, we ran an oracle prefetcher using a distance per Program Counter. As seen in the results the difference in performance between the oracle prefetchers at an optimal distance is very similar.

5.1.1 Number of Deltas

For every benchmark, there has been a statistical analysis of the total number of different deltas and the coverage of the most used deltas.

The results show that there is a large number of different deltas that are not easy to be predicted with a neural network model. We explored the idea of calculating the deltas between the same Program Counter load addresses. The results as shown below produce a reduction in the number of different deltas. Additionally, the coverage of the most used data is higher, therefore a prefetcher using those deltas can increase the performance.

	Global Delta Calculation			Per PC Delta Calculation		
Trace	Step	No of Deltas	Filter Coverage %	Step	No of Deltas	Filter Coverage %
410.bwaves-s0	22	1091140	89.82	3	5891	100
410.bwaves-s1	5	1288825	86.08	5	14841	100
429.mcf-s0	5	20340108	19.92	10	2385663	40.59
429.mcf-s1	29	30246003	21.25	15	2601155	48.78
433.milc-s0	26	2428251	72.48	6	36003	99.94
433.milc-s1	20	2701919	69.87	4	17422	100
437.leslie3d-s0	30	548790	82.98	13	90749	98.91
437.leslie3d-s1	30	436863	85.85	13	71364	99.39
450.soplex-s1	10	1268286	61.59	9	348188	89.3
459.GemsFDTD-s0	5	2513963	39.5	6	236930	97.83
459.GemsFDTD-s1	5	1882615	59.01	3	192071	96.01
462.libquantum-s0	10	134	100	15	99	100
462.libquantum-s1	10	99	100	9	60	100
470.lbm-s0	10	3032	100	2	7061	100
470.lbm-s1	10	3056	100	2	8911	100
471.omnetpp-s0	10	105854	93.81	8	25203	100
471.omnetpp-s1	10	5190010	29.09	3	2054069	58.69
473.astar-s0	11	570438	22.52	3	356358	52.38
473.astar-s1	5	577625	79.27	15	1549236	54.15
482.sphinx3-s0	10	302506	90.5	5	78868	98.25
482.sphinx3-s1	10	227673	94.09	4	57836	99.27
AVERAGE	**13.48**	**3415580.48**	**71.32**	**7.29**	**482760.86**	**87.31**

Table 5.1: Statistics of Number of Deltas and most common Coverage for SPEC06

SPEC06 has a large variation on the best step and there are a lot of different deltas as seen in Table 5.1. In almost all cases per PC delta analysis shows a significant reduction in the number of deltas while providing higher coverage. There are the 470.lbm benchmarks where the number of deltas is increased but not in an unmanageable number.

Trace	Step	No of Deltas	Coverage of Most Common Deltas
602.gcc-s0	2	123596	98.95
602.gcc-s1	5	49171	99.89
602.gcc-s2	1	255947	64.35
602.gcc-s3	3	34758	99.89
605.mcf-s0	5	14894767	28.01
605.mcf-s1	19	5196079	64.75
605.mcf-s2	11	1133246	98.52
605.mcf-s3	3	10633205	41.45
605.mcf-s4	7	1519034	93.92
605.mcf-s5	4	2037871	80.99
605.mcf-s6	4	5310885	31.66
605.mcf-s7	8	38335990	21.37
605.mcf-s8	7	9305141	12.46
607.cactuBSSN-s0	1	38989	99.81
607.cactuBSSN-s1	1	61576	98.32
607.cactuBSSN-s2	1	61940	98.34
607.cactuBSSN-s3	1	197	100
619.lbm-s0	2	2122	100
619.lbm-s1	4	7270	100
619.lbm-s2	3	5153	100
619.lbm-s3	4	7757	100
620.omnetpp-s0	2	2962202	34.43
620.omnetpp-s1	2	2586948	36.48
621.wrf-s0	5	19376	100
621.wrf-s1	10	238587	88.77
621.wrf-s2	10	245664	86.93
621.wrf-s3	10	38063	99.02
623.xalancbmk-s0	1	45182	99.87
623.xalancbmk-s1	1	10432	100
623.xalancbmk-s2	2	9112	100
623.xalancbmk-s3	3	116368	61.51
623.xalancbmk-s4	2	138131	90.36
623.xalancbmk-s5	3	153460	82.44
649.fotonik3d-s0	3	4174621	51.75
649.fotonik3d-s1	3	25990	100
654.roms-s0	5	95708	93.15
654.roms-s1	3	9354	100
654.roms-s2	4	364556	63.43
654.roms-s3	10	418753	74.14
654.roms-s4	5	22924	100
654.roms-s6	3	76171	98.41
654.roms-s7	6	354266	68.49
654.roms-s8	5	805	100
AVERAGE	**4.51**	**2351659.70**	**80.51**

Table 5.2: Statistics of Number of Deltas per PC and most common Coverage for SPEC17

SPEC17(**??**) has a much lower step for a better result, but the average number of different deltas is a lot higher than SPEC06. Keeping the most common deltas though still covers a similar percentage of the memory accesses. In 605.mcf and 620.omnetpp the extremely high number of deltas is not covered as well as all the other cases by our filtered deltas.

	Global Delta Calculation			Per PC Delta Calculation		
Trace	Step	No of Deltas	Filter Coverage %	Step	No of Deltas	Filter Coverage %
bc-0	32	3335945	65.34	9	544492	90.89
bc-12	35	3476554	60.65	7	606000	90.8
bc-3	22	3617740	66.27	11	737551	86.88
bc-5	33	3458007	66.91	17	780824	85.59
bfs-10	26	1273586	65.94	3	29343	100
bfs-14	1	752767	65.03	3	96196	98.71
bfs-3	32	1643839	67.68	7	166344	97.72
bfs-8	24	1332440	64	3	111429	98.43
cc-13	31	3322505	49.89	9	796832	75.95
cc-14	32	3073755	54.39	9	651927	79.77
cc-5	31	2905538	51.96	10	572540	77.18
cc-6	31	3295838	51.12	9	626657	77.06
pr-10	2	6156999	65.47	6	2651831	64.7
pr-3	11	8373962	47.73	6	2319806	66.97
pr-5	1	5626730	74.16	6	2037387	69.27
sssp-10	29	3171496	67.81	10	247025	95.57
sssp-14	29	3132051	67.72	10	269544	95.3
sssp-3	30	3258026	65.59	9	288783	94.57
sssp-5	29	3140000	67.97	9	229355	96.04
Average	**24.26**	**3386725.16**	**62.40**	**8.05**	**724414**	**86.39**

Table 5.3: Statistics of Number of Deltas and most common Coverage for GAP

GAP(**??**) has a larger delta vocabulary than all the other suites and the prediction step is higher in both Global and per PC. We see a large difference between global and per PC deltas which provide an advantage.

5.2 Prefetching Methods

Our solution is influenced mainly by the work of the papers Learning Memory Access Patterns [8] and Understanding Memory Access Patterns [7].

We treated the prefetching problem as a time series prediction problem. Therefore, we used a sequence of N recent accesses to predict the next useful address. The features that are used are the memory addresses along with the Program Counter(PC) of the load commands. The matching of PC and memory addresses is used to identify different data flows with similar patterns in accesses. In the current work, the prefetcher is operating at the physical address space and tries to predict addresses across pages. This creates the issue of an overwhelming number and complex patterns of memory accesses.

To counter this problem we can use the deltas of consecutive memory accesses, but this produces a significant number of useless prefetches that hinder the improvement of our prefetcher. These prefetches are not completed when they are needed and used by the application. So in order to increase the useful prefetches, we propose further predictions into the future. The deltas between memory accesses that are k load instructions away are calculated and used for prediction.

$\Delta_N = Addr_{N+k} - Addr_N$

We considered a different implementation of the prefetcher and tested it but was not very efficient for prefetching. The tested implementation had a different approach, instead of predicting deltas and the entire addresses together, we predicted each page with the corresponding page offset with separate mechanisms. The problem with this was the combination of both the mechanisms produced a multiplying error in the prefetches that resulted in inaccurate predictions.

Even though the number of different deltas is reduced to a more manageable level it is still not adequate for accurate prefetching. Hence, we organized the addresses by the Program Counter, because the delta patterns are more normalized when we consider each PC's address stream separately. This way each consecutive delta(between the same PC) represents a longer cycle duration in the execution, avoiding late prefetches. Even when further predictions are needed the different deltas are reduced in number as seen in the tables. The calculated deltas consider the results in the entirety of the memory addresses and across-page.

Some deltas that are not used repeatedly are filtered out from the predictions. Only deltas that appear more than 10 times are used and they are capped at 30000 different ones. Even with the above filter, we have a high percentage coverage of predicted addresses.

5.3 Implementation of the Prediction Model

5.3.1 Pytorch

PyTorch [35] is a machine learning library for Python that supports coding models, debugging them while using a Pythonic style and being efficient on hardware accelerators such as GPUs. PyTorch allows very different models to be developed by composing many individual layers and allowing the user to have control over the process. One of the benefits of PyTorch is the interoperability with other external libraries such as NumPy [59]. Additionally, the usage of PyTorch tensors provides a manageable and extensible interface for the programmer to control the data loading, data flow and transportation to different devices and memories while supporting the widely used CUDA for GPU acceleration.

5.3.2 Model Implementation

The prefetching problem as described above is very similar to the next word prediction problem [4] in Natural Language Processing (NLP). In the next word prediction problem given a sequence of words the model tries to predict the next one, similarly in the prefetching problem using a number of previously accessed addresses the model predicts a future value. In both examples, there are complex sequences and dependencies in recent history.

In NLP the use of neural networks and more specifically deep neural networks is extensive. One of the most commonly used is an LSTM network with an embedding layer to handle large vocabularies. As described the different number of deltas is restricted to 30000 of the most frequent in order to achieve satisfying accuracy in the LSTM model. In this case, the input is sequences of Program Counter and delta categories, which are individually embedded and concatenated. This result is fed to the LSTM to perform classification over the delta vocabulary. The output of the LSTM is fed through a Linear layer with an output dimension equaling the vocabulary size. The model produces at most the 2 most probable results if the probability exceeds a certain threshold. Each layer, besides the input and output of the network which is dependent on the vocabulary size, has a fixed size. The sequence size (loopback) is defined through experimentation and previously accepted work, that impacts the accuracy and performance of the network. The model is seen in Figure 5.1

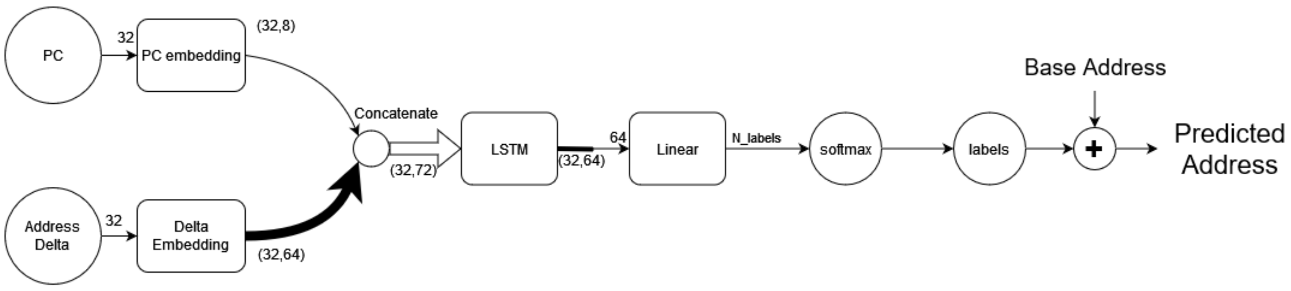


Figure 5.1: Complete Neural Model for Prefetching

5.3.3 Training the model

For the LSTM training, a stateless LSTM is preferred, because the differentiating profile during each benchmark needs to be more loosely influenced. Also, the use of a stateless LSTM provided the opportunity to use sampled data from the entire benchmark runs to represent the different behavior. This way the training dataset is smaller than in other cases, where training needs a significant portion during the beginning of training. The drawback is that some a priori knowledge about the areas of each benchmark that represent the entire run is needed and used. The sampling size used is about 6.5% of each run with a max size of at most 90000 samples in very large benchmarks. Because of the stateless nature of the LSTMs, the output of the network depends on the batch size used during training. Larger batches increase performance and influence the accuracy using larger interference from recent history. This is because stateless LSTMs only keep the cell state values within each batch. With the provided hardware limits the best value for batch size, through experimentation and limitation, was decided to be 64. As described before the ADAM optimizer was used for the training with initial learning rate=0.001.

5.3.4 Model Parameters

The parameters of the neural model were carefully selected after experimentation in model performance as shown in Section 5.3.5

Parameter	Value
Delta Embedding size	64
PC Embedding size	8
LSTM Layers	2
LSTM Size	64
Sequence Length	32
Linear Size	30000

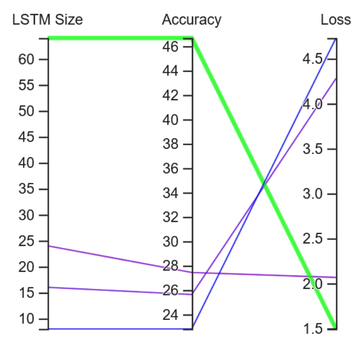
Table 5.4: Model Parameters

5.3.5 Training and Hyperparameter Results

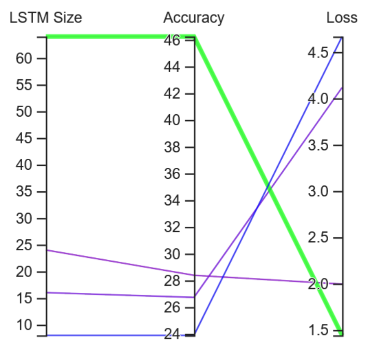
For the impact of model parameters, we trained the model with different combinations of parameters on some selected benchmarks of SPEC06 to see the impact on accuracy. The parameters we experimented with were the size of the LSTM model, the sequence length of the inputs, and the embedding size of the deltas.

LSTM Size

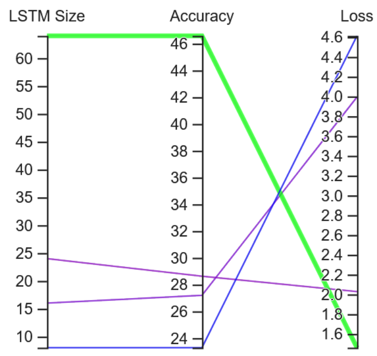
To find the best LSTM Size we trained and validated the accuracy of the model by running different LSTM Sizes with constant Sequence Sizes. The results are described in Figure 5.2



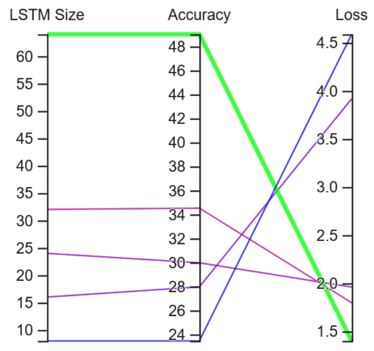
(a) LSTM Size with Sequence Size 8



(b) LSTM Size with Sequence Size 16



(c) LSTM Size with Sequence Size 24



(d) LSTM Size with Sequence Size 32

Figure 5.2: Exploring LSTM Size on different sequence Sizes

The figures Figure 5.2 were produced with the TensorBoard [60]. The first column shows the parameter values, the second shows the Accuracy of the model, and the final the loss of the model. Each line shows the average result of that parameter, with the green one being the best. From those results, we decided on an LSTM Size of 64 that produces higher accuracy on the model.

Sequence Size

In order to find the Sequence Size, we ran the model again with constant LSTM Size and differentiating the sequence size. The results are shown in Figure 5.3

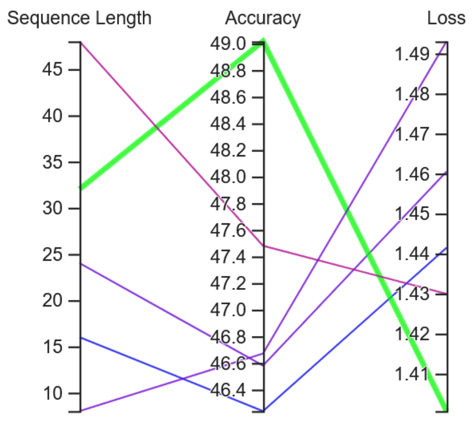


Figure 5.3: Exploring Sequence Size with LSTM Size 64

Embedding Size

Embedding Size was chosen at size 64 based on problems with similar vocabulary sizes in Natural Language Processing. For the embedding of Program Counters Because of their reduced size it was chosen to be 8.

Chapter 6

Evaluation

In this chapter, the metrics and the results of the current work are presented and evaluated in comparison to other state-of-the-art methods that were mentioned in Chapter 4 on benchmarks described in Section 6.1.2

6.1 Methodology

6.1.1 ChampSim Simulator

For the simulation and evaluation of the models, a modified version of the ChampSim [36] simulator was used. ChampSim is a trace-based simulator that is commonly used in many competitions regarding memory prefetching and branch prediction and for microarchitecture studies. It is widely used and supported with high customizability. It models out-of-order execution core and multicore systems while allowing full customization in branch prediction, data prefetching, and cache replacement policies.

The ChampSim simulator uses as input traces of previously executed programs and applications to simulate every time it needed the execution on many variations of some mechanisms. The traces are created with the usage of the PIN tool [37]. It produces a detailed performance output of each execution with statistics regarding the simulation.

The modified ChampSim simulator provides a framework to implement machine learning models in Python with great abstraction. Although in the current work the latest modifications were not used completely. The data loading for the training and inference of the ML models was overridden. A custom data loader in order to improve the performance of the operation was used, implemented with Pandas[61] and PyTorch. The file loading prefetch of the modified version was used to feed the results of the new and custom models with some adjustments regarding the simultaneous execution of a combination of prefetchers or files.

The file prefetcher provided with this version was edited to accommodate custom prefetchers. An oracle version of a prefetcher was implemented to fit the ChampSim simulator and a combined version of file loading with other implemented prefetchers was introduced.

The simulator allows modification of the branch predictors, the entire memory hierarchy, and each layer's characteristics and size. The parameters of the memories are shown in Table 6.1

	Instruction TLB	Data TLB	2nd Level TLB	L1 Instruction Cache
Set	16	16	128	64
Way	4	4	12	8
Read Queue Size	16	16	32	64
Write Queue Size	16	16	32	4
Prefetch Queue Size	0	0	0	32
MSHR Size	8	8	16	8
Latency	1	1	8	4

	L1 Data Cache	L2 Cache	Last Level Cache
Set	64	1024	$N_{CPUS} * 2048$
Way	12	8	16
Read Queue Size	64	32	$N_{CPUS}*$L2C_MSHR
Write Queue Size	4	32	$N_{CPUS}*$ L2C_MSHR
Prefetch Queue Size	8	16	$N_{CPUS} * 32$
MSHR Size	16	32	$N_{CPUS} * 64$
Latency	5	10	20

Table 6.1: ChampSim Characteristics of caches

6.1.2 Benchmarks

Three benchmark suites were provided by the Machine Learning Architecture Systems Competition 2021 in the context of the International Symposium of Computer Architecture 2021. These three benchmark suites are extensively used in computer architecture research for comparison because they provide a spherical experience of real use cases with different profiles.

SPEC 2006

The Standard Performance Evaluation Corporation (SPEC) 2006 Benchmark [38] suite is an industry-standardized, CPU-intensive for stress testing processors, memory subsystems and other areas. It profiles different workloads based on real-world applications. There are two major categories of SPEC 2006 the floating point components and the integer point components.

The integer point components are shown in Table 6.2 and the floating point in Table 6.3

Benchmark	Language	Application Area	Description
403.gcc	C	C Compiler	Based on gcc Version 3.2, generates code for Opteron
429.mcf	C	Combinatorial Optimization	Vehicle scheduling. Uses a network simplex algorithm to schedule public transport.
462.libquantum	C	Physics/ Quantum Computing	Simulates a quantum computer running Shor's polynomial-time factorization algorithm
471.omnetpp	C++	Discrete Event Simulation	Uses the OMNet++ discrete event simulator to model a large Ethernet campus network
473.astar	C++	Path-finding Algorithms	Pathfinding library for 2D maps including the well-known A* algorithm.

Table 6.2: Integer Components of SPEC CPU2006

Benchmark	Language	Application Area	Description
410.bwaves	Fortran	Fluid Dynamics	Computes 3D transonic transient laminar viscous flow
433.milc	C	Physics / quantum Chromodynamics	A gauge field generating program for lattice gauge theory programs with dynamical quarks
437.leslie3d	Fortran	Fluid Dynamics	Computational Fluid Dynamics (CFD) using Large-Eddy Simulations with Linear-Eddy Model in 3D. Uses the MacCormack Predictor-Corrector time integration scheme.
450.soplex	C++	Linear Programming, Optimization	Solves a linear program using a simplex algorithm and sparse linear algebra. Test cases include railroad planning and military airlift models.
459.GemsFDTD	Fortran	Computational Electromagnetics	Solves the Maxwell equations in 3D using the finite-difference time-domain (FDTD) method.
470.lbm	C	Fluid Dynamics	Implements the "Lattice-Boltzmann Method" to simulate incompressible fluids in 3D
482.sphinx3	C	Speech recognition	A widely-known speech recognition system from Carnegie Mellon University

Table 6.3: Floating Components of SPEC CPU2006

SPEC 2017

The Standard Performance Evaluation Corporation (SPEC) 2017 Benchmark [39] suite contains SPEC's next-generation, industry-standardized, CPU-intensive suites for measuring and comparing computational performance, stressing a system's processor, memory subsystems and compilers. It is designed to measure performance in a wide range of hardware based on real user applications. There are 4 categories SPECrate 2017 Integer and floating point and SPECspeed 2017 Integer and Floating Point. In the current thesis, only SPECspeed 2017 integer and floating components were used.

KLOC = line count (including comments/whitespace) for source files used in a build / 1000

The integer point components are shown in Table 6.4 and the floating point in Table 6.5

Benchmark	Language	KLOC	Description
602.gcc	C	1304	GNU C compiler
605.mcf	C	3	Route planning
620.omnetpp	C++	134	Discrete Event simulation - computer network
623.xalancbmk	C++	520	XML to HTML conversion via XSLT

Table 6.4: Integer Components of SPECspeed CPU2017

Benchmark	Language	KLOC	Description
607.cactuBSSN	C++, C, Fortran	257	Physics: relativity
605.mcf	C	3	Route planning
619.lbm	C	1	Fluid dynamics
621.wrf	Fortran, C	991	Weather forecasting
649.fotonik3d	Fortran	14	Computational Electromagnetics
654.roms	Fortran	210	Regional ocean modeling

Table 6.5: Floating Components of SPECspeed CPU2017

GAP

GAP[40] is a graph processing benchmark suite helping to standardize graph processing evaluations. It specifies graph kernels, input graphs and measurement methodologies. Every kernel is developed with C++11 using the best practices and features. The graph kernels used are the following:

- **Betweenness Centrality(bc)**: Approximates the Betweenness centrality score for all vertices in a graph.
- **Breadth-First Search(bfs)**: Traverses all vertices starting from a source vertex by traversing first all vertices at the current depth before moving to the next one.
- **Connected Components(cc)**: Labels all vertices by their connected component and

each connected component is assigned its own label. In directed graphs, it requires weakly connected components.

- **PageRank(pr)**: Counts the PageRank score for all vertices in the graph. PageRank woks by counting the number and quality of links to a vertex to determine an estimate of importance of that vertex.

- **Single-Source Shortest Paths(sssp)**: Computes the distances of the shortest paths from a given source vertex to every other reachable vertex. The distance between two vertices is the minimum sum of edge weights along a path connecting them.

6.1.3 Oracle Prefetching

A first approach was to run simulations using an oracle prefetcher that could always prefetch the next address correctly. The first iteration of the oracle prefetcher even with 100% accuracy does not provide the best results because it does not factor in the timeliness of the prefetches. Therefore an oracle prefetcher that predicts further steps into the future was simulated.

After considering the deltas for the optimal step it is seen that as we go further into the future deltas are more randomized and less regular.

This causes classification to be more difficult and covers way fewer load instructions. To reduce the number of different deltas it was explored to try differentiating per PC and finding the optimal future step for the same PC.

The results of both oracle prefetchers are in the following tables. In Table 6.6 we see that the use of per PC distance has a bit lower IPC performance in general, but the difference in IPC is minimal compared to the gains in accuracy and model implementation we gain in an actual model.

Here we have only the per PC oracle distance and we see that we have an important improvement on IPCs, while keeping a smaller step.

	Global Distance			**Per PC distance**		
Trace	Step	IPC	IPC Improvement %	Step	IPC	IPC Improvement %
410.bwaves-s0	22	1.29762	106.71	3	1.29601	106.46
410.bwaves-s1	5	1.2622	119.83	5	1.21089	110.89
429.mcf-s0	5	0.142182	88.44	10	0.141433	87.45
429.mcf-s1	29	0.126873	104.89	15	0.127265	105.52
433.milc-s0	26	0.952528	99.85	6	0.936459	96.48
433.milc-s1	20	0.916549	96.7	4	0.8965	92.41
437.leslie3d-s0	30	0.976443	102.74	13	0.99986	107.6
437.leslie3d-s1	30	0.991193	101.21	13	1.00746	104.51
450.soplex-s1	10	0.518373	70.7	9	0.551101	81.47
459.GemsFDTD-s0	5	0.676177	97.07	6	0.669932	95.25
459.GemsFDTD-s1	5	0.828642	69.96	3	0.828965	70.03
462.libquantum-s0	10	0.947657	86.36	15	0.958868	88.56
462.libquantum-s1	10	1.09665	51.87	9	1.08039	49.62
470.lbm-s0	10	0.72944	25.92	2	0.712361	22.97
470.lbm-s1	10	0.737215	25.86	2	0.71974	22.89
471.omnetpp-s0	10	0.600527	10.54	8	0.598602	10.18
471.omnetpp-s1	10	0.503991	62.86	3	0.498911	61.22
473.astar-s0	11	1.11121	14.43	3	1.1097	14.27
473.astar-s1	5	0.517312	15.31	15	0.515967	15.01
482.sphinx3-s0	10	1.331	99.94	5	1.31683	97.81
482.sphinx3-s1	10	1.39591	99.77	4	1.38183	97.76
MEANS	**13.4761904762**	**0.7289969748**	**73.8552380952**	**7.2857142857**	**0.7261998886**	**73.2550607369**

Table 6.6: Best step Oracle prefetchers in SPEC06

	Global Distance			**Per PC distance**		
Trace	Step	IPC	IPC Improvement %	Step	IPC	IPC Improvement %
bc-0	32	0.225297	51.58	9	0.212032	42.65
bc-12	35	0.207981	44.67	7	0.206353	43.54
bc-3	22	0.261554	60.5	11	0.227824	39.8
bc-5	33	0.358093	63.86	17	0.282438	29.24
bfs-10	26	0.490346	80.13	3	0.370895	36.25
bfs-14	1	0.462941	72.22	3	0.35984	33.87
bfs-3	32	0.489791	76.04	7	0.371151	33.4
bfs-8	24	0.471714	75.4	3	0.360378	34
cc-13	31	0.377658	69.67	9	0.317701	42.73
cc-14	32	0.414972	68.75	9	0.350914	42.7
cc-5	31	0.460715	69.07	10	0.390226	43.2
cc-6	31	0.336557	64.67	9	0.29465	44.16
pr-10	2	0.113417	83.7	6	0.0868574	40.68
pr-3	11	0.112821	87.76	6	0.0853714	42.08
pr-5	1	0.113471	82.96	6	0.0870978	40.43
sssp-10	29	0.592979	75.02	10	0.488813	44.28
sssp-14	29	0.592694	75.06	10	0.488295	44.22
sssp-3	30	0.59222	74.96	9	0.488981	44.46
sssp-5	29	0.592943	75.11	9	0.48919	44.46
MEANS	**24.2631578947**	**0.3360217918**	**71.1121052632**	**8.0526315789**	**0.2759496478**	**40.3236842105**

Table 6.7: Best step Oracle prefetchers in GAP

GAP suite shows a different profile than the other ones by having a significant difference with global distance being oracle being better, but we see that even with an oracle prefetcher the theoretical best margin for improvement is smaller.

6.1.4 Prefetcher Setups

For every benchmark of the suites we ran and tested the following prefetcher setups.

The non Machine Learning setups we ran are:

- Baseline(no) without any prefetcher
- Next Line Prefetcher(next_line)
- Best Offset Prefetcher(bo)
- Irregular Stream Buffer(sisb)

And the Machine Learning setups were:

- TransFetch
- Our LSTM model (LSTM)

Finally, we implemented collaboration prefetchers with some of the previous ones:

- Irregular Stream Buffer with Best Offset Prefetcher(sisb_bo)
- Our model with Next Line Prefetcher(LSTM_next)
- Our model with Best Offset Prefetcher(LSTM_bo)
- Our model with Irregular Stream Buffer(LSTM_sisb)

All setups had a prefetch degree of 2 prefetches and were evaluated mainly on the performance of IPC and IPC Improvement. The other metrics we present are for assisting the design process and understanding the results of the prefetchers.

6.2 Results

6.2.1 Prefetch Distance results

To show the improvement of prefetching with a farther distance than one we trained and ran the models with distance one and with optimal distance as found by oracle prefetching.

The results are shown in Figure 6.1 for SPEC06

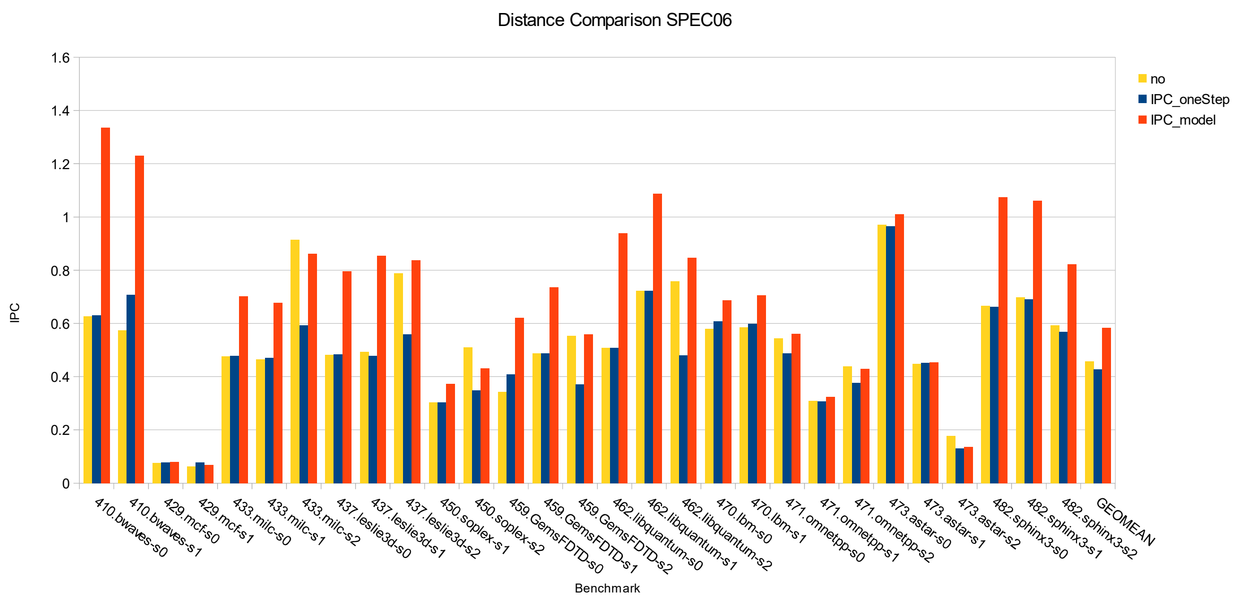


Figure 6.1: IPC Comparison of One step, Optimal Step and no prefetching Models SPEC06

Here we notice the overwhelming advantage of distance prefetching as it improves overwhelmingly over the one-Step model. There is only one case where one-Step performs better, 429.mcf-s1, where the total number of deltas on the required step is high and the coverage of the filtered deltas is reduced compared to other cases.

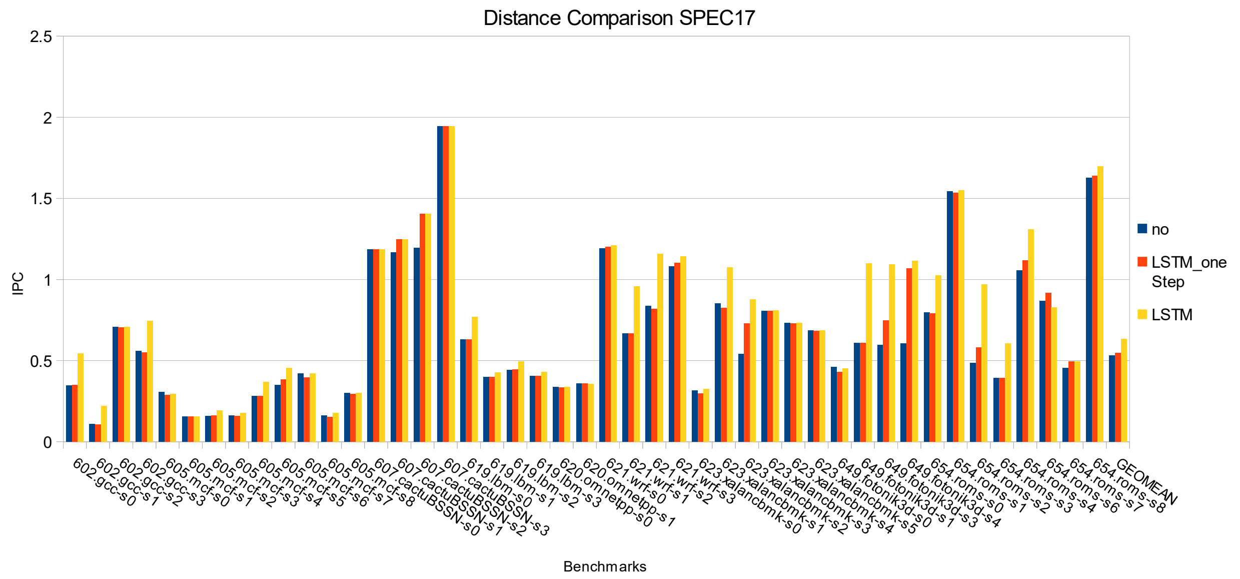


Figure 6.2: IPC Comparison of One step, Optimal Step and no prefetching Models SPEC17

In SPEC17 we see a different profile in some cases than before, where the optimal step is one, e.g. 602.gcc-s2, 607.cactuBSSN and 623.xalancbmk. Again we have an outlier case such as 654.roms-s6 where oneStep performs better than the optimal Step.

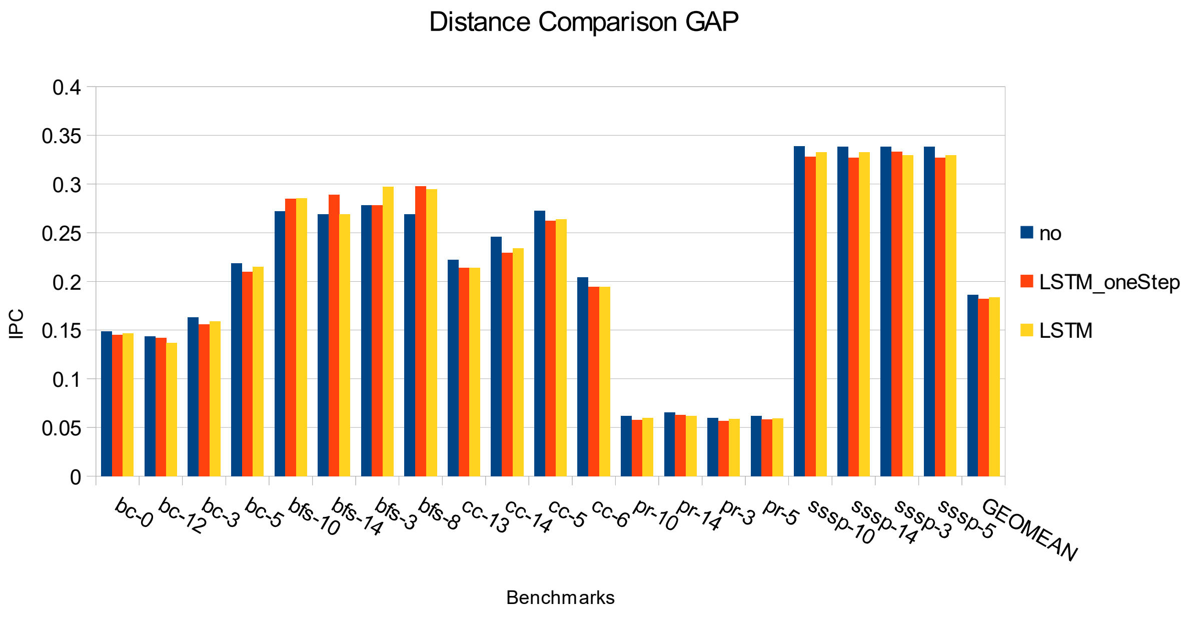


Figure 6.3: IPC Comparison of One step, Optimal Step and no prefetching Models GAP

Here we see that in most cases both LSTM models lose performance except in bfs benchmarks, but still, the optimal step performs better than the oneStep. The entire benchmark suite has a high number of deltas and lower IPC on basis than the previous ones. This means that the aggressive nature of prefetching 2 addresses constantly impacts the performance negatively.

The geometric mean of IPCs per Benchmark suite is shown in Figure 6.4.

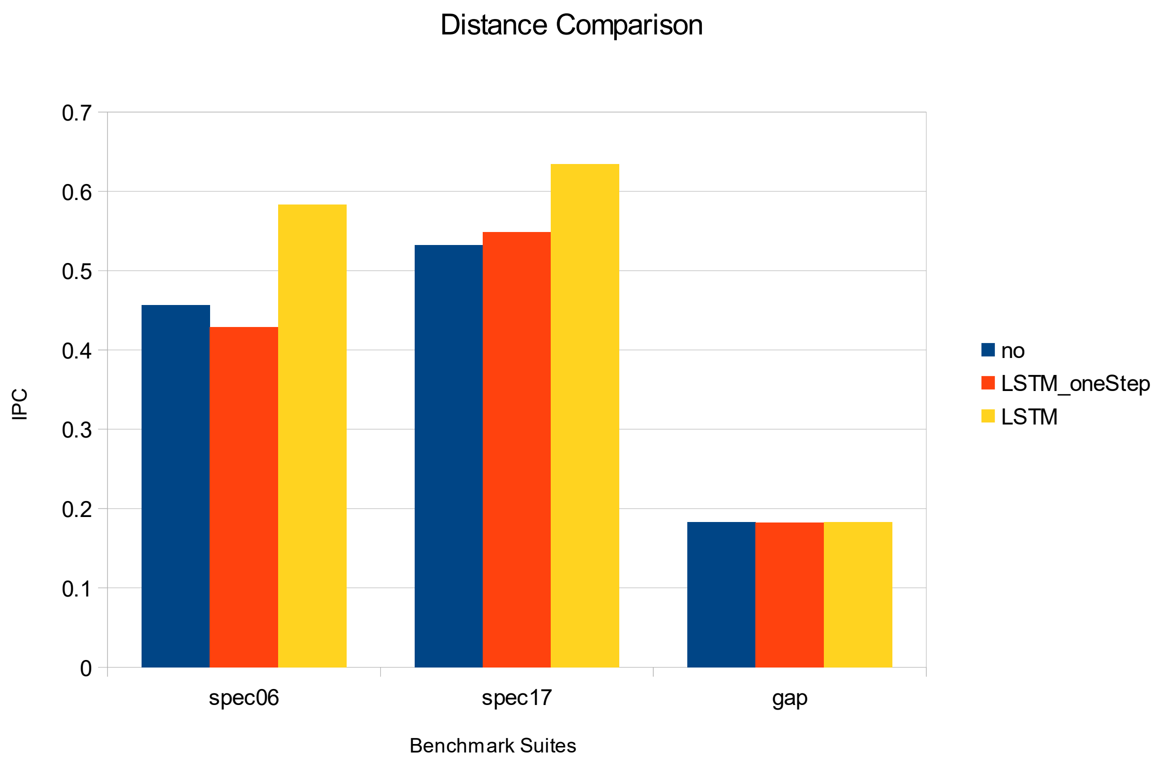


Figure 6.4: IPC Comparison of One step, Optimal Step and no prefetching Models

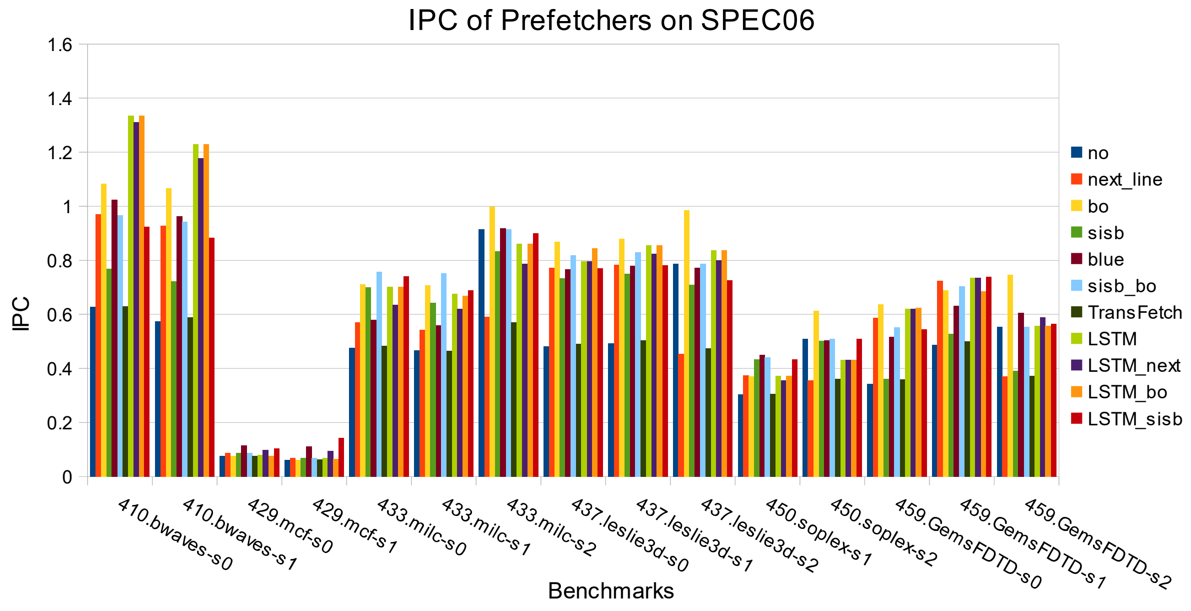
We see that almost everywhere the farther distance produces much better results than simply the next accessed address. The next address prefetching mostly harms performance, because the prefetches do not arrive on time and use the limited resources and bandwidth.

6.2.2 Model Results

We explored different prefetcher models, non Machine Learning, and Machine Learning ones. The non Machine Learning prefetchers we simulated were the Next Line Prefetcher(next_line)The Best Offset(bo) and Irregular Stream Buffer(sisb) were tested on their own and working together(sisb_bo). The only Machine Learning prefetcher that was tested was TransFetch but the prefetch degree was reduced to 2 to match all the other prefetchers. Our LSTM model(LSTM) is tested on its own and along with three other prefetchers, the Best Offset Prefetcher(LSTM_bo), the next line(LSTM_next), and the Irregular Stream Buffer(LSTM_sisb). The complete table results are in Appendix A.

Instructions Per Cycle

For **SPEC06** the results are shown in Figure 6.5



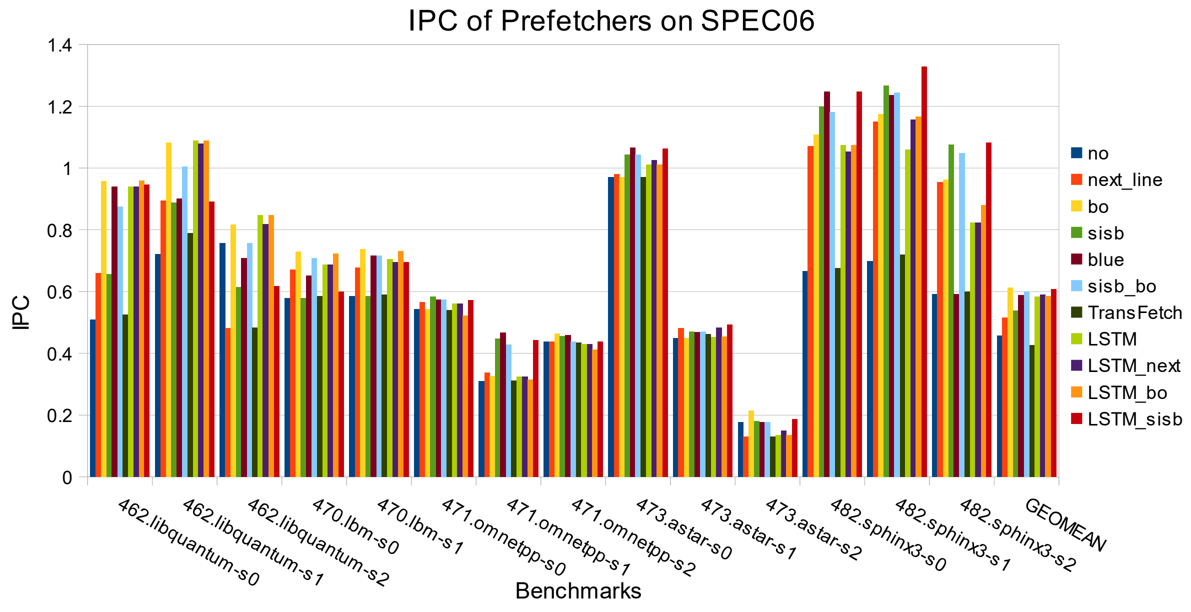


Figure 6.5: Spec06 IPC

From the graphs, we see that in most cases the better performing prefetchers are the Best Offset and the LSTM with ISB prefetcher being very close, even though each one of them on its own is not the best in most cases. There are some exceptions such as 410.bwaves where the plain LSTM model performs the best, while ISB and LSTM_sisb are not on par, which can be explained by the results shown in Section 6.2.2. Blue in some cases like 429.mcf is the best prefetcher and performs well in 471.omnetpp. Another exception happens in 433.milc and 482.sphinx3, where simpler methods ISB and next line respectively are outperforming the other ones.

For **SPEC17** the results are shown in Figure 6.6

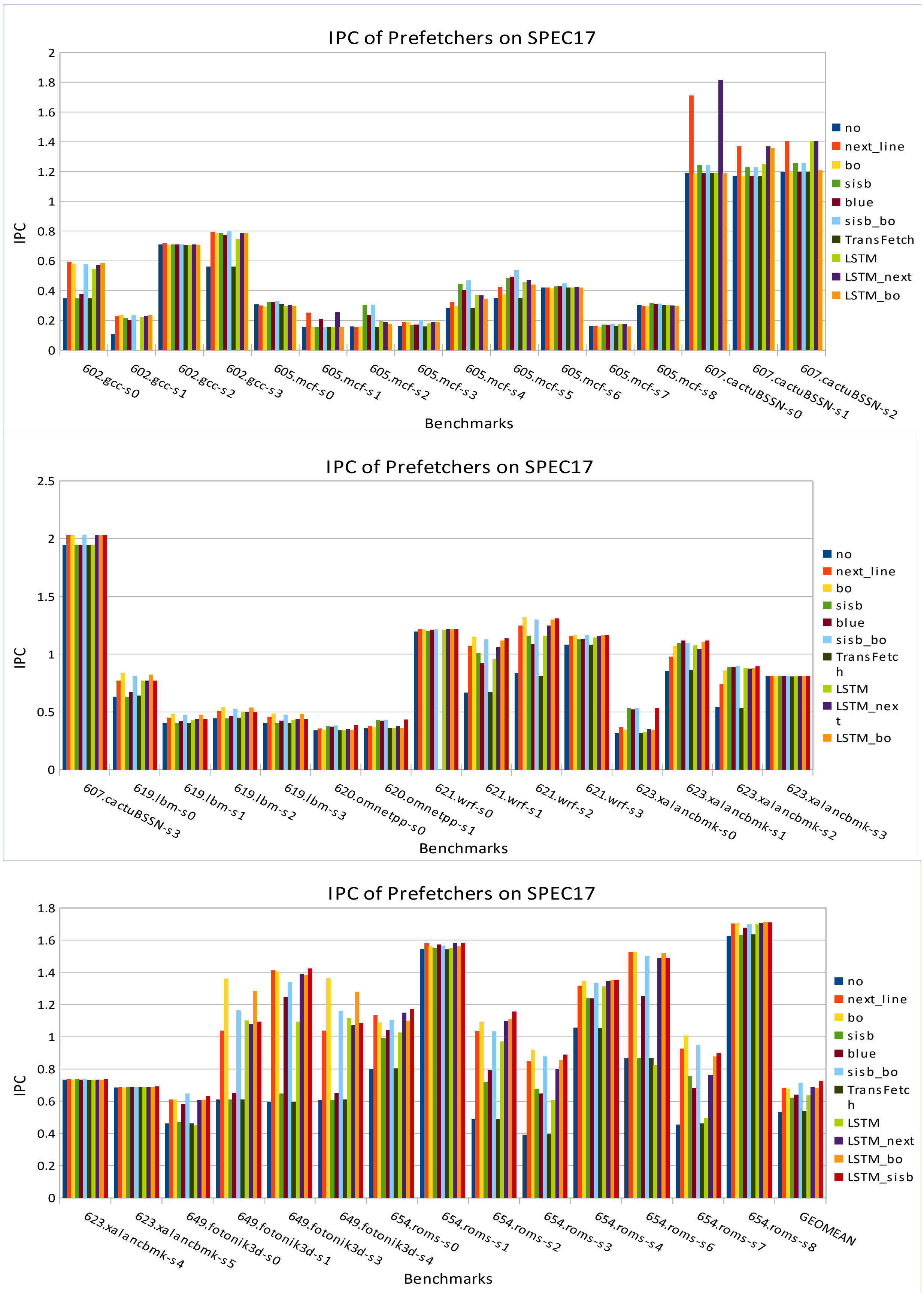
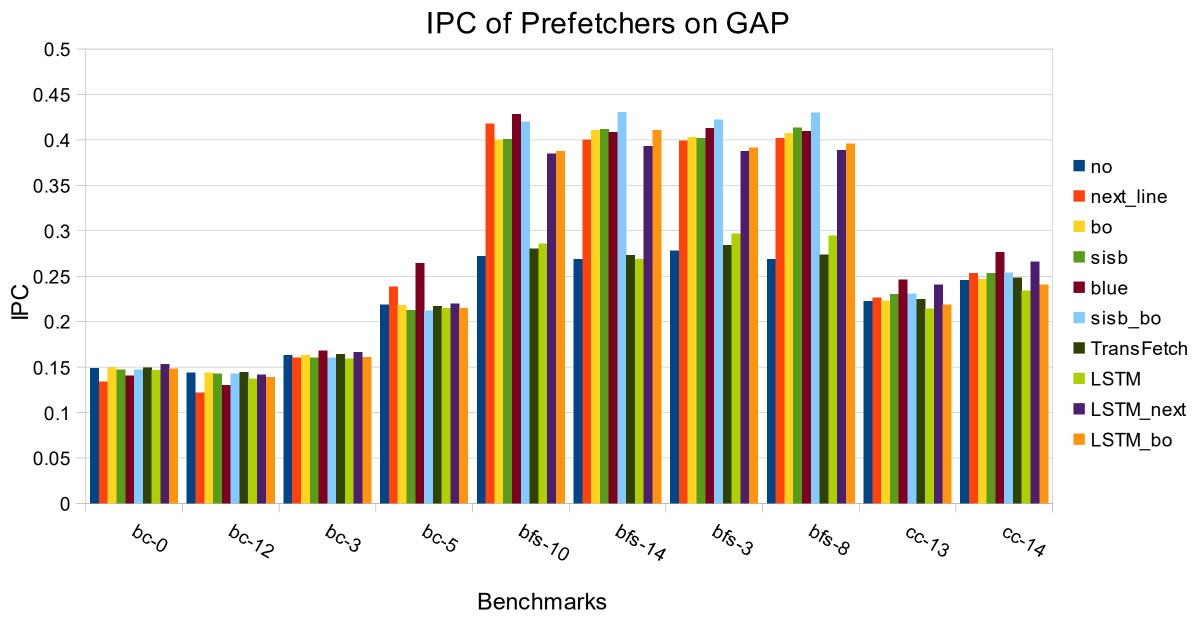


Figure 6.6: Spec17 IPC

In the updated SPEC Suite the top performers are sisb_bo and LSTM_sisb, even though on their own they fall behind. There are some outlier cases such as 607.cactuBSSN where next line and LSTM_next_line. Furthermore there are cases where all prefetchers are close such as 602.gcc-s2, 605.mcf-s8, 623.xalancbmk-s3, -s4, -s5.

For **GAP** the results are shown in Figure 6.7



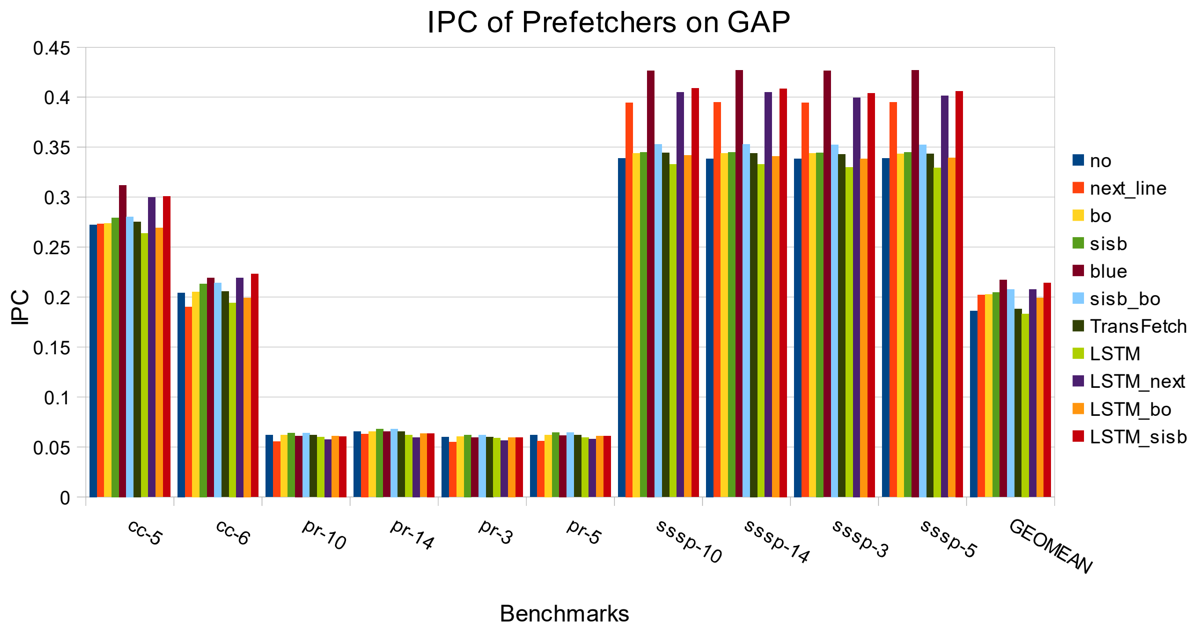


Figure 6.7: GAP IPC

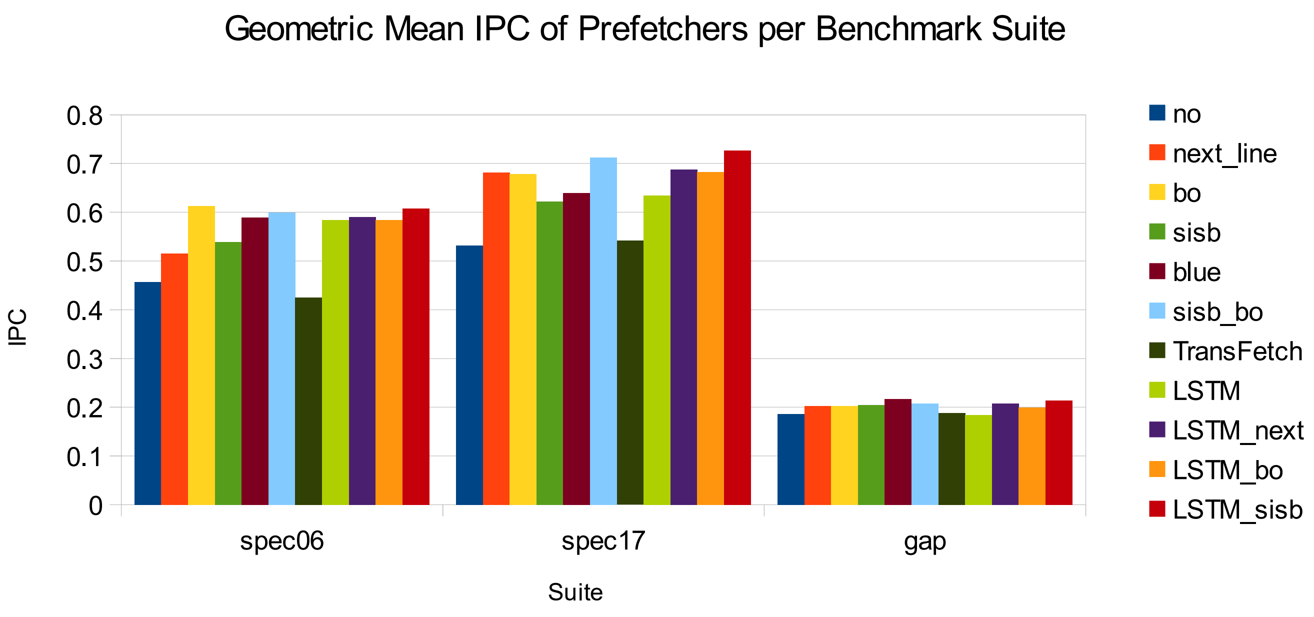


Figure 6.8: Geometric mean of models per Suite

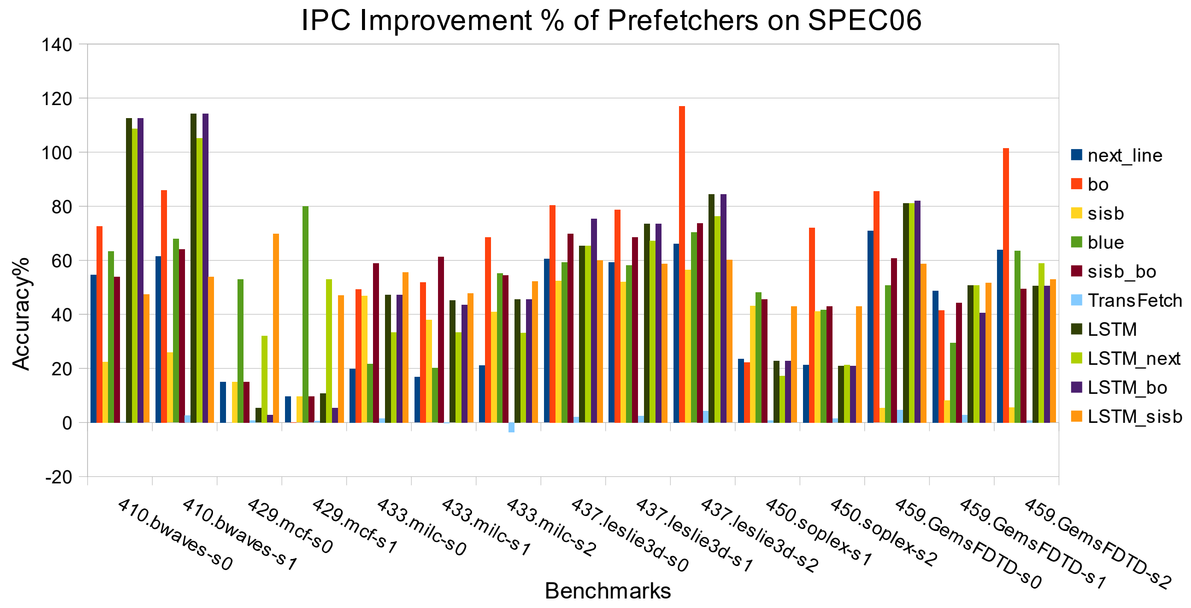
In GAP we see a different profile than the other benchmark suites. Here the top performer is the blue prefetcher and again our LSTM_sisb is a close second. The bfs benchmarks give a different profile where almost all prefetchers perform well.

To have the complete picture we present the geometric means of each model per benchmark suite in Figure 6.8

Here we see the overall result differences between each suite. Our LSTM model on its own performs decently in SPEC06 and SPEC17 and not well in GAP where we have lower IPCs. LSTM_sisb is the better performer in SPEC17 and second best in SPEC06 and GAP. Another great performer is sisb_bo being close to the top performers in all suites. BO outperforms all others in SPEC06, but in SPEC17 and GAP it has a similar performance as the next line. Blue is the best in GAP, but it falls behind in SPEC06 and SPEC17. Another solid performing prefetcher is our LSTM when working with next_line or BO. The Irregular Stream Buffer is not among the top performers just like our LSTM but when combined with it is generally the best. TransFetch seems to be very ineffective, which can be attributed to the reduction of the degree of the proposed solution. These results in IPC are explained by the Coverage **??** and Accuracy Figure 6.20 figures

IPC Improvement

For **SPEC06** the results are shown in Figure 6.9



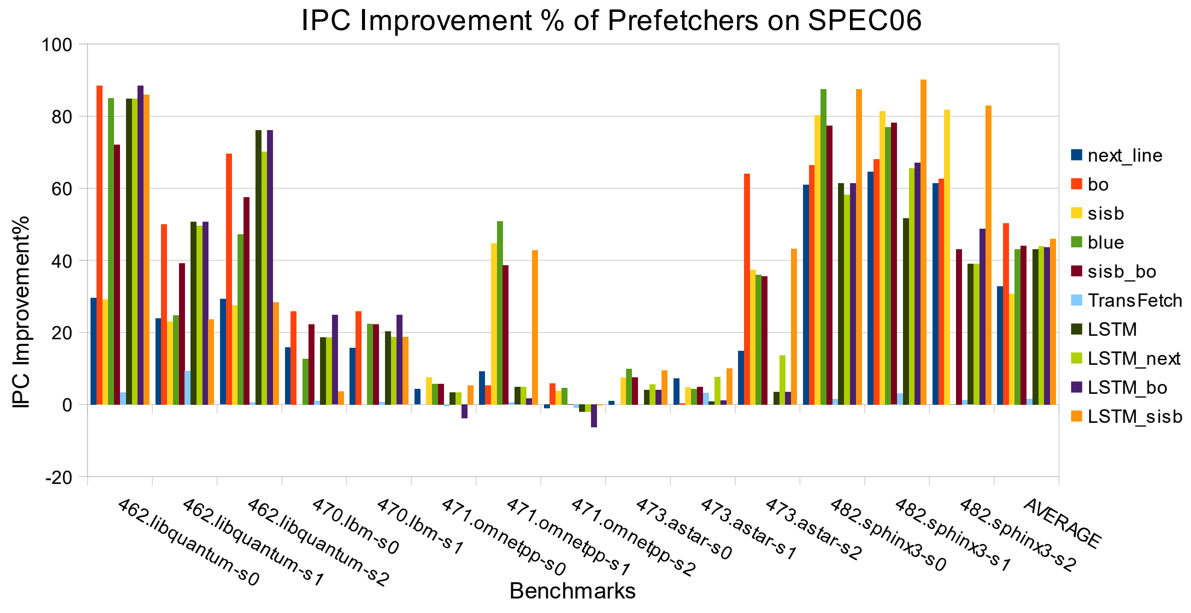


Figure 6.9: Spec06 IPC Improvement %

When looking at the improvement percentage of IPCs we see the differences in all cases even if they are at low IPC cases. The 471.omnetpp and 429.mcf are cases that the blue prefetcher that performs better while the others do not improve by a large margin. In 410.bwaves our LSTM models all perform outstandingly. In 437.leslie3d-s2 and 459.GemsFDTD-s2 the Best

Offset Prefetcher has a significant advantage over all others.

For **SPEC17** the results are shown in Figure 6.10

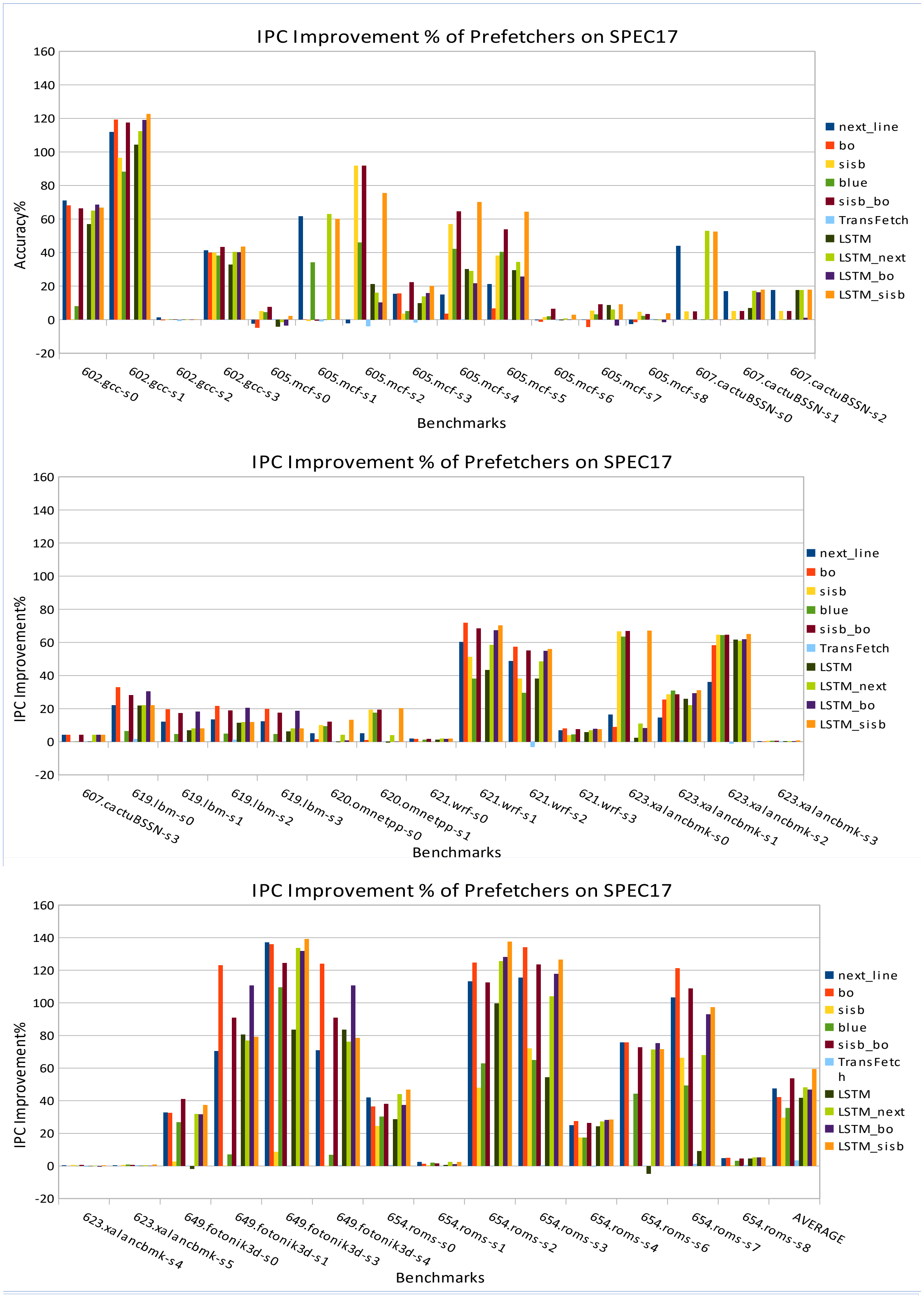
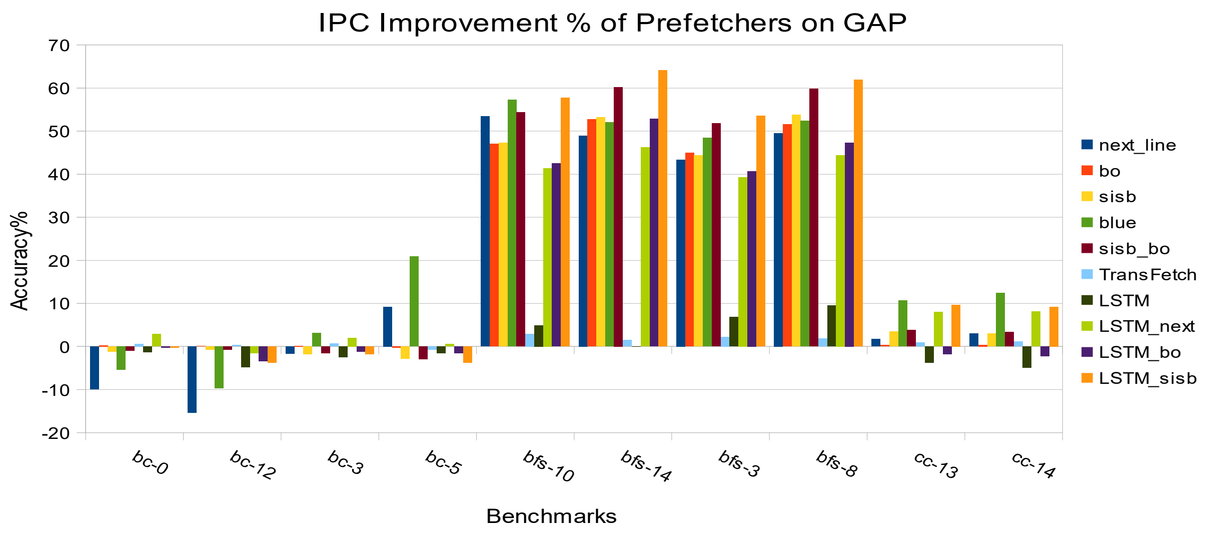


Figure 6.10: Spec17 IPC Improvement

SPEC17 we can group the benchmarks into two categories based on the impact of prefetching. The hard to predict with all prefetchers having a small impact are 605.mcf, 623.xalancbmk, and some 654.roms, and the easier to predict with higher performance gains in almost all prefetchers. In the greater performance gain category, the differences between the prefetchers are observable. If we compare it to the IPC, the next line seems to gain a greater improvement percentage than what was seen in just the IPC.

For **GAP** the results are shown in Figure 6.11



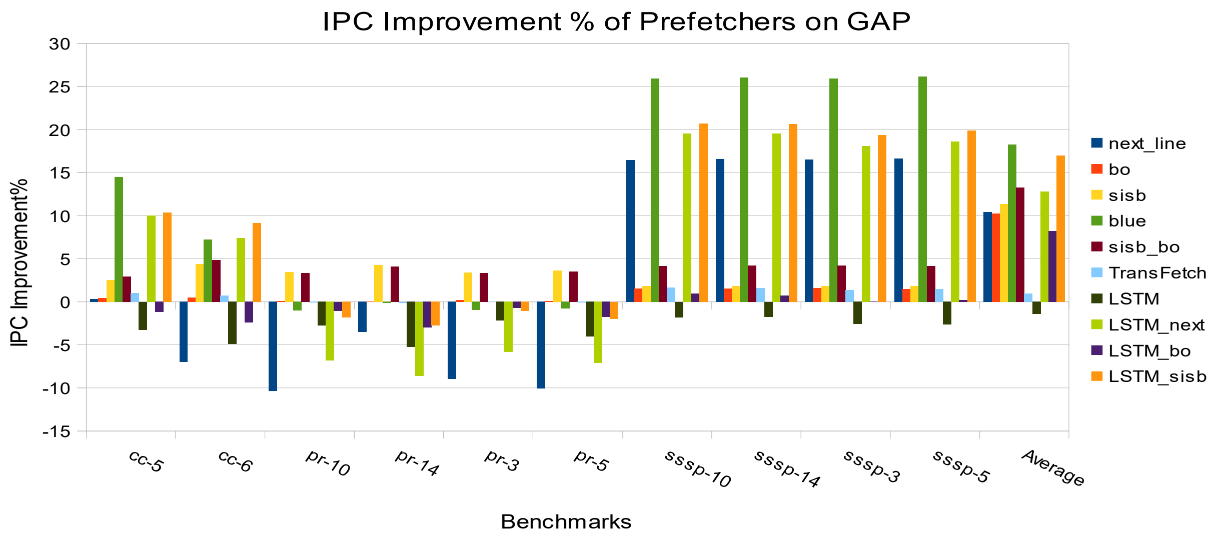


Figure 6.11: GAP IPC Improvement

GAP benchmark suite has benchmarks with negatively performing prefetchers. Here the plain LSTM model is almost everywhere negative and the simple ISB has smaller performance

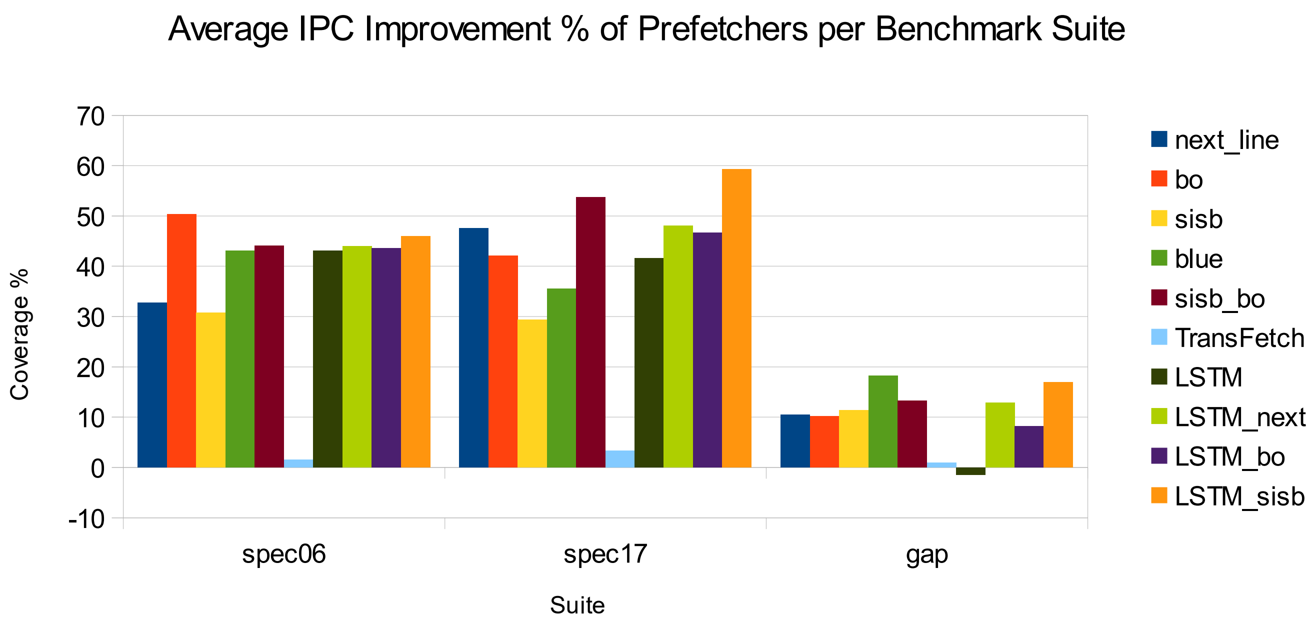


Figure 6.12: Average of IPC Improvement % of models per Suite

gains, but when both are used alongside one another improves the IPC by a large percentage outperforming their isolated on their one, being the second performer. We can clearly see the difference of bfs benchmarks from the others where all prefetchers, except TransFetch, perform well. The winner in this suite is blue where there is improvement in most cases and in some has a large difference from the others. Here the pr benchmarks are the deviation from the norm where only sisb performs well. The big difference here from the SPEC suites is that the Best Offset prefetcher does not work in some cases or has a small impact.

To have the complete picture we present the average means of each model per benchmark suite in Figure 6.12

The percentage improvement shows similar results as the plain IPC in Figure 6.8 but we can see more clearly the difference in the performance of each prefetcher. We can see that LSTM Model in GAP is hurting performance by a little bit and that the TransFetch has little impact compared to everything else. Another thing we can observe better is that Best Offset shines in SPEC06 but trails off in the other two Suites.

Accuracy

For each test run and benchmark, we present the accuracy in the following graphs.

For **SPEC06** the results are shown in Figure 6.13

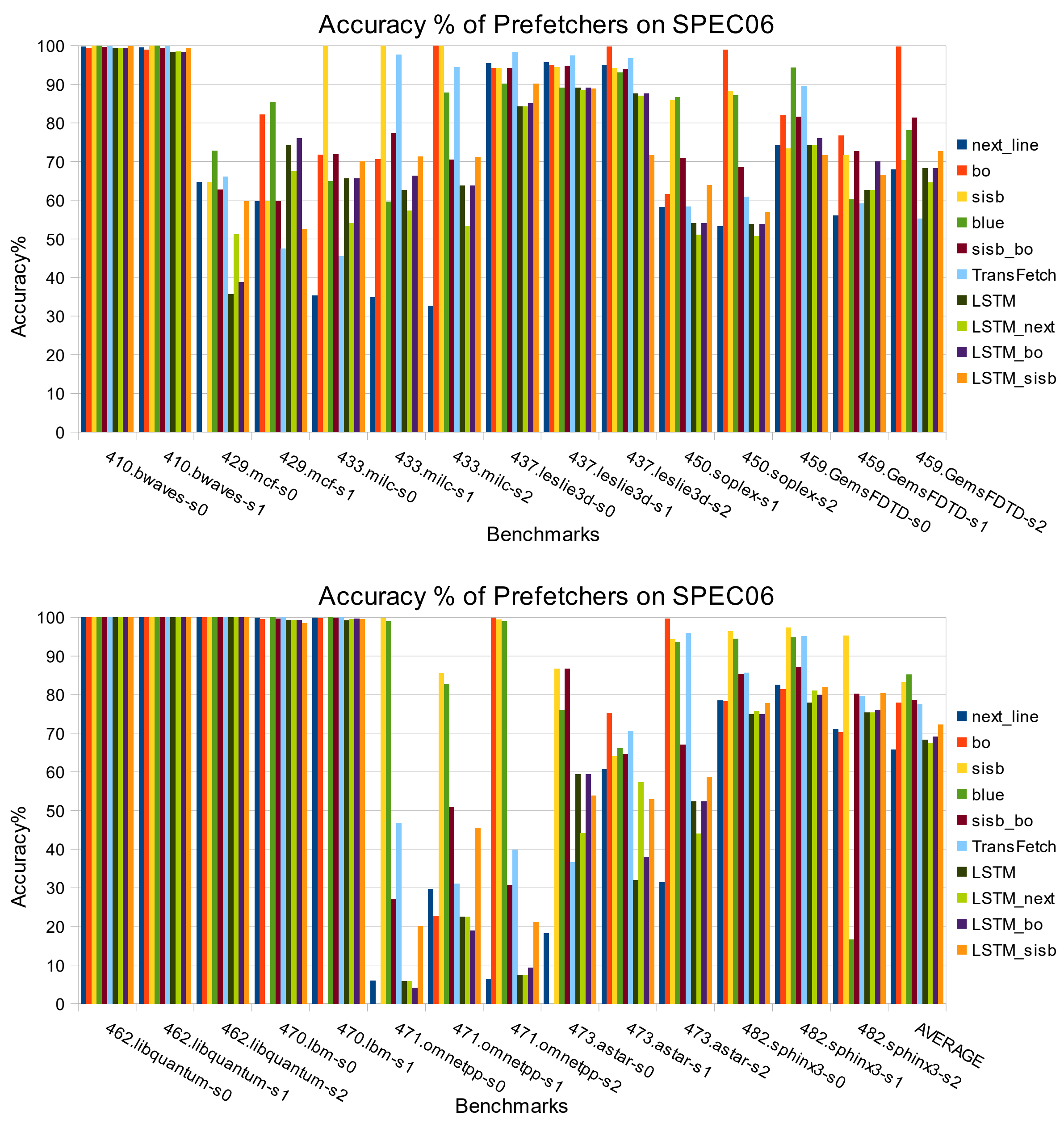
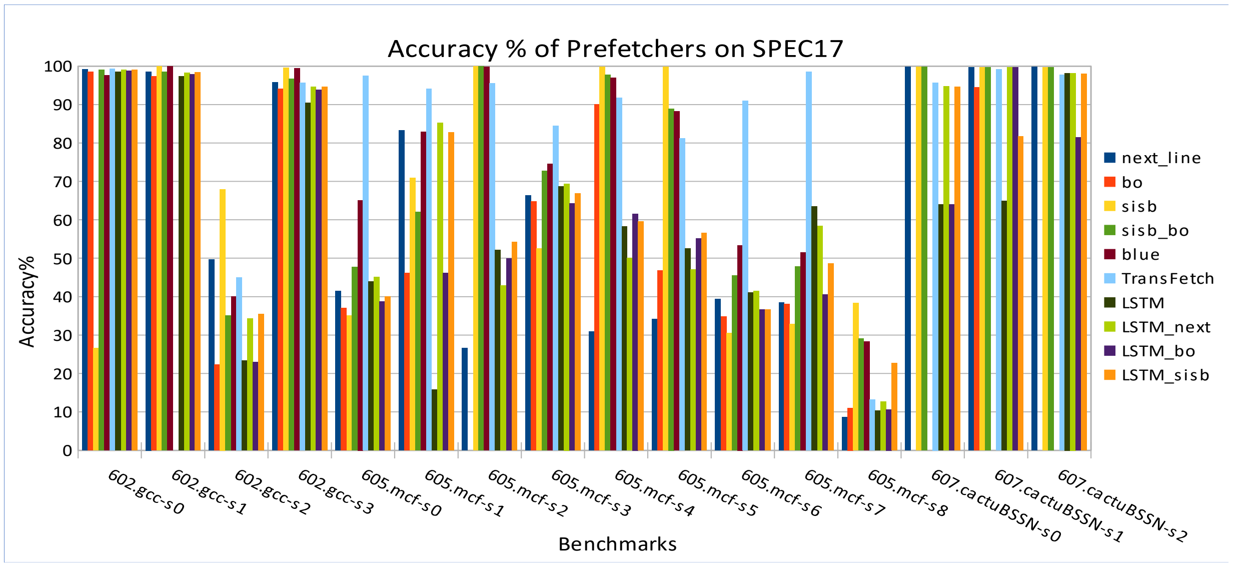
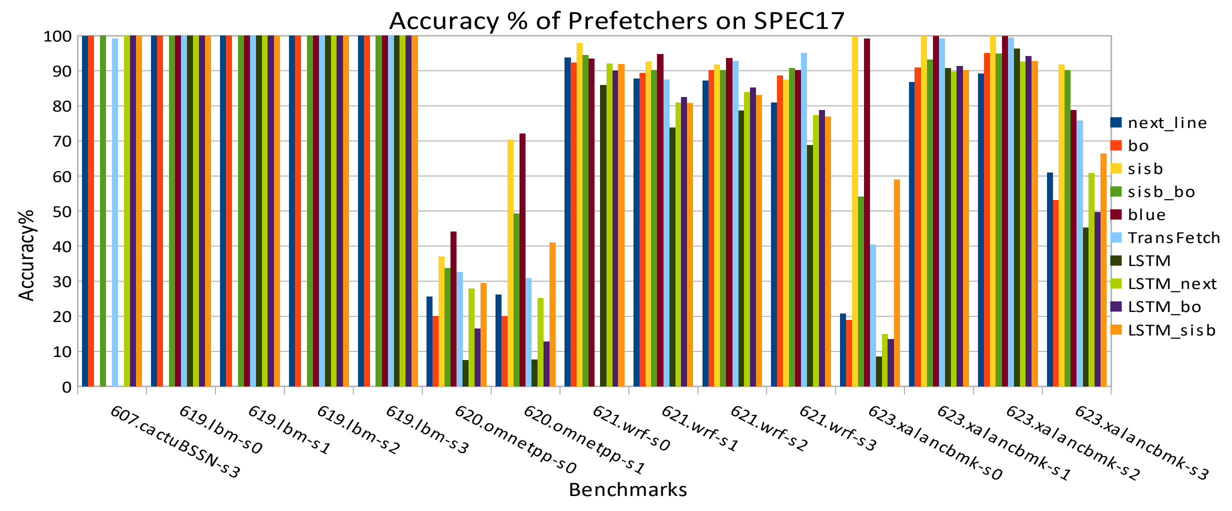


Figure 6.13: Spec06 Accuracy

Accuracy shows that there are easier to predict benchmarks where all have high accuracy in their predictions such as 410.bwaves and 462.libquantum and others where most struggle with only a couple of them having higher accuracy, like 429.mcf, 433.milc, 471.omnetpp and 473.astar. As expected high performers have high accuracy, but the difference is that sisb has high accuracy in its predictions just as TransFetch. Our LSTM has average accuracy in total predictions.

For **SPEC17** the results are shown in Figure 6.14





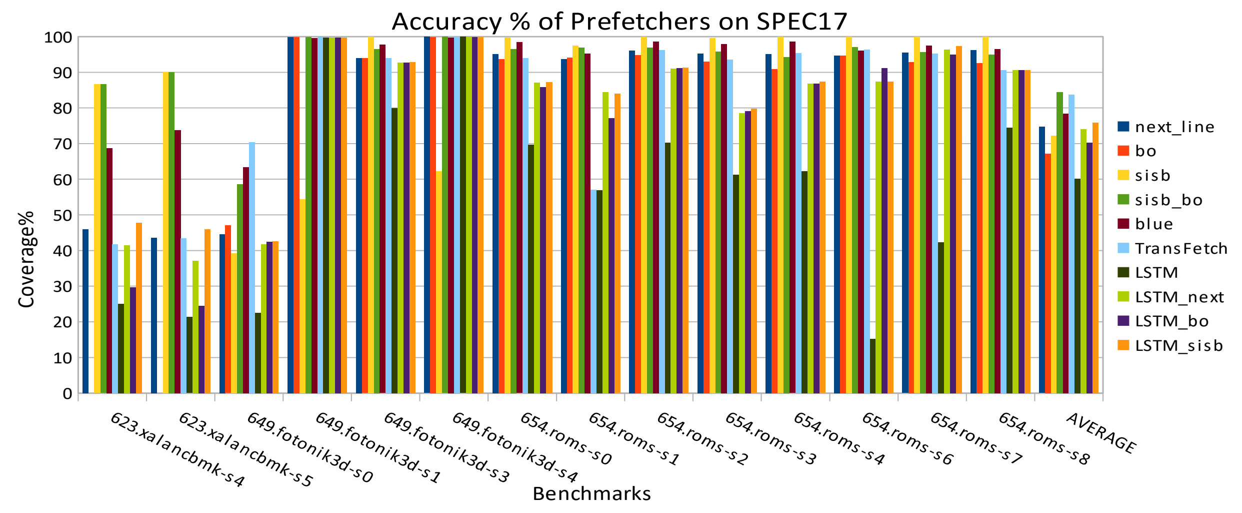
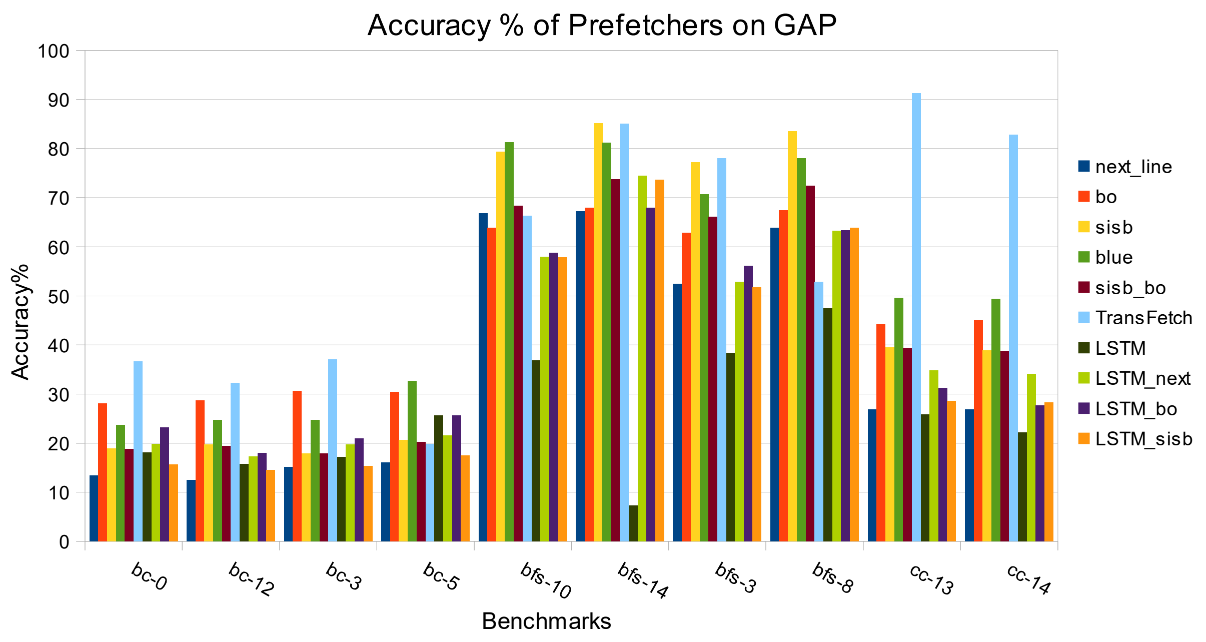


Figure 6.14: Spec17 Accuracy

The image here is similar profile of SPEC06 with high accuracy and low accuracy benchmarks. Peculiar cases are the 607.cactuBSSN where the prefetchers that perform have high accuracy, while others do not perform. Accuracy is highest with sisb_bo and Transfetch, but the top IPC performer, LSTM_sisb, is not that close in accuracy.

For **GAP** the results are shown in Figure 6.15



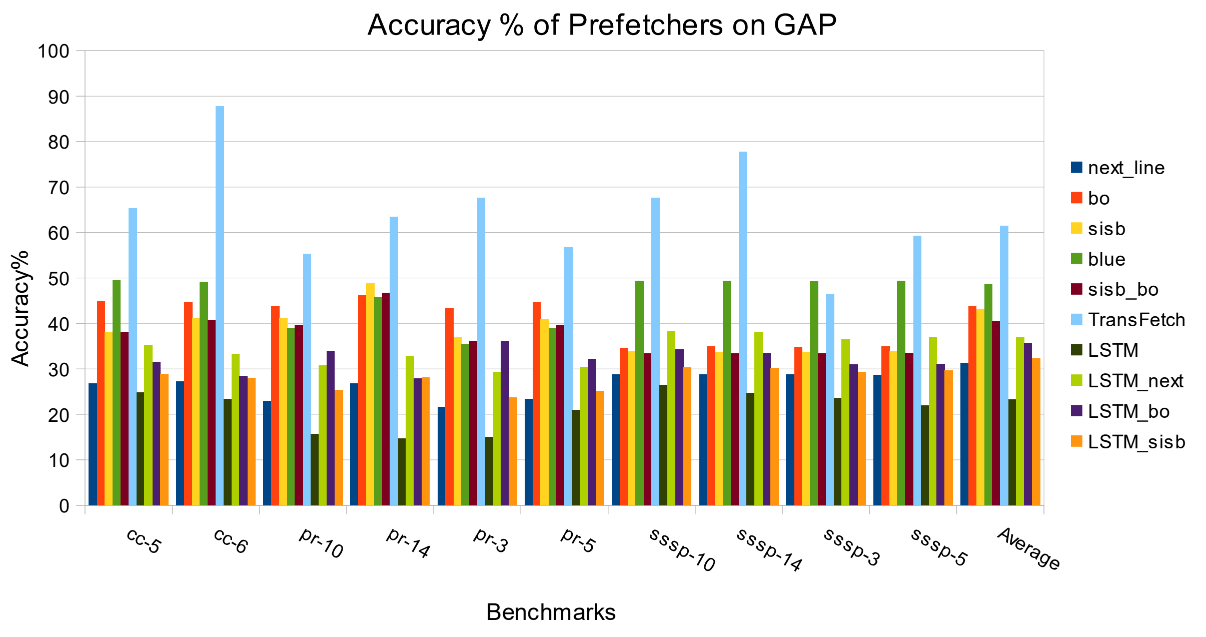


Figure 6.15: GAP Accuracy

All prefetchers in GAP have low accuracy in almost all cases, except in bfs where all have increased around 60%. Again TransFetch is very accurate followed by Blue BOP and SISB. All LSTM models have average accuracy compared to the other prefetchers.

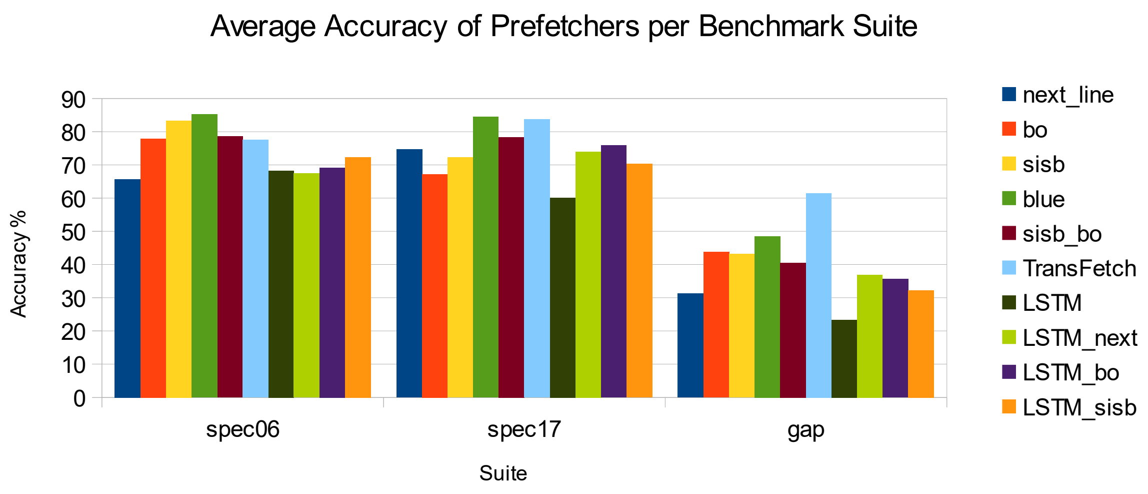
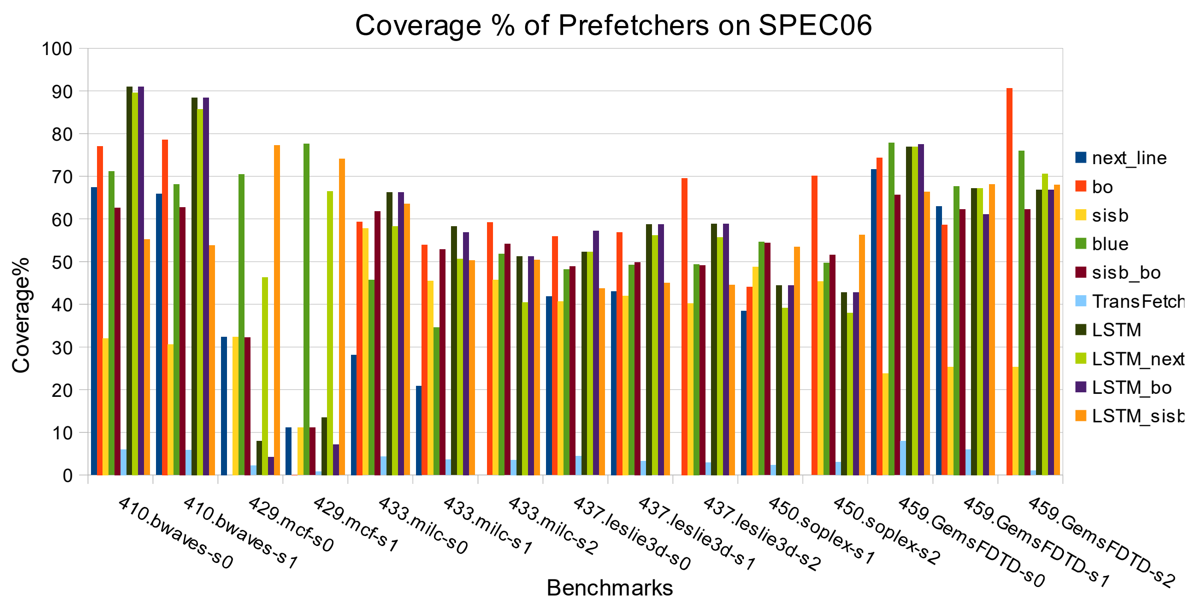


Figure 6.16: Average Accuracy % of models per Suite

Generally, the accuracy of the prefetchers is similar in the SPEC suites with the difference of next_line prefetcher having higher accuracy than Best Offset and SISB in the newer version of the suite. Transfetch has high accuracy predictions and our LSTM, individually has lower accuracy than the others, and average combined the others even if it outperforms them

Coverage

For **SPEC06** the results are shown in Figure 6.17



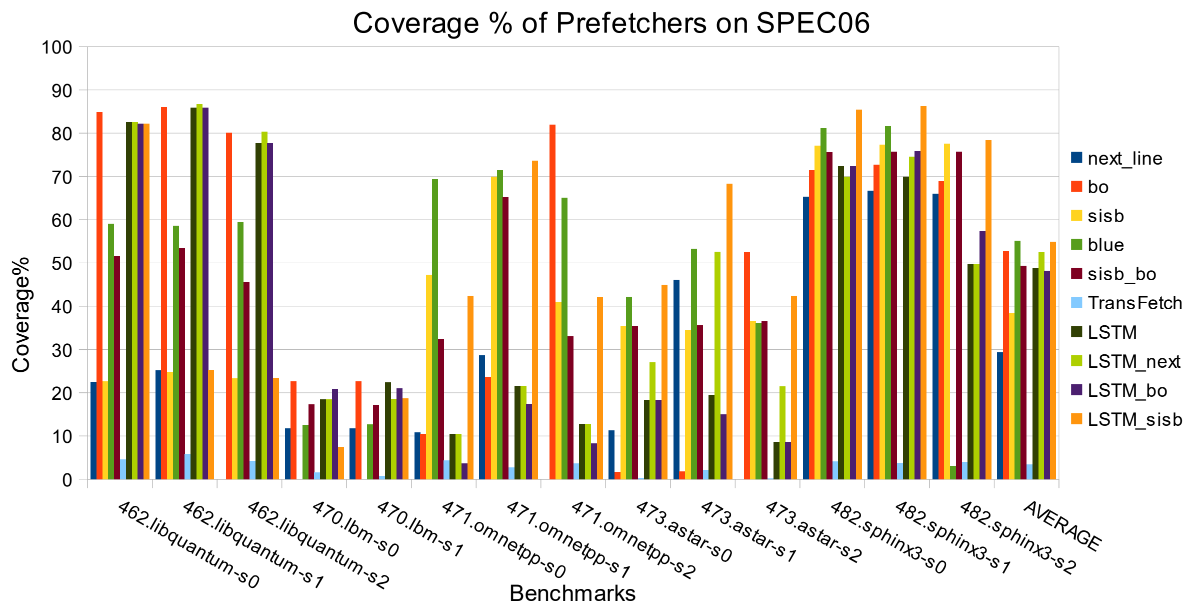


Figure 6.17: Spec06 Coverage

Coverage percentage matches the IPC Improvement chart very closely in all benchmarks and shows the top performers have higher coverage. TransFetch has the lowest coverage of all making it the worst.

For **SPEC17** the results are shown in Figure 6.18

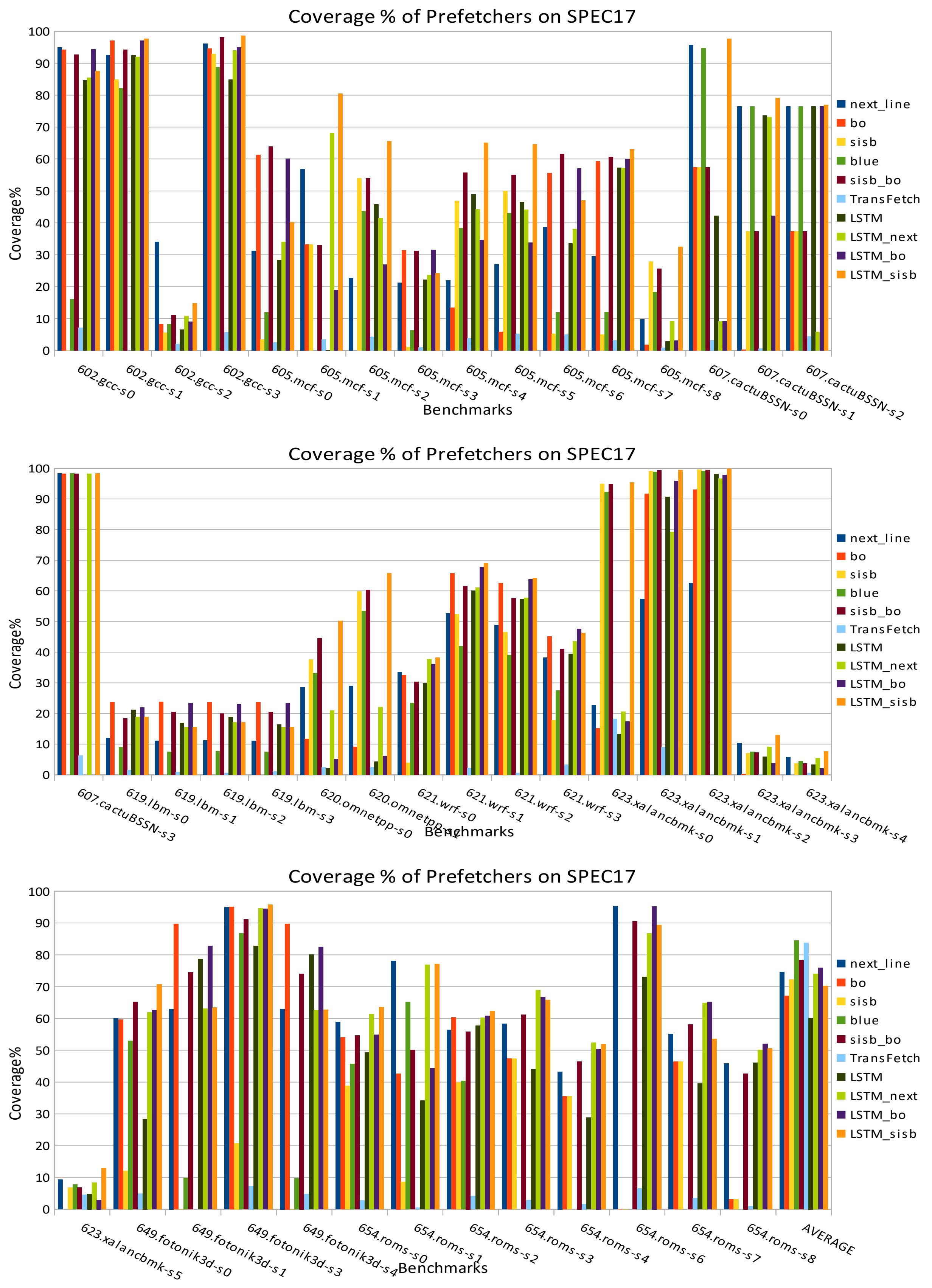
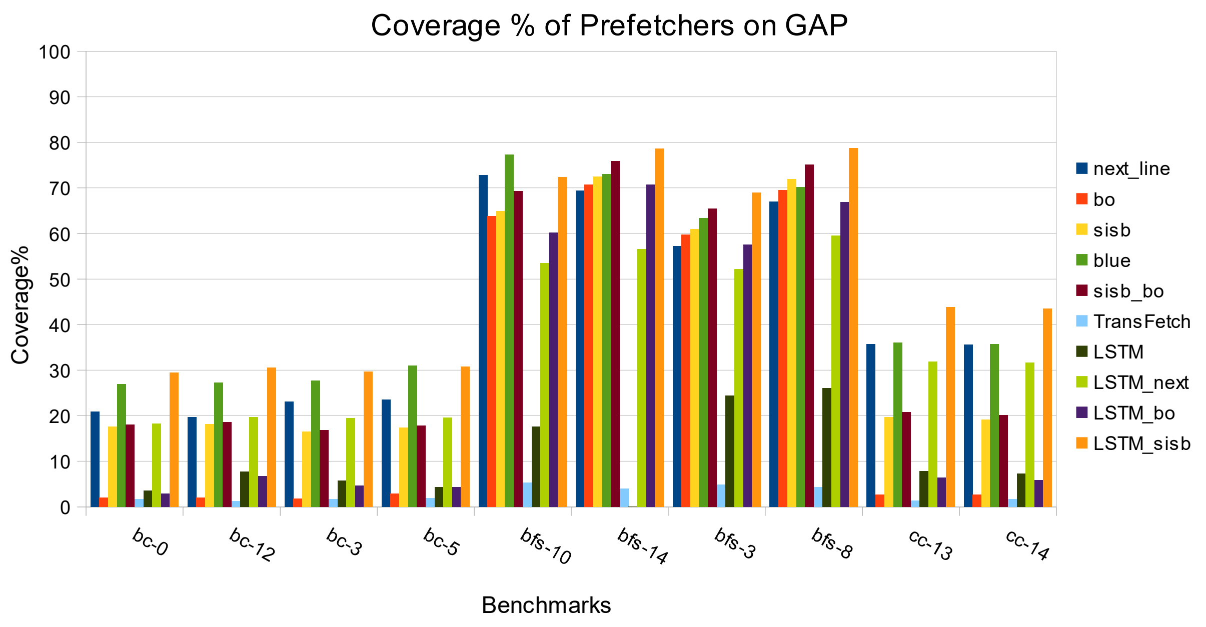


Figure 6.18: Spec17 Coverage

Again here we have a high correlation between coverage and IPC Improvement. The exceptions seem the 605.mcf-s6, -s7, 607.cactuBSSN and some 654.roms where some prefetchers with high coverage do not improve IPC by a lot.

For **GAP** the results are shown in Figure 6.19



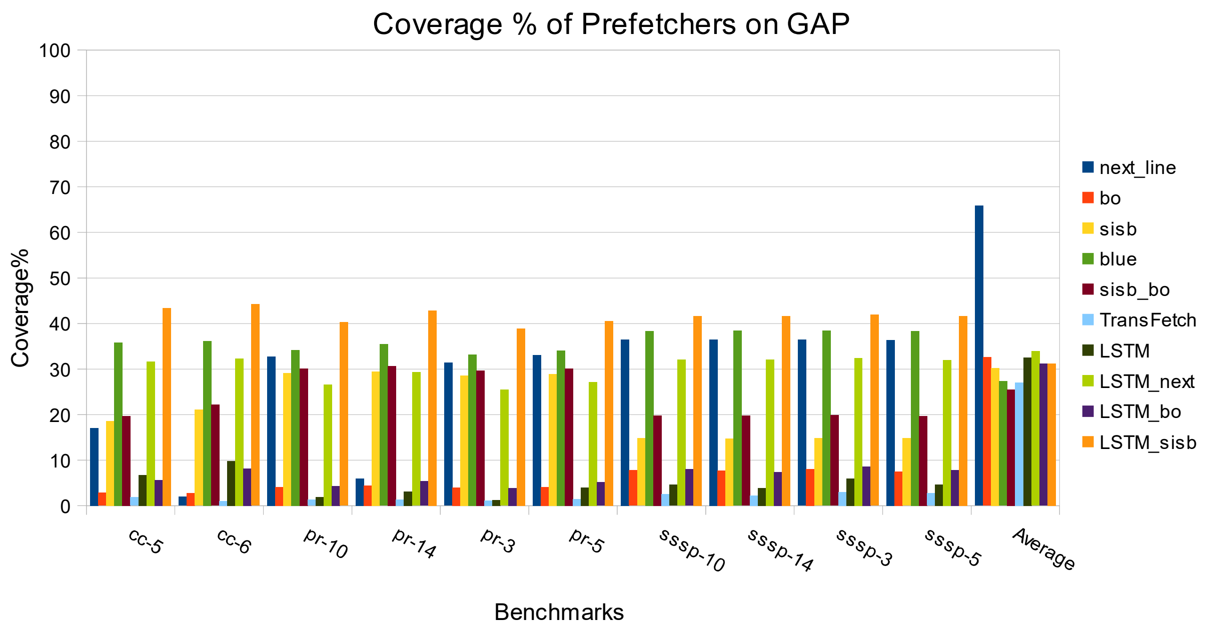


Figure 6.19: GAP Coverage

Here we have similarities in coverage in all benchmarks except the bfs where the profile of all prefetchers is very different.

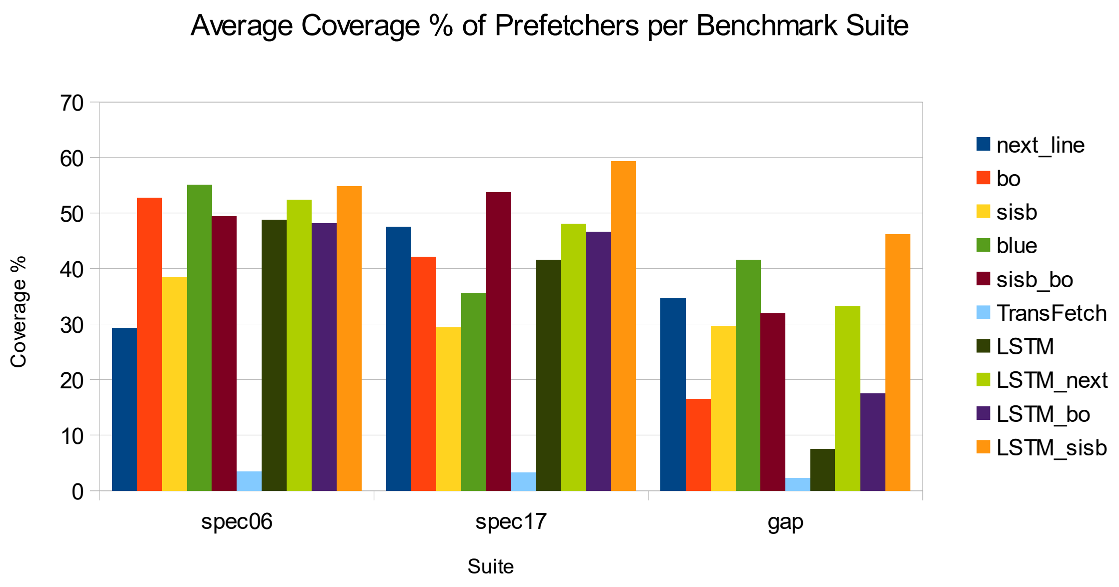
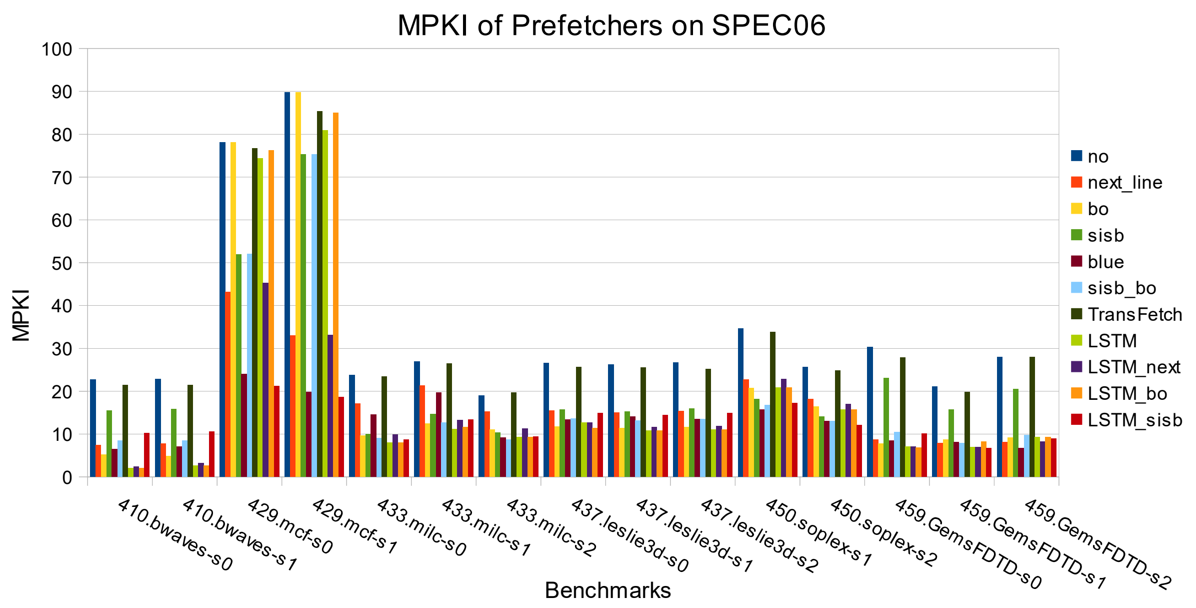


Figure 6.20: Average Coverage % of models per Suite

From the average coverage, we see that the highest average coverage is in LSTM_sisb, blue and BOP with BOP being the winner in performance. We see that LSTM_sisb has one of the better coverage of all prefetchers consistently on all suites with only BLUE being able to be consistent in all benchmarks. Here the issue of TransFetch is clearly shown and explains the lack of performance gains even though the accuracy is higher.

Misses Per Thousand Instructions

For **SPEC06** the Misses Per Thousand(K) Instructions (MPKI)are shown in Figure 6.21



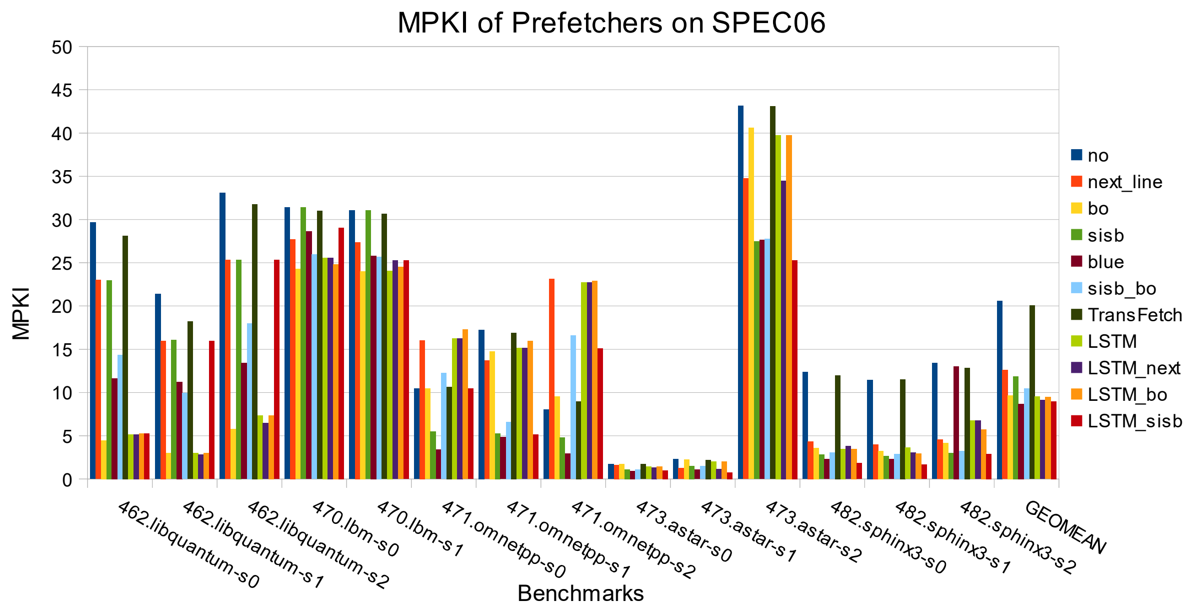
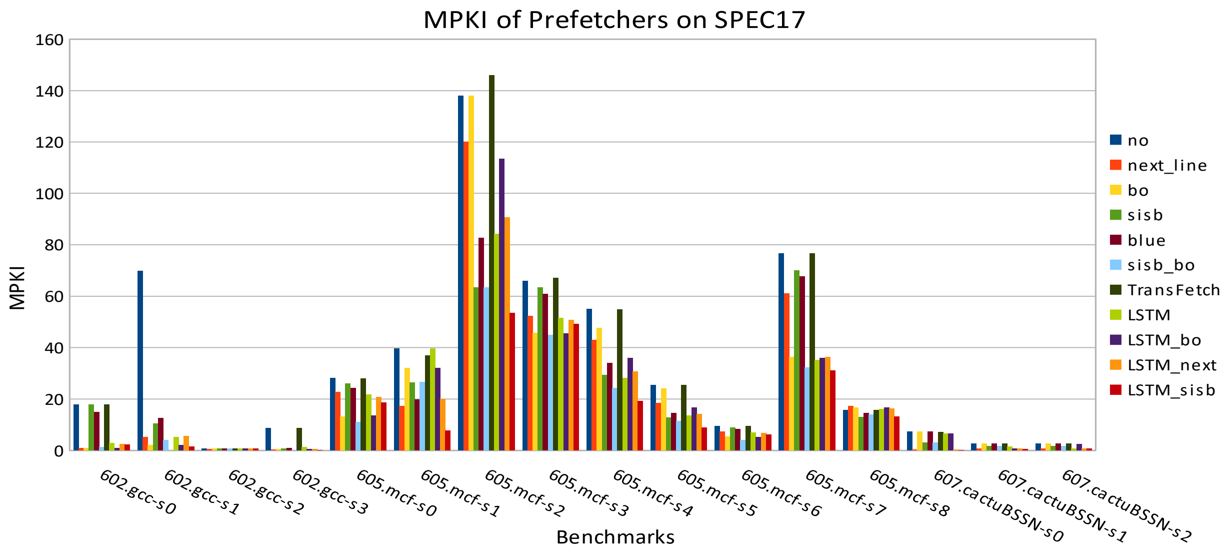


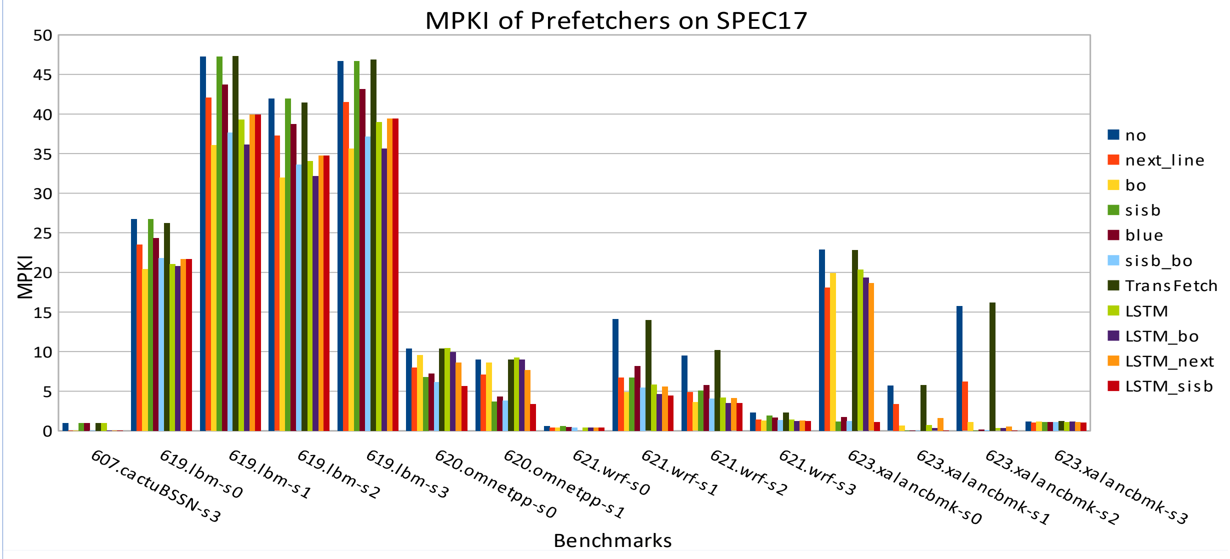
Figure 6.21: Spec06 MPKI

In SPEC06 the number of MPKI is pretty similar to most benchmarks except the 429.mcf where we have a lot more misses than the other benchmarks and in 473.astar where the number of misses is much lower. Except for those two cases, all prefetchers reduce the number of misses either drastically, like in 482.soplex or by a few in 470.lbm. These abnormalities explain the difference in performance gain of each prefetcher because the high miss rate means more difficult addresses, but with greater opportunity for prediction and way lower misses means less margin for improvement by prefetching. Our aggressive approach for example in

429.mcf even with lower accuracy in prefetches results in higher coverage and reduction in the MPKI.

For **SPEC17** the MPKI is shown in Figure 6.22





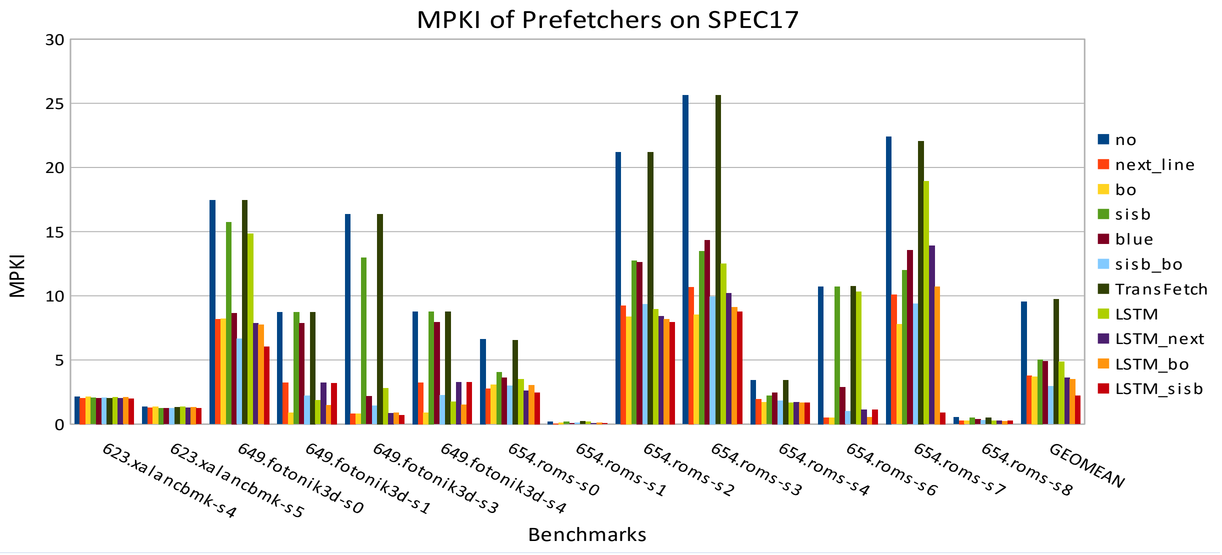
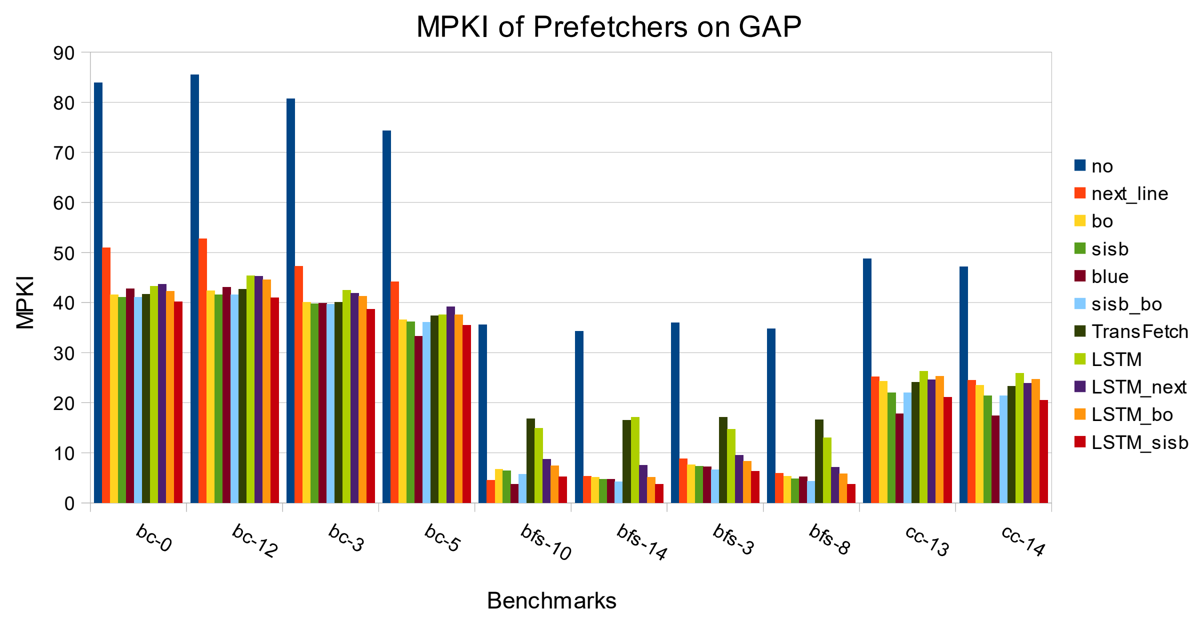


Figure 6.22: Spec17 MPKI

The observations in SPEC17 once more are similar to the SPEC06, with very low MPKI benchmarks having small or almost no performance gain by a prefetcher. Again the 605.mcf has high density of misses but gives the opportunity to our prefetcher to shine. A difference in the MPKI data is in 602.gcc-s1, 523.xalanbmk-s2 and some 654.roms with the 649.fotonik3d where we have a drastic reduction in the misses of the test.

For **GAP** the results are shown in Figure 6.23



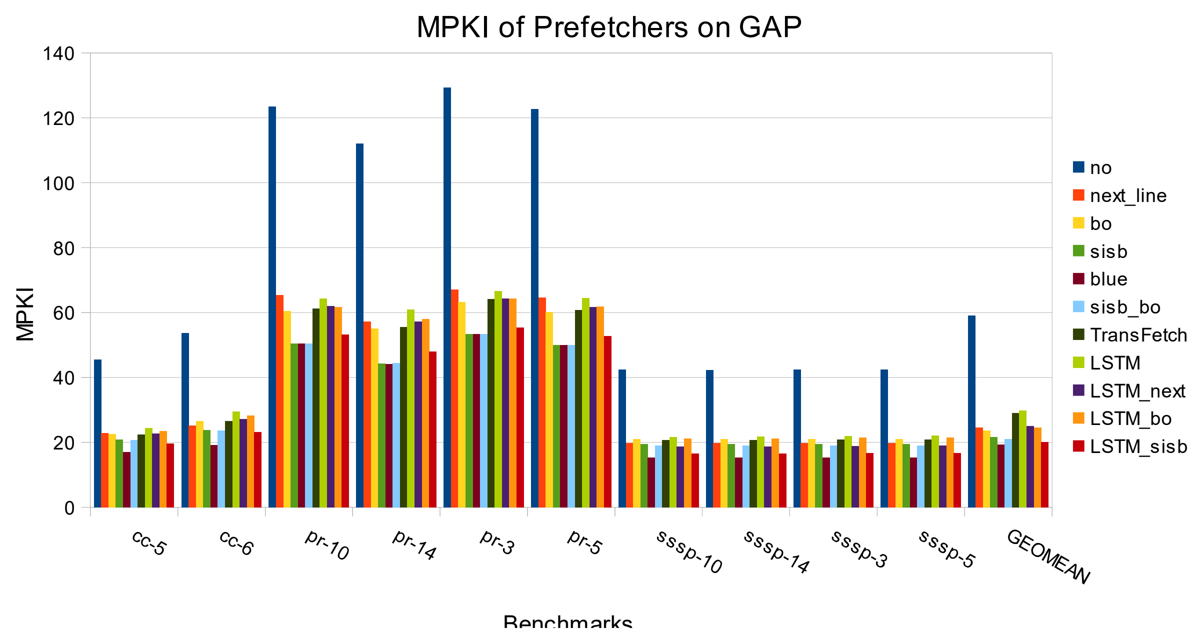


Figure 6.23: GAP MPKI

Finally, GAP has a similar pattern of MPKI with each prefetcher in all benchmarks, with the only difference being the scale.

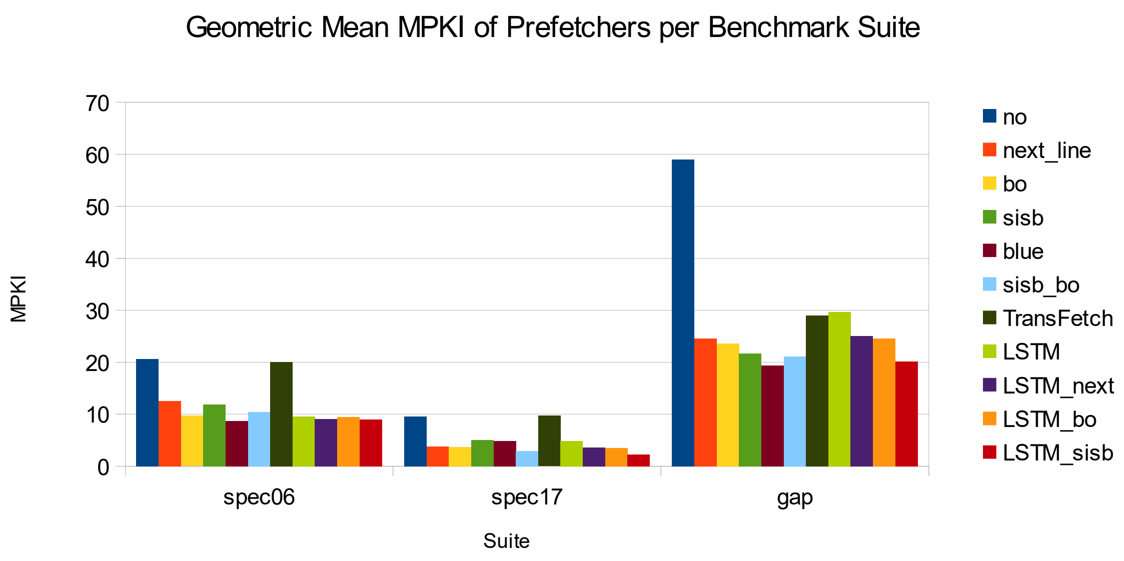


Figure 6.24: Geometric Mean MPKI of Models per Suite

The large difference in misses in GAP from the other suites is obvious in Figure 6.24.

Chapter 7

Conclusions

This study focuses on memory prefetching using a machine learning approach. In this chapter, the conclusion as well as other suggestions for future work are discussed.

7.1 Summary

From the prefetchers that were simulated and explored through experimentation, we draw conclusions on the application of Machine Learning in prefetchers. The usage of Neural networks and other Machine Learning Techniques in prefetching in a simulated and simplified environment provides promising results for their extended use in the future either on their own or complementary to other techniques. Our approach in itself in a general case provides similar or better performance gains with SISB and in some cases outperforms even the state of the art Best Offset Prefetcher. When using our model along with the Best Offset Prefetcher or even better with the SISB Prefetcher on a general case we have better results than all other tested prefetchers. Our exploration showed that for every model the timing of prefetchers and the optimal prefetch distance have a critical impact on performance gains. The drawbacks of the LSTM are that we need an analysis of common address deltas and some knowledge about the timing and delays of memory for a system and application profile. Additionally, our use of Neural Networks and other work that was presented does not consider completely the hardware implementation cost of the prefetcher and the time it will take to make the actual prediction when used online in a real system.

7.2 Future work

Our analysis and results show promising signs for future exploration in Machine Learning for Prefetching in a wider and more feasible context.

Firstly, our prefetching mechanism works in an offline environment with a trace simulator and provided memory traces for training and generation. Adaptations toward online training and generation could provide useful steps for actual implementation. Additionally, it could provide better results for the actual model as some parameters such as the distance of a prefetch and the timing could be dynamically adapted instead of a static value. Furthermore, our traces were provided from a run of an application without a prefetcher on the accesses of Last Level Cache. If there was an introduction of an additional layer of prefetchers in previous Cache Levels it would change our memory stream that arrives in our model and would need a new training process.

Secondly, the LSTM model we implemented is trained with sampled data from the run of the application. The sampling in this work is done offline from the traces that were provided. A study of a sampling method or an adaptation of the model to be trained by the start of the run is necessary for improvement and a feasible online prefetcher.

The delta labels were derived from a statistical analysis of the traces to find the most commonly used to use as categories for the prefetcher. A different representation or encoding of the memory patterns can be explored to see the impact on the model and its complexity.

The Neural Network uses as input features only the memory address deltas and the Program Counters. A change in the features to include the page address and/or a block address impact the complexity and the effectiveness of the model. It has not been tested and experimented with in our work.

Finally, application in virtual address space instead of physical could provide an easier prediction for our Neural Network and possibly better performance. Hence adaptations and experimentation in virtual space are suggested.

Appendix A

Model Result Tables

A.1 SPEC06

Table A.1: Spec 06 Accuracy of Prefetching Models

Trace	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
410.bwaves-s0	99.71	99.45	99.99	100.00	99.64	99.91	99.40	99.45	99.40	99.81
410.bwaves-s1	99.52	98.93	99.99	100.00	99.27	99.94	98.41	98.45	98.41	99.35
429.mcf-s0	64.67	0.00	64.67	72.85	62.77	66.05	35.70	51.12	38.75	59.66
429.mcf-s1	59.65	82.16	59.65	85.39	59.65	47.45	74.11	67.46	76.11	52.56
433.milc-s0	35.31	71.78	99.99	64.92	71.84	45.57	65.61	54.12	65.61	70.03
433.milc-s1	34.91	70.57	99.99	59.63	77.24	97.65	62.63	57.28	66.40	71.28
433.milc-s2	32.73	100.00	99.96	87.89	70.53	94.49	63.77	53.40	63.77	71.23
437.leslie3d-s0	95.47	94.24	94.18	90.22	94.23	98.21	84.25	84.25	85.15	90.16
437.leslie3d-s1	95.70	95.06	94.45	89.12	94.82	97.47	89.13	88.56	89.13	88.87
437.leslie3d-s2	95.05	99.73	94.20	93.09	93.82	96.74	87.62	86.97	87.62	71.62
450.soplex-s1	58.21	61.58	86.01	86.66	70.86	58.39	54.11	51.03	54.11	63.88
450.soplex-s2	53.19	98.92	88.32	87.20	68.46	60.93	53.86	50.67	53.86	56.92
459.GemsFDTD-s0	74.17	82.07	73.42	94.35	81.67	89.60	74.21	74.21	76.03	71.63
459.GemsFDTD-s1	56.02	76.68	71.61	60.13	72.69	59.18	62.58	62.58	70.04	66.51
459.GemsFDTD-s2	67.95	99.79	70.31	78.07	81.42	55.26	68.30	64.55	68.30	72.69
462.libquantum-s0	99.99	99.99	100.00	100.00	99.99	100.00	100.00	100.00	100.00	100.00
462.libquantum-s1	99.99	99.99	100.00	100.00	100.00	100.00	99.99	100.00	99.99	100.00
462.libquantum-s2	99.99	100.00	100.00	100.00	99.99	99.99	99.99	99.99	99.99	100.00
470.lbm-s0	99.82	99.60	0.00	99.98	99.67	99.96	99.31	99.31	99.35	98.46
470.lbm-s1	99.90	99.78	0.00	99.99	99.82	99.96	99.19	99.60	99.62	99.60
471.omnetpp-s0	5.93	0.00	99.90	98.97	27.11	46.75	5.84	5.84	4.10	20.06
471.omnetpp-s1	29.68	22.74	85.55	82.76	50.84	31.03	22.54	22.54	18.87	45.48
471.omnetpp-s2	6.42	99.81	99.43	99.00	30.69	39.87	7.52	7.52	9.28	21.11
473.astar-s0	18.16	0.00	86.70	76.02	86.70	36.58	59.44	44.12	59.44	53.89
473.astar-s1	60.67	75.08	64.06	66.05	64.64	70.56	32.00	57.36	38.02	52.88
473.astar-s2	31.45	99.62	94.29	93.58	67.01	95.84	52.37	43.98	52.37	58.68
482.sphinx3-s0	78.47	78.24	96.37	94.44	85.26	85.64	74.82	75.64	74.82	77.75
482.sphinx3-s1	82.49	81.31	97.30	94.86	87.14	95.13	77.84	80.98	79.87	81.96
482.sphinx3-s2	71.11	70.23	95.28	16.52	80.19	79.63	75.37	75.37	76.04	80.38
AVERAGE	**65.7350945469**	**77.8401711815**	**83.2970444792**	**85.2301948554**	**78.5514895032**	**77.5097975227**	**68.2719772344**	**67.4600847103**	**69.1189245058**	**72.29078968**

Table A.2: Spec 06 Coverage of Prefetching Models

Trace	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
410.bwaves-s0	67.39	77.01	32.00	71.13	62.63	6.04	90.97	89.61	90.97	55.18
410.bwaves-s1	65.85	78.55	30.58	68.10	62.77	5.81	88.45	85.72	88.45	53.83
429.mcf-s0	32.42	0.00	32.42	70.43	32.28	2.23	7.98	46.30	4.20	77.26
429.mcf-s1	11.09	0.01	11.09	77.60	11.09	0.80	13.55	66.50	7.17	74.10
433.milc-s0	28.22	59.33	57.76	45.71	61.78	4.26	66.26	58.28	66.26	63.54
433.milc-s1	20.89	53.91	45.45	34.58	52.83	3.62	58.25	50.59	56.88	50.27
433.milc-s2	0.00	59.22	45.70	51.86	54.12	3.50	51.26	40.52	51.26	50.48
437.leslie3d-s0	41.86	55.89	40.75	48.18	48.90	4.44	52.27	52.27	57.17	43.79
437.leslie3d-s1	43.06	56.89	42.01	49.22	49.90	3.31	58.75	56.09	58.75	45.07
437.leslie3d-s2	0.00	69.55	40.24	49.39	49.11	2.88	58.89	55.72	58.89	44.57
450.soplex-s1	38.53	44.09	48.83	54.60	54.38	2.41	44.49	39.18	44.49	53.47
450.soplex-s2	0.00	70.06	45.35	49.68	51.56	3.01	42.84	38.01	42.84	56.32
459.GemsFDTD-s0	71.58	74.31	23.76	77.86	65.60	8.01	76.91	76.91	77.51	66.31
459.GemsFDTD-s1	62.93	58.61	25.32	67.70	62.30	5.91	67.18	67.18	61.09	68.17
459.GemsFDTD-s2	0.00	90.69	25.32	75.96	62.30	1.02	66.83	70.64	66.83	67.98
462.libquantum-s0	22.46	84.90	22.58	59.05	51.55	4.60	82.55	82.55	82.21	82.21
462.libquantum-s1	25.21	86.06	24.81	58.55	53.42	5.92	85.96	86.78	85.96	25.31
462.libquantum-s2	0.00	80.14	23.27	59.48	45.53	4.25	77.72	80.41	77.72	23.40
470.lbm-s0	11.74	22.66	0.00	12.53	17.29	1.61	18.50	18.50	20.90	7.49
470.lbm-s1	11.81	22.63	0.00	12.73	17.25	0.77	22.43	18.64	20.98	18.65
471.omnetpp-s0	10.85	10.44	47.23	69.36	32.42	4.38	10.46	10.46	3.60	42.38
471.omnetpp-s1	28.64	23.67	69.90	71.41	65.23	2.79	21.55	21.55	17.46	73.69
471.omnetpp-s2	0.00	81.92	41.06	65.15	33.03	3.66	12.79	12.79	8.30	42.12
473.astar-s0	11.33	1.71	35.46	42.14	35.46	0.26	18.35	27.08	18.35	44.99
473.astar-s1	46.10	1.77	34.51	53.35	35.62	2.13	19.52	52.57	15.06	68.28
473.astar-s2	0.00	52.42	36.56	36.17	36.55	0.18	8.61	21.41	8.61	42.39
482.sphinx3-s0	65.27	71.44	77.13	81.17	75.61	4.11	72.33	69.92	72.33	85.44
482.sphinx3-s1	66.71	72.68	77.28	81.69	75.72	3.81	69.86	74.61	75.81	86.20
482.sphinx3-s2	65.97	68.91	77.52	3.10	75.78	3.98	49.68	49.68	57.38	78.33
AVERAGE	**29.3064983451**	**52.7406274568**	**38.4100100639**	**55.0987169006**	**49.3801557535**	**3.4374170364**	**48.8000346488**	**52.4295766747**	**48.1878029872**	**54.8694353489**

Table A.3: Spec 06 MPKI of Prefetching Models

Trace	no	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
410.bwaves-s0	22.80	7.44	5.24	15.51	6.57	8.52	21.44	2.06	2.37	2.06	10.22
410.bwaves-s1	22.83	7.80	4.90	15.85	7.06	8.50	21.51	2.64	3.26	2.64	10.54
429.mcf-s0	78.08	43.15	78.08	51.92	24.00	52.05	76.73	74.38	45.24	76.21	21.27
429.mcf-s1	89.71	33.00	89.71	75.32	19.88	75.32	85.36	80.90	33.13	85.01	18.65
433.milc-s0	23.83	17.11	9.69	10.07	14.53	9.11	23.39	8.04	9.94	8.04	8.69
433.milc-s1	26.92	21.29	12.41	14.68	19.65	12.70	26.50	11.24	13.30	11.61	13.39
433.milc-s2	19.05	15.30	11.14	10.34	9.17	8.74	19.74	9.28	11.33	9.28	9.43
437.leslie3d-s0	26.59	15.47	11.74	15.77	13.40	13.60	25.65	12.73	12.73	11.42	14.97
437.leslie3d-s1	26.30	14.99	11.36	15.27	14.05	13.19	25.52	10.89	11.59	10.89	14.47
437.leslie3d-s2	26.68	15.37	11.62	15.96	13.52	13.60	25.24	11.01	11.85	11.01	14.86
450.soplex-s1	34.65	22.77	20.72	18.22	15.71	16.80	33.77	20.85	22.84	20.85	17.25
450.soplex-s2	25.72	18.14	16.41	14.09	13.03	13.04	24.83	15.78	17.07	15.78	12.06
459.GemsFDTD-s0	30.34	8.65	7.81	23.13	8.48	10.45	27.88	7.02	7.02	6.84	10.13
459.GemsFDTD-s1	21.09	7.84	8.74	15.76	8.12	7.96	19.86	6.94	6.94	8.22	6.73
459.GemsFDTD-s2	28.00	8.11	9.16	20.60	6.77	9.79	28.04	9.36	8.28	9.36	9.02
462.libquantum-s0	29.67	23.01	4.48	22.97	11.65	14.38	28.12	5.18	5.18	5.28	5.28
462.libquantum-s1	21.39	15.99	2.98	16.08	11.24	9.96	18.21	3.00	2.83	3.00	15.97
462.libquantum-s2	33.06	25.35	5.77	25.37	13.40	18.01	31.78	7.37	6.48	7.37	25.33
470.lbm-s0	31.39	27.71	24.28	31.39	28.63	25.97	30.99	25.58	25.58	24.83	29.04
470.lbm-s1	31.05	27.39	24.03	31.05	25.81	25.69	30.63	24.09	25.27	24.54	25.26
471.omnetpp-s0	10.44	16.03	10.44	5.52	3.38	12.26	10.67	16.28	16.28	17.31	10.48
471.omnetpp-s1	17.25	13.70	14.78	5.30	4.84	6.62	16.92	15.17	15.17	15.95	5.12
471.omnetpp-s2	8.03	23.11	9.57	4.78	2.93	16.60	8.98	22.73	22.73	22.89	15.09
473.astar-s0	1.71	1.64	1.71	1.11	0.94	1.11	1.72	1.44	1.34	1.44	1.01
473.astar-s1	2.30	1.28	2.26	1.52	1.10	1.49	2.23	2.01	1.15	2.01	0.78
473.astar-s2	43.16	34.76	40.59	27.48	27.66	27.73	43.07	39.74	34.49	39.74	25.30
482.sphinx3-s0	12.39	4.37	3.59	2.84	2.31	3.06	12.00	3.49	3.79	3.49	1.83
482.sphinx3-s1	11.47	4.01	3.27	2.64	2.29	2.88	11.53	3.65	3.07	2.93	1.67
482.sphinx3-s2	13.41	4.59	4.19	3.01	13.04	3.26	12.85	6.78	6.78	5.74	2.92
GEOMEAN	**20.6042695889**	**12.5841523379**	**9.6965534244**	**11.8749216563**	**8.6925368242**	**10.4757738619**	**20.0566441717**	**9.5277327846**	**9.11860808**	**9.4899797368**	**8.9752128605**

Table A.4: Spec 06 IPC of Prefetching Models

Trace	no	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
410.bwaves-s0	0.627719	0.97124	1.08347	0.769187	1.025	0.966209	0.628766	1.33476	1.31052	1.33476	0.924855
410.bwaves-s1	0.574165	0.927065	1.06754	0.722737	0.963845	0.941938	0.589139	1.23071	1.17727	1.23071	0.883972
429.mcf-s0	0.0754501	0.0867634	0.0754501	0.0867634	0.115426	0.0867653	0.0760755	0.0794506	0.0996377	0.0775442	0.105153
429.mcf-s1	0.0619213	0.067848	0.0619213	0.067848	0.111416	0.067848	0.063891	0.0686353	0.0947505	0.0653003	0.143177
433.milc-s0	0.476603	0.571604	0.711456	0.699589	0.57962	0.757263	0.484252	0.701862	0.63596	0.701862	0.741229
433.milc-s1	0.465946	0.544144	0.707398	0.64247	0.559747	0.751799	0.464749	0.67636	0.621423	0.668727	0.688293
433.milc-s2	0.914237	0.591874	0.997419	0.833651	0.918696	0.914237	0.57036	0.862171	0.787835	0.862171	0.900815
437.leslie3d-s0	0.481619	0.773049	0.868505	0.734298	0.767414	0.818049	0.491632	0.796015	0.796015	0.844885	0.77063
437.leslie3d-s1	0.492603	0.784305	0.880325	0.749525	0.779004	0.830418	0.504146	0.855031	0.823802	0.855031	0.781437
437.leslie3d-s2	0.788483	0.453852	0.984872	0.710037	0.772787	0.788483	0.473674	0.837239	0.800011	0.837239	0.726697
450.soplex-s1	0.303673	0.375018	0.371106	0.434826	0.450168	0.442246	0.306012	0.372929	0.356074	0.372929	0.433993
450.soplex-s2	0.509669	0.356265	0.613058	0.502503	0.504718	0.509669	0.361422	0.431117	0.432203	0.431117	0.509369
459.GemsFDTD-s0	0.343098	0.586333	0.636344	0.361491	0.517346	0.551633	0.359184	0.621316	0.621316	0.624561	0.544527
459.GemsFDTD-s1	0.487524	0.724832	0.68989	0.527662	0.6316	0.703571	0.501033	0.7354	0.7354	0.685265	0.73933
459.GemsFDTD-s2	0.553682	0.370469	0.746385	0.391396	0.605938	0.553682	0.373633	0.558039	0.588746	0.558039	0.56638
462.libquantum-s0	0.508504	0.659507	0.958226	0.656965	0.940633	0.874909	0.526023	0.93992	0.93992	0.95877	0.946037
462.libquantum-s1	0.72206	0.895071	1.08288	0.888421	0.900569	1.00523	0.789819	1.08845	1.08009	1.08845	0.892274
462.libquantum-s2	0.757841	0.481181	0.816084	0.613822	0.70834	0.757841	0.484279	0.847548	0.819017	0.847548	0.617874
470.lbm-s0	0.579287	0.670723	0.729214	0.579287	0.652658	0.708535	0.585157	0.687348	0.687348	0.722921	0.600564
470.lbm-s1	0.585696	0.677485	0.737399	0.585696	0.717075	0.716116	0.590556	0.70487	0.69546	0.731224	0.695467
471.omnetpp-s0	0.543263	0.56643	0.543263	0.583952	0.57436	0.574308	0.540783	0.561531	0.561531	0.522418	0.572267
471.omnetpp-s1	0.309453	0.337843	0.325689	0.447871	0.466984	0.428847	0.311482	0.324485	0.324485	0.315029	0.441943
471.omnetpp-s2	0.438614	0.438666	0.464339	0.455178	0.459047	0.438614	0.434534	0.429668	0.429668	0.41121	0.438021
473.astar-s0	0.971072	0.981	0.971072	1.04421	1.06682	1.04421	0.971385	1.0108	1.02544	1.0108	1.06298
473.astar-s1	0.448605	0.481246	0.449796	0.470003	0.468115	0.470914	0.462947	0.452812	0.482855	0.454089	0.49354
473.astar-s2	0.17735	0.130815	0.214611	0.179675	0.177811	0.17735	0.130727	0.135339	0.148722	0.135339	0.187261
482.sphinx3-s0	0.665671	1.07174	1.10747	1.19992	1.24864	1.18046	0.676209	1.07416	1.05329	1.07416	1.24871
482.sphinx3-s1	0.69874	1.14996	1.17467	1.26713	1.23651	1.24483	0.72012	1.06015	1.15654	1.1676	1.32884
482.sphinx3-s2	0.59185	0.954801	0.962273	1.07593	0.591198	1.04801	0.599737	0.822737	0.822737	0.880496	1.08265
GEOMEAN	**0.4568214358**	**0.5152333879**	**0.6124512412**	**0.5388532098**	**0.5894768557**	**0.5992289652**	**0.4257763222**	**0.5834198792**	**0.5899890861**	**0.5841703618**	**0.6076263678**

Table A.5: Spec 06 IPC Improvement % of Prefetching Models

Trace	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
410.bwaves-s0	54.73	72.60	22.54	63.29	53.92	0.17	112.64	108.77	112.64	47.34
410.bwaves-s1	61.46	85.93	25.88	67.87	64.05	2.61	114.35	105.04	114.35	53.96
429.mcf-s0	14.99	0.00	14.99	52.98	15.00	0.83	5.30	32.06	2.78	69.82
429.mcf-s1	9.57	0.01	9.57	79.93	9.57	0.50	10.84	53.02	5.46	47.12
433.milc-s0	19.93	49.28	46.79	21.61	58.89	1.60	47.26	33.44	47.26	55.52
433.milc-s1	16.78	51.82	37.89	20.13	61.35	-0.26	45.16	33.37	43.52	47.72
433.milc-s2	21.10	68.52	40.85	55.22	54.46	-3.63	45.67	33.11	45.67	52.20
437.leslie3d-s0	60.51	80.33	52.46	59.34	69.85	2.08	65.28	65.28	75.43	60.01
437.leslie3d-s1	59.22	78.71	52.16	58.14	68.58	2.34	73.57	67.23	73.57	58.63
437.leslie3d-s2	66.13	117.00	56.45	70.27	73.73	4.37	84.47	76.27	84.47	60.12
450.soplex-s1	23.49	22.21	43.19	48.24	45.63	0.77	22.81	17.26	22.81	42.91
450.soplex-s2	21.28	72.08	41.05	41.67	43.06	1.45	21.01	21.32	21.01	42.97
459.GemsFDTD-s0	70.89	85.47	5.36	50.79	60.78	4.69	81.09	81.09	82.04	58.71
459.GemsFDTD-s1	48.68	41.51	8.23	29.55	44.32	2.77	50.84	50.84	40.56	51.65
459.GemsFDTD-s2	63.89	101.47	5.65	63.56	49.45	0.85	50.63	58.92	50.63	52.88
462.libquantum-s0	29.70	88.44	29.20	84.98	72.06	3.45	84.84	84.84	88.55	86.04
462.libquantum-s1	23.96	49.97	23.04	24.72	39.22	9.38	50.74	49.58	50.74	23.57
462.libquantum-s2	29.25	69.60	27.57	47.21	57.50	0.64	76.14	70.21	76.14	28.41
470.lbm-s0	15.78	25.88	0.00	12.67	22.31	1.01	18.65	18.65	24.79	3.67
470.lbm-s1	15.67	25.90	0.00	22.43	22.27	0.83	20.35	18.74	24.85	18.74
471.omnetpp-s0	4.26	0.00	7.49	5.72	5.71	-0.46	3.36	3.36	-3.84	5.34
471.omnetpp-s1	9.17	5.25	44.73	50.91	38.58	0.66	4.86	4.86	1.80	42.81
471.omnetpp-s2	-1.10	5.85	3.76	4.65	-0.01	-0.94	-2.05	-2.05	-6.26	-0.15
473.astar-s0	1.02	0.00	7.53	9.86	7.53	0.03	4.09	5.60	4.09	9.46
473.astar-s1	7.28	0.27	4.77	4.35	4.97	3.20	0.94	7.63	1.22	10.02
473.astar-s2	14.93	64.06	37.35	35.93	35.57	-0.07	3.46	13.69	3.46	43.15
482.sphinx3-s0	61.00	66.37	80.26	87.58	77.33	1.58	61.36	58.23	61.36	87.59
482.sphinx3-s1	64.58	68.11	81.34	76.96	78.15	3.06	51.72	65.52	67.10	90.18
482.sphinx3-s2	61.32	62.59	81.79	-0.11	43.06	1.33	39.01	39.01	48.77	82.93
AVERAGE	**32.7408818783**	**50.317728399**	**30.754356593**	**43.1189300053**	**44.0312829038**	**1.5463819047**	**43.0484653702**	**43.9617707647**	**43.6196427582**	**45.9766655003**

A.2 SPEC17

Table A.6: Spec 17 Accuracy of Prefetching Models

Trace	next_line	bo	sisb	sisb_bo	blue	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
602.gcc-s0	99.20	98.54	26.56	99.08	97.59	99.27	98.50	99.05	98.71	98.99
602.gcc-s1	98.59	97.41	99.98	98.46	99.91		97.31	98.24	97.86	98.32
602.gcc-s2	49.73	22.39	67.96	35.14	39.97	44.99	23.39	34.32	23.04	35.55
602.gcc-s3	95.76	94.04	99.59	96.65	99.44	95.66	90.48	94.65	93.82	94.61
605.mcf-s0	41.39	36.99	35.10	47.70	65.10	97.42	43.94	45.07	38.71	40.07
605.mcf-s1	83.25	46.21	70.94	62.08	82.92	94.03	15.81	85.23	46.23	82.74
605.mcf-s2	26.66	0.00	100.00	100.00	99.76	95.48	52.16	42.84	49.98	54.17
605.mcf-s3	66.32	64.75	52.49	72.70	74.61	84.40	68.68	69.38	64.24	66.87
605.mcf-s4	30.88	90.06	99.85	97.74	96.94	91.71	58.24	50.14	61.60	59.58
605.mcf-s5	34.22	46.84	99.81	88.94	88.27	81.20	52.56	47.12	55.17	56.58
605.mcf-s6	39.42	34.80	30.47	45.46	53.41	90.94	41.04	41.41	36.63	36.66
605.mcf-s7	38.47	38.07	32.90	47.78	51.57	98.49	63.46	58.37	40.55	48.67
605.mcf-s8	8.66	11.07	38.31	29.06	28.38	13.25	10.38	12.72	10.67	22.66
607.cactuBSSN-s0	99.78	0.00	99.77	99.77	0.00	95.73	64.02	94.77	64.02	94.66
607.cactuBSSN-s1	99.75	94.43	99.74	99.74	0.00	99.25	64.86	99.71	99.69	81.80
607.cactuBSSN-s2	99.78	0.00	99.69	99.69	0.00	97.69	98.19	98.19	81.50	98.06
607.cactuBSSN-s3	99.94	99.96	0.00	99.98	0.00	99.19	0.00	99.98	99.98	99.98
619.lbm-s0	99.99	99.99	0.00	99.99	99.99	99.99	100.00	100.00	99.99	100.00
619.lbm-s1	99.99	99.98	0.00	99.99	99.99	99.98	99.99	99.99	99.99	99.99
619.lbm-s2	99.99	99.98	0.00	99.99	99.99	99.99	99.99	99.99	99.99	99.99
619.lbm-s3	99.98	99.98	0.00	99.99	99.98	99.97	99.99	99.99	99.98	99.99
620.omnetpp-s0	25.55	20.10	37.00	33.79	44.17	32.61	7.42	27.83	16.51	29.39
620.omnetpp-s1	26.13	20.04	70.39	49.29	72.05	30.93	7.71	25.11	12.79	40.96
621.wrf-s0	93.75	92.29	97.86	94.45	93.49		85.95	91.94	89.95	91.93
621.wrf-s1	87.80	89.32	92.62	90.20	94.76	87.51	73.82	80.92	82.49	80.76
621.wrf-s2	87.14	90.13	91.67	90.23	93.57	92.77	78.57	83.92	85.17	82.96
621.wrf-s3	80.96	88.61	87.44	90.70	90.09	95.04	68.74	77.42	78.79	76.83
623.xalancbmk-s0	20.76	18.98	99.69	54.14	99.23	40.52	8.48	14.98	13.53	59.01
623.xalancbmk-s1	86.75	90.94	100.00	93.20	99.81	99.18	90.71	89.82	91.33	90.22
623.xalancbmk-s2	89.19	95.03	99.97	94.91	99.86	99.46	96.26	92.58	94.11	92.79
623.xalancbmk-s3	61.00	53.13	91.79	90.18	78.69	75.76	45.32	60.80	49.72	66.43
623.xalancbmk-s4	46.00	0.00	86.63	86.63	68.76	41.72	25.03	41.55	29.65	47.75
623.xalancbmk-s5	43.61	0.00	90.10	90.10	73.76	43.44	21.32	37.12	24.48	45.99
649.fotonik3d-s0	44.59	47.04	39.18	58.61	63.42	70.36	22.48	41.79	42.46	42.62
649.fotonik3d-s1	99.91	99.88	54.44	99.91	99.59	99.83	99.80	99.79	99.77	99.75
649.fotonik3d-s3	94.03	93.90	99.99	96.55	97.70	93.98	79.91	92.76	92.69	92.84
649.fotonik3d-s4	99.92	99.90	62.29	99.93	99.76	99.92	99.92	99.91	99.90	99.89
654.roms-s0	95.12	93.73	99.74	96.46	98.40	94.03	69.69	87.01	85.88	87.23
654.roms-s1	93.68	94.13	97.49	96.95	95.17	56.98	56.90	84.37	77.15	83.96
654.roms-s2	96.07	94.76	99.99	96.98	98.60	96.17	70.23	91.11	91.12	91.39
654.roms-s3	95.27	93.03	99.57	95.72	97.91	93.61	61.22	78.42	79.09	79.74
654.roms-s4	95.14	90.87	99.97	94.31	98.54	95.41	62.19	86.80	86.85	87.31
654.roms-s6	94.72	94.67	100.00	97.12	96.04	96.39	15.26	87.41	91.21	87.41
654.roms-s7	95.48	92.92	99.94	95.61	97.56	95.24	42.28	96.39	94.96	97.40
654.roms-s8	96.26	92.55	100.00	95.00	96.51	90.53	74.47	90.56	90.56	90.62
AVERAGE	**74.68**	**67.14**	**72.24**	**84.46**	**78.34**	**83.72**	**60.15**	**74.03**	**70.28**	**75.89**

Table A.7: Spec Coverage of Prefetching Models

Trace	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
602.gcc-s0	95.02	94.23	0.01	16.06	92.74	7.21	84.62	85.51	94.37	87.72
602.gcc-s1	92.62	97.10	84.96	82.14	94.24	0.00	92.52	92.07	97.12	97.70
602.gcc-s2	34.03	8.39	5.57	8.36	11.17	2.07	6.62	10.78	9.06	14.89
602.gcc-s3	96.19	94.59	92.89	88.87	98.19	5.81	84.87	94.10	95.00	98.72
605.mcf-s0	31.24	61.36	3.50	12.01	63.93	2.50	28.44	34.03	60.18	40.17
605.mcf-s1	56.87	33.32	33.32	0.00	33.04	3.55	0.01	68.08	19.06	80.55
605.mcf-s2	22.68	0.00	54.02	43.65	54.02	4.33	45.75	41.57	26.97	65.56
605.mcf-s3	21.23	31.40	1.17	6.29	31.25	1.03	22.19	23.63	31.56	24.26
605.mcf-s4	22.00	13.40	46.84	38.28	55.77	3.82	48.98	44.33	34.66	65.11
605.mcf-s5	27.07	5.83	50.04	43.04	55.03	5.28	46.57	44.09	33.84	64.59
605.mcf-s6	38.67	55.62	5.28	12.07	61.58	5.04	33.58	38.12	57.09	47.13
605.mcf-s7	29.62	59.34	5.10	12.18	60.62	3.29	57.33	57.14	59.95	63.09
605.mcf-s8	9.81	1.89	27.95	18.33	25.70	0.87	2.88	9.35	3.15	32.57
607.cactuBSSN-s0	95.74	57.42	57.42	94.71	57.42	3.23	42.20	9.14	9.14	97.73
607.cactuBSSN-s1	76.46	0.32	37.39	76.43	37.39	0.60	73.62	73.22	42.20	79.13
607.cactuBSSN-s2	76.43	37.36	37.36	76.47	37.36	4.38	76.47	5.84	76.47	76.97
607.cactuBSSN-s3	98.39	98.21	0.00	98.36	98.20	6.32	0.00	98.21	0.00	98.36
619.lbm-s0	11.93	23.64	0.00	9.00	18.44	1.57	21.22	18.85	22.02	18.85
619.lbm-s1	11.06	23.77	0.00	7.49	20.40	0.96	16.86	15.56	23.49	15.57
619.lbm-s2	11.22	23.72	0.00	7.73	19.97	0.53	18.88	17.12	23.15	17.14
619.lbm-s3	11.07	23.68	0.00	7.54	20.42	1.06	16.46	15.52	23.50	15.53
620.omnetpp-s0	28.70	11.68	37.66	33.13	44.52	2.44	2.07	20.98	5.10	50.26
620.omnetpp-s1	28.94	9.19	59.82	53.39	60.36	2.44	4.26	22.06	6.17	65.75
621.wrf-s0	33.57	32.63	4.00	23.48	30.31	0.00	29.84	37.78	36.20	38.21
621.wrf-s1	52.70	65.75	52.29	41.98	61.64	2.26	60.16	61.09	67.74	69.09
621.wrf-s2	48.79	62.55	46.56	39.10	57.69	0.61	57.29	57.73	63.76	64.14
621.wrf-s3	38.29	45.16	17.72	27.56	41.06	3.24	39.49	43.56	47.67	46.30
623.xalancbmk-s0	22.67	15.15	94.87	92.29	94.80	18.27	13.39	20.55	17.34	95.37
623.xalancbmk-s1	57.43	91.68	99.06	98.89	99.26	9.03	90.75	79.21	95.88	99.43
623.xalancbmk-s2	62.56	93.08	99.61	99.09	99.49	0.00	98.10	96.66	97.86	99.86
623.xalancbmk-s3	10.30	0.17	7.01	7.44	7.28	0.14	5.86	9.09	3.75	12.96
623.xalancbmk-s4	5.74	0.00	3.66	4.40	3.66	0.61	3.27	5.48	2.09	7.62
623.xalancbmk-s5	9.39	0.00	6.84	7.81	6.84	4.64	4.88	8.36	2.94	12.89
649.fotonik3d-s0	59.94	59.67	12.08	53.04	65.20	4.93	28.30	61.92	62.54	70.68
649.fotonik3d-s1	63.02	89.74	0.04	9.80	74.45	0.00	78.65	63.11	82.81	63.44
649.fotonik3d-s3	94.98	95.01	20.80	86.74	91.20	7.24	82.84	94.72	94.43	95.82
649.fotonik3d-s4	63.03	89.81	0.04	9.64	74.08	4.87	80.12	62.63	82.44	62.76
654.roms-s0	58.92	54.07	38.83	45.73	54.59	2.78	49.23	61.45	54.90	63.56
654.roms-s1	78.06	42.68	8.62	65.20	50.14	0.52	34.17	76.88	44.29	77.11
654.roms-s2	56.37	60.40	39.84	40.39	55.87	4.22	57.67	60.24	60.88	62.42
654.roms-s3	58.33	47.40	47.40	0.00	61.23	2.87	44.04	68.87	66.75	65.86
654.roms-s4	43.28	35.49	35.49	0.00	46.47	1.67	28.79	52.42	50.41	51.89
654.roms-s6	95.28	0.00	0.00	0.00	90.61	6.63	73.11	86.77	95.26	89.35
654.roms-s7	55.06	46.41	46.41	0.00	58.03	3.56	39.55	64.88	65.25	53.60
654.roms-s8	45.79	3.15	3.15	0.00	42.68	1.03	46.07	50.19	52.03	50.61
AVERAGE	**47.57**	**42.12**	**29.44**	**35.51**	**53.74**	**3.28**	**41.60**	**48.06**	**46.68**	**59.34**

Table A.8: Spec MPKI of Prefetching Models

Trace	no	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
602.gcc-s0	17.776	0.886	1.025	17.770	14.918	1.291	17.777	2.733	2.575	0.980	2.183
602.gcc-s1	69.873	5.153	2.028	10.511	12.479	4.027		5.224	5.540	1.970	1.607
602.gcc-s2	0.625	0.498	0.660	0.598	0.613	0.613	0.668	0.662	0.637	0.675	0.620
602.gcc-s3	8.628	0.329	0.467	0.614	0.960	0.156	8.630	1.306	0.510	0.428	0.111
605.mcf-s0	28.213	22.631	13.138	25.989	24.384	10.962	28.008	21.724	20.796	13.523	18.563
605.mcf-s1	39.677	17.352	32.115	26.463	19.874	26.574	37.034	39.687	20.107	32.116	7.805
605.mcf-s2	137.913	120.093	137.913	63.419	82.617	63.419	145.960	84.231	90.747	113.363	53.488
605.mcf-s3	65.855	52.327	45.737	63.491	60.980	44.799	67.170	51.569	50.796	45.559	49.093
605.mcf-s4	55.131	43.010	47.744	29.305	34.029	24.386	54.777	28.129	30.697	35.991	19.239
605.mcf-s5	25.433	18.553	23.951	12.706	14.486	11.438	25.461	13.592	14.224	16.771	9.008
605.mcf-s6	9.463	7.264	5.372	8.780	8.311	4.101	9.571	6.971	6.844	5.181	6.164
605.mcf-s7	76.696	61.102	36.329	70.082	67.625	32.332	76.685	35.161	36.359	35.948	31.135
605.mcf-s8	15.811	17.323	16.577	12.978	14.485	13.911	15.714	16.129	16.403	16.675	13.155
607.cactuBSSN-s0	7.257	0.309	7.257	3.090	7.257	3.090	7.169	6.597	0.384	6.597	0.165
607.cactuBSSN-s1	2.643	0.622	2.634	1.654	2.643	1.654	2.644	1.530	0.623	0.708	0.552
607.cactuBSSN-s2	2.645	0.623	2.645	1.656	2.645	1.656	2.657	0.622	0.622	2.491	0.609
607.cactuBSSN-s3	0.965	0.016	0.017	0.965	0.965	0.017	0.964	0.965	0.016	0.017	0.016
619.lbm-s0	26.748	23.558	20.424	26.748	24.341	21.815	26.267	21.071	21.706	20.800	21.705
619.lbm-s1	47.290	42.059	36.052	47.290	43.750	37.645	47.300	39.316	39.933	36.131	39.927
619.lbm-s2	41.957	37.251	32.005	41.957	38.714	33.579	41.427	34.034	34.774	32.202	34.768
619.lbm-s3	46.698	41.527	35.641	46.698	43.176	37.163	46.903	39.011	39.452	35.663	39.444
620.omnetpp-s0	10.413	7.981	9.611	6.776	7.201	6.183	10.408	10.472	8.640	9.953	5.639
620.omnetpp-s1	9.014	7.099	8.646	3.721	4.328	3.851	9.029	9.273	7.661	8.993	3.404
621.wrf-s0	0.628	0.418	0.424	0.603	0.480	0.438		0.441	0.391	0.412	0.388
621.wrf-s1	14.120	6.723	4.888	6.754	8.212	5.456	13.974	5.824	5.618	4.653	4.467
621.wrf-s2	9.508	4.906	3.613	5.099	5.810	4.061	10.197	4.232	4.113	3.511	3.500
621.wrf-s3	2.330	1.442	1.282	1.918	1.689	1.375	2.351	1.424	1.323	1.239	1.259
623.xalancbmk-s0	22.866	18.115	19.913	1.173	1.765	1.219	22.837	20.343	18.631	19.365	1.085
623.xalancbmk-s1	5.682	3.367	0.657	0.053	0.062	0.057	5.782	0.725	1.640	0.325	0.045
623.xalancbmk-s2	15.754	6.207	1.119	0.062	0.144	0.082	16.208	0.306	0.552	0.349	0.024
623.xalancbmk-s3	1.170	1.073	1.168	1.089	1.093	1.087	1.203	1.128	1.088	1.143	1.043
623.xalancbmk-s4	2.134	2.042	2.134	2.057	2.044	2.057	2.040	2.099	2.046	2.108	2.000
623.xalancbmk-s5	1.346	1.278	1.346	1.256	1.248	1.256	1.318	1.346	1.291	1.339	1.228
649.fotonik3d-s0	17.434	8.186	8.241	15.744	8.667	6.656	17.432	14.855	7.850	7.745	6.055
649.fotonik3d-s1	8.714	3.222	0.894	8.711	7.877	2.227	8.737	1.864	3.222	1.465	3.192
649.fotonik3d-s3	16.373	0.823	0.816	12.968	2.170	1.441	16.373	2.809	0.864	0.889	0.685
649.fotonik3d-s4	8.773	3.244	0.894	8.769	7.928	2.274	8.778	1.744	3.278	1.506	3.267
654.roms-s0	6.642	2.746	3.062	4.064	3.619	3.022	6.559	3.516	2.612	3.045	2.468
654.roms-s1	0.209	0.047	0.122	0.191	0.077	0.105	0.220	0.176	0.056	0.134	0.055
654.roms-s2	21.180	9.241	8.387	12.741	12.624	9.347	21.208	8.966	8.421	8.204	7.960
654.roms-s3	25.640	10.685	8.524	13.488	14.347	9.942	25.621	12.502	10.196	9.123	8.753
654.roms-s4	3.448	1.956	1.710	2.224	2.455	1.846	3.449	1.687	1.735	1.677	1.659
654.roms-s6	10.730	0.506	0.508	10.730	2.885	1.007	10.735	10.337	1.144	0.562	1.143
654.roms-s7	22.403	10.068	7.785	12.006	13.544	9.404	22.076	18.917	13.899	10.716	0.898
654.roms-s8	0.524	0.284	0.257	0.508	0.379	0.301	0.504	0.283	0.261	0.252	0.259
GEOMEAN	**9.541**	**3.781**	**3.717**	**5.032**	**4.893**	**2.941**	**9.726**	**4.872**	**3.602**	**3.486**	**2.212**

Table A.9: Spec 17 IPC of Prefetching Models

Trace	no	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
602.gcc-s0	0.346953	0.593622	0.582907	0.347038	0.374877	0.57686	0.346936	0.544781	0.572015	0.584675	0.578192
602.gcc-s1	0.108285	0.229326	0.237433	0.212636	0.203717	0.235443		0.22125	0.229863	0.237161	0.241029
602.gcc-s2	0.708308	0.718212	0.706925	0.709703	0.709715	0.709571	0.704027	0.707715	0.708779	0.70713	0.709688
602.gcc-s3	0.560882	0.792024	0.784683	0.784852	0.774811	0.803533	0.560214	0.744283	0.78726	0.786185	0.805261
605.mcf-s0	0.307714	0.300472	0.292817	0.323053	0.321359	0.330729	0.308517	0.295368	0.303333	0.297095	0.314421
605.mcf-s1	0.155993	0.251981	0.156307	0.15472	0.209321	0.155084	0.154377	0.156019	0.254282	0.15632	0.249759
605.mcf-s2	0.159445	0.156243	0.159445	0.305708	0.232757	0.305708	0.153327	0.193337	0.185122	0.175657	0.279704
605.mcf-s3	0.162544	0.187359	0.187773	0.168244	0.170684	0.198665	0.159889	0.178572	0.185103	0.188362	0.195074
605.mcf-s4	0.284014	0.326016	0.293826	0.445648	0.403729	0.467368	0.284855	0.369521	0.366599	0.34572	0.48339
605.mcf-s5	0.350759	0.425055	0.374009	0.484586	0.49255	0.53927	0.350329	0.454277	0.47147	0.44103	0.576352
605.mcf-s6	0.421035	0.41951	0.41615	0.42782	0.429044	0.448513	0.419911	0.419623	0.423782	0.420823	0.432807
605.mcf-s7	0.162732	0.162816	0.155581	0.171377	0.167683	0.177352	0.1625	0.176902	0.172513	0.157171	0.177413
605.mcf-s8	0.301439	0.293754	0.297511	0.315716	0.308253	0.311456	0.302544	0.301307	0.300302	0.297553	0.312794
607.cactuBSSN-s0	1.18812	1.71096	1.18812	1.24528	1.18742	1.24528	1.18758	1.18796	1.8167	1.18796	1.81073
607.cactuBSSN-s1	1.16871	1.36756	1.1693	1.22677	1.16823	1.22677	1.16889	1.24833	1.36986	1.35893	1.37725
607.cactuBSSN-s2	1.19513	1.40427	1.19513	1.25611	1.19411	1.25611	1.19413	1.40534	1.40534	1.20759	1.40734
607.cactuBSSN-s3	1.94516	2.03005	2.02979	1.94516	1.94507	2.02977	1.94627	1.94507	2.02991	2.02977	2.02991
619.lbm-s0	0.630792	0.769476	0.837771	0.630792	0.671484	0.809107	0.641061	0.769091	0.770614	0.823158	0.770799
619.lbm-s1	0.401126	0.449437	0.479506	0.401126	0.419354	0.470568	0.402145	0.42893	0.43369	0.474749	0.433705
619.lbm-s2	0.44375	0.504177	0.539648	0.44375	0.465065	0.527988	0.449341	0.494106	0.496191	0.534902	0.496319
619.lbm-s3	0.405232	0.455077	0.485606	0.405232	0.424227	0.47653	0.404978	0.430719	0.437945	0.48054	0.437922
620.omnetpp-s0	0.33866	0.356105	0.34381	0.373023	0.370803	0.380067	0.338889	0.337846	0.353409	0.34129	0.383395
620.omnetpp-s1	0.35927	0.377275	0.362657	0.428745	0.422006	0.428766	0.359364	0.357115	0.37392	0.359515	0.432346
621.wrf-s0	1.19376	1.21703	1.21431	1.19603	1.20983	1.21344		1.20936	1.21645	1.21402	1.21654
621.wrf-s1	0.667492	1.0705	1.14747	1.00981	0.922028	1.12514	0.66876	0.957364	1.05752	1.11666	1.13659
621.wrf-s2	0.837928	1.24635	1.31931	1.15779	1.08636	1.29974	0.811737	1.15778	1.24546	1.29764	1.30805
621.wrf-s3	1.08062	1.15573	1.16726	1.12425	1.12864	1.16291	1.08156	1.14213	1.15653	1.16573	1.16291
623.xalancbmk-s0	0.317432	0.369245	0.346165	0.529325	0.519401	0.529868	0.317653	0.325209	0.35273	0.343657	0.530813
623.xalancbmk-s1	0.85296	0.977182	1.07017	1.09741	1.11681	1.09758	0.859661	1.07398	1.04261	1.10219	1.11773
623.xalancbmk-s2	0.541822	0.737069	0.858151	0.891483	0.890549	0.892453	0.534814	0.876416	0.872463	0.877661	0.893796
623.xalancbmk-s3	0.806889	0.809417	0.806904	0.811756	0.811868	0.811747	0.80522	0.809823	0.81068	0.8091	0.813343
623.xalancbmk-s4	0.733173	0.736088	0.733173	0.737497	0.733545	0.737497	0.730357	0.73173	0.732799	0.730931	0.735185
623.xalancbmk-s5	0.685018	0.68654	0.685018	0.689209	0.689638	0.689209	0.687371	0.68622	0.687549	0.686273	0.690151
649.fotonik3d-s0	0.460413	0.611118	0.610422	0.472021	0.583244	0.64862	0.460627	0.452755	0.606918	0.60631	0.631842
649.fotonik3d-s1	0.609727	1.03837	1.36014	0.609948	0.652052	1.1644	0.610849	1.10006	1.07834	1.28441	1.0927
649.fotonik3d-s3	0.595547	1.41208	1.4049	0.647388	1.24688	1.33634	0.596145	1.09329	1.391	1.38055	1.42346
649.fotonik3d-s4	0.607638	1.03765	1.36144	0.60784	0.649587	1.16047	0.609031	1.11467	1.07074	1.27995	1.08458
654.roms-s0	0.798485	1.13311	1.08913	0.994318	1.03948	1.10252	0.802632	1.02605	1.15015	1.09692	1.17259
654.roms-s1	1.5435	1.58109	1.56155	1.54808	1.57324	1.56617	1.5408	1.55196	1.58104	1.56047	1.58165
654.roms-s2	0.486277	1.03591	1.09236	0.71881	0.791379	1.03298	0.486393	0.970702	1.09683	1.10952	1.15508
654.roms-s3	0.392961	0.846803	0.91961	0.675679	0.648371	0.878034	0.394439	0.606638	0.801518	0.855309	0.890102
654.roms-s4	1.05538	1.31722	1.34561	1.2395	1.23787	1.33184	1.05245	1.31046	1.34299	1.3522	1.35462
654.roms-s6	0.867945	1.52572	1.52531	0.867947	1.25217	1.49951	0.868976	0.827027	1.48801	1.52	1.48893
654.roms-s7	0.455147	0.925477	1.00677	0.757035	0.679495	0.94995	0.461111	0.496784	0.764112	0.878277	0.897924
654.roms-s8	1.62633	1.70215	1.70716	1.63067	1.67561	1.6979	1.63458	1.69895	1.70785	1.7103	1.70801
GEOMEAN	**0.5322920879**	**0.6814938571**	**0.6783717687**	**0.6224306421**	**0.6395259549**	**0.7126217753**	**0.5415294365**	**0.6345593256**	**0.6881024561**	**0.6819840414**	**0.7264252444**

Table A.10: Spec 17 IPC Improvement of Prefetching Models

Trace	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
602.gcc-s0	71.10	68.01	0.02	8.05	66.26	0.00	57.02	64.87	68.52	66.65
602.gcc-s1	111.78	119.27	96.37	88.13	117.43		104.32	112.28	119.02	122.59
602.gcc-s2	1.40	-0.20	0.20	0.20	0.18	-0.60	-0.08	0.07	-0.17	0.19
602.gcc-s3	41.21	39.90	39.93	38.14	43.26	-0.12	32.70	40.36	40.17	43.57
605.mcf-s0	-2.35	-4.84	4.98	4.43	7.48	0.26	-4.01	-1.42	-3.45	2.18
605.mcf-s1	61.53	0.20	-0.82	34.19	-0.58	-1.04	0.02	63.01	0.21	60.11
605.mcf-s2	-2.01	0.00	91.73	45.98	91.73	-3.84	21.26	16.10	10.17	75.42
605.mcf-s3	15.27	15.52	3.51	5.01	22.22	-1.63	9.86	13.88	15.88	20.01
605.mcf-s4	14.79	3.45	56.91	42.15	64.56	0.30	30.11	29.08	21.73	70.20
605.mcf-s5	21.18	6.63	38.15	40.42	53.74	-0.12	29.51	34.41	25.74	64.32
605.mcf-s6	-0.36	-1.16	1.61	1.90	6.53	-0.27	-0.34	0.65	-0.05	2.80
605.mcf-s7	0.05	-4.39	5.31	3.04	8.98	-0.14	8.71	6.01	-3.42	9.02
605.mcf-s8	-2.55	-1.30	4.74	2.26	3.32	0.37	-0.04	-0.38	-1.29	3.77
607.cactuBSSN-s0	44.01	0.00	4.81	-0.06	4.81	-0.05	-0.01	52.91	-0.01	52.40
607.cactuBSSN-s1	17.01	0.05	4.97	-0.04	4.97	0.02	6.81	17.21	16.28	17.84
607.cactuBSSN-s2	17.50	0.00	5.10	-0.09	5.10	-0.08	17.59	17.59	1.04	17.76
607.cactuBSSN-s3	4.36	4.35	0.00	0.00	4.35	0.06	0.00	4.36	4.35	4.36
619.lbm-s0	21.99	32.81	0.00	6.45	28.27	1.63	21.92	22.17	30.50	22.20
619.lbm-s1	12.04	19.54	0.00	4.54	17.31	0.25	6.93	8.12	18.35	8.12
619.lbm-s2	13.62	21.61	0.00	4.80	18.98	1.26	11.35	11.82	20.54	11.85
619.lbm-s3	12.30	19.83	0.00	4.69	17.59	-0.06	6.29	8.07	18.58	8.07
620.omnetpp-s0	5.15	1.52	10.15	9.49	12.23	0.07	-0.24	4.36	0.78	13.21
620.omnetpp-s1	5.01	0.94	19.34	17.46	19.34	0.03	-0.60	4.08	0.07	20.34
621.wrf-s0	1.95	1.72	0.19	1.35	1.65		1.31	1.90	1.70	1.91
621.wrf-s1	60.38	71.91	51.28	38.13	68.56	0.19	43.43	58.43	67.29	70.28
621.wrf-s2	48.74	57.45	38.17	29.65	55.11	-3.13	38.17	48.64	54.86	56.11
621.wrf-s3	6.95	8.02	4.04	4.44	7.62	0.09	5.69	7.02	7.88	7.62
623.xalancbmk-s0	16.32	9.05	66.75	63.63	66.92	0.07	2.45	11.12	8.26	67.22
623.xalancbmk-s1	14.56	25.47	28.66	30.93	28.68	0.79	25.91	22.23	29.22	31.04
623.xalancbmk-s2	36.04	58.38	64.53	64.36	64.71	-1.29	61.75	61.02	61.98	64.96
623.xalancbmk-s3	0.31	0.00	0.60	0.62	0.60	-0.21	0.36	0.47	0.27	0.80
623.xalancbmk-s4	0.40	0.00	0.59	0.05	0.59	-0.38	-0.20	-0.05	-0.31	0.27
623.xalancbmk-s5	0.22	0.00	0.61	0.67	0.61	0.34	0.18	0.37	0.18	0.75
649.fotonik3d-s0	32.73	32.58	2.52	26.68	40.88	0.05	-1.66	31.82	31.69	37.23
649.fotonik3d-s1	70.30	123.07	0.04	6.94	90.97	0.18	80.42	76.86	110.65	79.21
649.fotonik3d-s3	137.11	135.90	8.70	109.37	124.39	0.10	83.58	133.57	131.81	139.02
649.fotonik3d-s4	70.77	124.05	0.03	6.90	90.98	0.23	83.44	76.21	110.64	78.49
654.roms-s0	41.91	36.40	24.53	30.18	38.08	0.52	28.50	44.04	37.38	46.85
654.roms-s1	2.44	1.17	0.30	1.93	1.47	-0.17	0.55	2.43	1.10	2.47
654.roms-s2	113.03	124.64	47.82	62.74	112.43	0.02	99.62	125.56	128.17	137.54
654.roms-s3	115.49	134.02	71.95	65.00	123.44	0.38	54.38	103.97	117.66	126.51
654.roms-s4	24.81	27.50	17.45	17.29	26.20	-0.28	24.17	27.25	28.12	28.35
654.roms-s6	75.79	75.74	0.00	44.27	72.77	0.12	-4.71	71.44	75.13	71.55
654.roms-s7	103.34	121.20	66.33	49.29	108.71	1.31	9.15	67.88	92.97	97.28
654.roms-s8	4.66	4.97	0.27	3.03	4.40	0.51	4.47	5.01	5.16	5.02
AVERAGE	**47.57**	**42.12**	**29.44**	**35.51**	**53.74**	**3.28**	**41.60**	**48.06**	**46.68**	**59.34**

A.3 GAP

Table A.11: GAP Accuracy of Prefetching Models

Trace	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
bc-0	13.34	28.07	18.93	23.69	18.78	36.66	18.14	19.88	23.15	15.62
bc-12	12.44	28.67	19.72	24.71	19.42	32.22	15.76	17.28	18.00	14.53
bc-3	15.19	30.59	17.91	24.74	17.94	37.05	17.16	19.70	20.97	15.39
bc-5	16.03	30.43	20.66	32.68	20.26	19.84	25.64	21.53	25.64	17.47
bfs-10	66.87	63.86	79.32	81.31	68.35	66.32	36.87	57.98	58.79	57.85
bfs-14	67.18	67.88	85.19	81.23	73.74	85.07	7.28	74.40	67.87	73.63
bfs-3	52.45	62.81	77.21	70.74	66.11	78.04	38.34	52.87	56.10	51.69
bfs-8	63.91	67.46	83.55	77.99	72.44	52.86	47.46	63.22	63.36	63.91
cc-13	26.89	44.23	39.48	49.54	39.39	91.26	25.89	34.82	31.31	28.61
cc-14	26.90	44.97	38.84	49.41	38.77	82.79	22.15	34.10	27.74	28.30
cc-5	26.78	44.87	38.09	49.47	38.10	65.26	24.76	35.23	31.54	28.83
cc-6	27.28	44.56	41.03	49.12	40.77	87.74	23.35	33.24	28.42	27.93
pr-10	22.92	43.88	41.17	39.00	39.71	55.28	15.69	30.76	33.92	25.32
pr-14	26.77	46.14	48.76	45.87	46.72	63.47	14.69	32.78	27.84	28.04
pr-3	21.58	43.41	37.01	35.43	36.11	67.62	15.04	29.33	36.14	23.69
pr-5	23.32	44.68	40.94	38.96	39.70	56.74	20.91	30.43	32.18	25.14
sssp-10	28.72	34.52	33.78	49.38	33.41	67.62	26.45	38.32	34.30	30.33
sssp-14	28.77	34.95	33.72	49.37	33.42	77.80	24.67	38.14	33.52	30.24
sssp-3	28.76	34.82	33.70	49.24	33.35	46.31	23.66	36.50	30.96	29.31
sssp-5	28.67	34.98	33.82	49.39	33.46	59.23	21.96	36.95	31.13	29.61
Average	**31.24**	**43.79**	**43.14**	**48.56**	**40.50**	**61.46**	**23.29**	**36.87**	**35.64**	**32.27**

Table A.12: GAP Coverage of Prefetching Models

Trace	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
bc-0	20.85	2.07	17.55	26.94	18.00	1.61	3.51	18.30	2.95	29.39
bc-12	19.63	2.02	18.09	27.26	18.56	1.22	7.69	19.69	6.71	30.52
bc-3	23.11	1.82	16.48	27.63	16.82	1.65	5.75	19.48	4.59	29.65
bc-5	23.47	2.82	17.34	31.01	17.83	1.90	4.27	19.61	4.27	30.77
bfs-10	72.73	63.84	64.89	77.35	69.24	5.27	17.53	53.50	60.17	72.39
bfs-14	69.40	70.68	72.52	73.04	75.86	3.99	0.00	56.50	70.67	78.60
bfs-3	57.25	59.75	60.94	63.41	65.51	4.82	24.38	52.17	57.55	68.92
bfs-8	66.98	69.47	71.98	70.20	75.12	4.30	26.07	59.53	66.86	78.67
cc-13	35.63	2.64	19.74	35.99	20.74	1.38	7.79	31.85	6.38	43.79
cc-14	35.61	2.70	19.18	35.67	20.17	1.63	7.23	31.59	5.82	43.53
cc-5	17.05	2.82	18.60	35.88	19.67	1.84	6.73	31.65	5.67	43.41
cc-6	1.97	2.79	21.05	36.16	22.17	1.02	9.81	32.30	8.13	44.25
pr-10	32.70	4.10	29.08	34.18	30.13	1.31	1.91	26.63	4.37	40.24
pr-14	5.98	4.39	29.45	35.44	30.63	1.38	3.13	29.29	5.36	42.87
pr-3	31.36	4.01	28.60	33.20	29.65	1.12	1.27	25.44	3.88	38.85
pr-5	33.10	4.14	28.93	34.09	30.12	1.47	3.98	27.11	5.15	40.50
sssp-10	36.44	7.77	14.79	38.37	19.80	2.58	4.61	32.04	8.02	41.67
sssp-14	36.44	7.76	14.75	38.39	19.75	2.24	3.87	32.07	7.36	41.64
sssp-3	36.43	8.04	14.81	38.39	19.84	3.00	5.98	32.42	8.61	41.93
sssp-5	36.38	7.51	14.80	38.37	19.70	2.71	4.70	31.98	7.82	41.64
Average	**34.63**	**16.56**	**29.68**	**41.55**	**31.97**	**2.32**	**7.51**	**33.16**	**17.52**	**46.16**

Table A.13: GAP MPKI of Prefetching Models

Trace	no	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
bc-0	83.953	50.993	41.622	41.122	42.736	41.073	41.710	43.280	43.717	42.295	40.208
bc-12	85.494	52.774	42.365	41.612	43.079	41.596	42.611	45.373	45.256	44.539	40.984
bc-3	80.680	47.284	40.053	39.782	39.845	39.709	40.111	42.479	41.876	41.314	38.702
bc-5	74.255	44.177	36.567	36.110	33.311	36.064	37.328	37.537	39.194	37.537	35.468
bfs-10	35.525	4.586	6.665	6.432	3.781	5.712	16.824	14.887	8.733	7.382	5.247
bfs-14	34.284	5.366	5.057	4.738	4.714	4.175	16.467	17.143	7.532	5.060	3.711
bfs-3	36.007	8.867	7.622	7.330	7.225	6.606	17.134	14.764	9.538	8.285	6.283
bfs-8	34.717	5.855	5.321	4.877	5.255	4.343	16.643	12.994	7.109	5.812	3.742
cc-13	48.810	25.129	24.278	22.038	17.791	21.968	24.092	26.255	24.584	25.316	21.076
cc-14	47.178	24.502	23.443	21.444	17.412	21.382	23.264	25.893	23.871	24.723	20.467
cc-5	45.488	22.860	22.608	20.833	17.013	20.759	22.474	24.383	22.804	23.455	19.652
cc-6	53.618	25.147	26.651	23.808	19.188	23.721	26.568	29.486	27.241	28.273	23.237
pr-10	123.430	65.323	60.509	50.449	50.540	50.495	61.260	64.293	62.117	61.725	53.140
pr-14	112.056	57.248	55.078	44.309	44.137	44.423	55.546	60.889	57.287	57.897	48.059
pr-3	129.327	67.113	63.353	53.326	53.403	53.315	64.105	66.613	64.402	64.365	55.386
pr-5	122.714	64.538	60.192	49.955	49.954	49.980	60.831	64.422	61.789	61.855	52.704
sssp-10	42.405	19.864	21.034	19.427	15.270	19.087	20.792	21.673	18.747	21.161	16.567
sssp-14	42.371	19.840	21.018	19.420	15.267	19.074	20.791	21.721	18.730	21.206	16.566
sssp-3	42.399	19.851	21.021	19.435	15.287	19.091	20.907	22.050	18.901	21.437	16.683
sssp-5	42.446	19.893	21.047	19.444	15.282	19.108	20.838	22.069	18.941	21.409	16.705
GEOMEAN	**58.997**	**24.566**	**23.655**	**21.694**	**19.351**	**21.105**	**28.990**	**29.717**	**25.041**	**24.563**	**20.104**

Table A.14: GAP IPC of Prefetching Models

Trace	no	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
bc-0	0.148631	0.133852	0.149042	0.146921	0.140625	0.147235	0.149519	0.146736	0.152909	0.148158	0.148221
bc-12	0.143755	0.121713	0.143891	0.142658	0.129805	0.142738	0.144194	0.136908	0.141493	0.138838	0.138382
bc-3	0.162961	0.160196	0.163169	0.160115	0.168119	0.160378	0.164043	0.159004	0.166275	0.161093	0.160133
bc-5	0.218528	0.238594	0.217884	0.212327	0.26423	0.212163	0.21688	0.21509	0.219658	0.21509	0.210351
bfs-10	0.27221	0.41761	0.400215	0.400841	0.428018	0.420279	0.280261	0.285578	0.384858	0.38782	0.429336
bfs-14	0.268798	0.400398	0.410631	0.411783	0.408619	0.430627	0.272816	0.268793	0.392992	0.410669	0.441116
bfs-3	0.27822	0.398926	0.403126	0.401675	0.412869	0.42232	0.284421	0.297185	0.387581	0.391411	0.427234
bfs-8	0.268925	0.402166	0.407541	0.413472	0.409714	0.429774	0.27393	0.294545	0.388363	0.39619	0.43556
cc-13	0.222583	0.226374	0.22329	0.230291	0.246189	0.231002	0.224698	0.214298	0.240351	0.218618	0.244071
cc-14	0.245896	0.25327	0.246857	0.25336	0.276349	0.254154	0.248568	0.233959	0.265794	0.240431	0.268564
cc-5	0.272496	0.273303	0.273668	0.279419	0.311996	0.280446	0.275294	0.263642	0.299795	0.26925	0.300711
cc-6	0.20438	0.190078	0.205393	0.21333	0.219124	0.214335	0.205818	0.194359	0.219532	0.199476	0.223125
pr-10	0.0617397	0.0553322	0.0618017	0.0638679	0.0611028	0.0637824	0.0616827	0.0600467	0.057529	0.0610851	0.0606023
pr-14	0.0653616	0.0630617	0.0653413	0.0681568	0.0652749	0.0680287	0.0652966	0.0619235	0.059748	0.0634147	0.0635506
pr-3	0.060085	0.0547061	0.0602161	0.0621426	0.0595019	0.0620929	0.0600697	0.0587871	0.0565747	0.0596491	0.0594312
pr-5	0.0620195	0.0557739	0.0620678	0.0642729	0.0615355	0.0641982	0.0619669	0.0595369	0.057615	0.0609203	0.0607942
sssp-10	0.338792	0.394572	0.343947	0.34491	0.426712	0.352768	0.344288	0.332712	0.405045	0.341956	0.408948
sssp-14	0.338557	0.394741	0.34382	0.344716	0.426814	0.352742	0.343949	0.332701	0.404749	0.340947	0.40848
sssp-3	0.338487	0.394462	0.34384	0.344685	0.426253	0.352678	0.342965	0.329773	0.39977	0.338471	0.404133
sssp-5	0.338611	0.394877	0.343608	0.344854	0.42716	0.352656	0.343621	0.329682	0.401703	0.339337	0.406043
GEOMEAN	**0.1862023737**	**0.2020443904**	**0.2025045654**	**0.2047042129**	**0.2170872789**	**0.2076389075**	**0.1879488549**	**0.1834200878**	**0.2076947942**	**0.1988752023**	**0.2140711006**

Table A.15: GAP IPC Imrovement of Prefetching Models

Trace	next_line	bo	sisb	blue	sisb_bo	TransFetch	LSTM	LSTM_next	LSTM_bo	LSTM_sisb
bc-0	-9.94	0.28	-1.15	-5.39	-0.94	0.60	-1.27	2.88	-0.32	-0.28
bc-12	-15.33	0.09	-0.76	-9.70	-0.71	0.31	-4.76	-1.57	-3.42	-3.74
bc-3	-1.70	0.13	-1.75	3.17	-1.59	0.66	-2.43	2.03	-1.15	-1.74
bc-5	9.18	-0.29	-2.84	20.91	-2.91	-0.75	-1.57	0.52	-1.57	-3.74
bfs-10	53.41	47.02	47.25	57.24	54.40	2.96	4.91	41.38	42.47	57.72
bfs-14	48.96	52.77	53.19	52.02	60.20	1.49	0.00	46.20	52.78	64.11
bfs-3	43.39	44.89	44.37	48.40	51.79	2.23	6.82	39.31	40.68	53.56
bfs-8	49.55	51.54	53.75	52.35	59.81	1.86	9.53	44.41	47.32	61.96
cc-13	1.70	0.32	3.46	10.61	3.78	0.95	-3.72	7.98	-1.78	9.65
cc-14	3.00	0.39	3.04	12.38	3.36	1.09	-4.85	8.09	-2.22	9.22
cc-5	0.30	0.43	2.54	14.50	2.92	1.03	-3.25	10.02	-1.19	10.35
cc-6	-7.00	0.50	4.38	7.21	4.87	0.70	-4.90	7.41	-2.40	9.17
pr-10	-10.38	0.10	3.45	-1.03	3.31	-0.09	-2.74	-6.82	-1.06	-1.84
pr-14	-3.52	-0.03	4.28	-0.13	4.08	-0.10	-5.26	-8.59	-2.98	-2.77
pr-3	-8.95	0.22	3.42	-0.97	3.34	-0.03	-2.16	-5.84	-0.73	-1.09
pr-5	-10.07	0.08	3.63	-0.78	3.51	-0.08	-4.00	-7.10	-1.77	-1.98
sssp-10	16.46	1.52	1.81	25.95	4.13	1.62	-1.79	19.56	0.93	20.71
sssp-14	16.60	1.55	1.82	26.07	4.19	1.59	-1.73	19.55	0.71	20.65
sssp-3	16.54	1.58	1.83	25.93	4.19	1.32	-2.57	18.10	0.00	19.39
sssp-5	16.62	1.48	1.84	26.15	4.15	1.48	-2.64	18.63	0.21	19.91
Average	**10.44**	**10.23**	**11.38**	**18.24**	**13.29**	**0.94**	**-1.42**	**12.81**	**8.23**	**16.96**

Appendix B

Model Code

```
from torch import nn,zeros,cat

class LSTM1(nn.Module):
    def __init__(self, n_vocab,n_pc, embedding_dim = 128,
        embedding_offset_dim = 8, pc_dim=16, lstm_size = 16,
           num_layers = 2):
        super(LSTM1, self).__init__()

        self.embedding_dim = embedding_dim
        self.embedding_offset_dim = embedding_offset_dim
        self.n_pc = n_pc
        self.pc_dim = pc_dim
        self.lstm_in = self.embedding_dim  + self.pc_dim
        self.lstm_size = lstm_size
        self.num_layers = num_layers
        self.n_vocab = n_vocab
        self.embedding = nn.Embedding(
            num_embeddings = n_vocab,
            embedding_dim = self.embedding_dim,
        )

        self.pc_embedding = nn.Embedding(
            num_embeddings = n_pc,
            embedding_dim = self.pc_dim
        )
        self.lstm = nn.LSTM(
                input_size = self.lstm_in,
                hidden_size = self.lstm_size,
                num_layers = self.num_layers,
                batch_first = True,
                dropout = 0.1,
                bidirectional = False)

        self.fc = nn.Sequential(
            nn.Linear(self.lstm_size, n_vocab),
        )

    def init_state(self, sequence_length):
        return (zeros(self.num_layers, sequence_length, self.
           lstm_size),
        zeros(self.num_layers, sequence_length, self.lstm_size))

    def forward(self, x, prev_state=None):

        embed = self.embedding(x[:, :, 0])
        pc_embed = self.pc_embedding(x[:, :, 1])
        feats = cat([embed, pc_embed], dim=2)

        if(prev_state):
            output, state = self.lstm(feats, prev_state)
        else:
            output,state = self.lstm(feats)

        logits = self.fc(output)

        if(prev_state):
            return logits, state
        return logits
```

Appendix C

Biblography

[1] J. Mukundan and J. F. Martinez. "MORSE: Multi-objective reconfigurable self-optimizing memory scheduler". In: IEEE, Feb. 2012, pp. 1–12. ISBN: 978-1-4673-0826-7. DOI: 10.1109/HPCA.2012.6168945.

[2] D. A. Jimenez. "An optimized scaled neural branch predictor". In: IEEE, Oct. 2011, pp. 113–118. ISBN: 978-1-4577-1954-7. DOI: 10.1109/ICCD.2011.6081385.

[3] D. Jimenez and C. Lin. "Dynamic branch prediction with perceptrons". In: IEEE Comput. Soc, 2001, pp. 197–206. ISBN: 0-7695-1019-1. DOI: 10.1109/HPCA.2001.903263.

[4] A. F. Ganai and F. Khursheed. "Predicting next Word using RNN and LSTM cells: Stastical Language Modeling". In: IEEE, Nov. 2019, pp. 469–474. ISBN: 978-1-7281-0899-5. DOI: 10.1109/ICIIP47207.2019.8985885.

[5] G. Petneházi. "Recurrent Neural Networks for Time Series Forecasting". In: (Dec. 2018).

[6] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (8 Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[7] P. Braun and H. Litz. "Understanding Memory Access Patterns for Prefetching". In: *International Workshop on AI-assisted Design for Architecture (AIDArc), held in conjunction with ISCA* (2019). URL:

[8] M. Hashemi et al. "Learning Memory Access Patterns". In: (Mar. 2018). URL:

[9] G. Ayers et al. "Classifying memory access patterns for prefetching". In: Association for Computing Machinery, Mar. 2020, pp. 513–526. ISBN: 9781450371025. DOI: 10.1145/3373376.3378498.

[10] M. Schuster and K. Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45 (11 1997), pp. 2673–2681. ISSN: 1053587X. DOI: 10.1109/78.650093.

[11] W. Y. Chen et al. "Data access microarchitectures for superscalar processors with compiler-assisted data prefetching". In: ACM Press, 1991, pp. 69–73. ISBN: 0897914600. DOI: 10.1145/123465.123478.

[12] S. Ainsworth and T. M. Jones. "Software Prefetching for Indirect Memory Accesses". In: *ACM Transactions on Computer Systems* 36 (3 Aug. 2019), pp. 1–34. ISSN: 0734-2071. DOI: 10.1145/3319393.

[13] J. W. C. Fu, J. H. Patel, and B. L. Janssens. "Stride directed prefetching in scalar processors". In: *ACM SIGMICRO Newsletter* 23 (1-2 Dec. 1992), pp. 102–110. ISSN: 1050-916X. DOI: 10.1145/144965.145006.

[14] S. Iacobovici et al. "Effective stream-based and execution-based data prefetching". In: ACM Press, 2004, p. 1. ISBN: 1581138393. DOI: 10.1145/1006209.1006211.

[15] F. Dahlgren and P. Stenstrom. "Effectiveness of hardware-based stride and sequential prefetching in shared-memory multiprocessors". In: IEEE Comput. Soc. Press, 1995, pp. 68–77. ISBN: 0-8186-6445-2. DOI: 10.1109/HPCA.1995.386554.

[16] M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad. "Domino Temporal Data Prefetcher". In: IEEE, Feb. 2018, pp. 131–142. ISBN: 978-1-5386-3659-6. DOI: 10.1109/HPCA.2018.00021.

[17] M. Bakhshalipour et al. "Bingo spatial data prefetcher". In: Institute of Electrical and Electronics Engineers Inc., Mar. 2019, pp. 399–411. ISBN: 9781728114446. DOI: 10.1109/HPCA.2019.00053.

[18] A. Seznec. "A new case for the TAGE branch predictor". In: ACM Press, 2011, p. 117. ISBN: 9781450310536. DOI: 10.1145/2155620.2155635.

[19] D. Joseph and D. Grunwald. "Prefetching using Markov predictors". In: ACM Press, 1997, pp. 252–263. ISBN: 0897919017. DOI: 10.1145/264107.264207.

[20] K. Nesbit and J. Smith. "Data Cache Prefetching Using a Global History Buffer". In: IEEE, 2004, pp. 96–96. ISBN: 0-7695-2053-7. DOI: 10.1109/HPCA.2004.10030.

[21] M. Annavaram, J. M. Patel, and E. S. Davidson. "Data prefetching by dependence graph precomputation". In: ACM Press, 2001, pp. 52–61. ISBN: 0769511627. DOI: 10.1145/379240.379251.

[22] J. Collins et al. "Speculative precomputation: long-range prefetching of delinquent loads". In: IEEE Comput. Soc, 2001, pp. 14–25. ISBN: 0-7695-1162-7. DOI: 10.1109/ISCA.2001.937427.

[23] A. Naithani et al. "Precise Runahead Execution". In: IEEE, Feb. 2020, pp. 397–410. ISBN: 978-1-7281-6149-5. DOI: 10.1109/HPCA47549.2020.00040.

[24] S. VERMA and D. M. KOPPELMAN. “THE INTERACTION AND RELATIVE EFFECTIVENESS OF HARDWARE AND SOFTWARE DATA PREFETCH”. In: *Journal of Circuits, Systems and Computers* 21 (02 Apr. 2012), p. 1240002. ISSN: 0218-1266. DOI: 10.1142/S0218126612400026.

[25] S. Haykin. *Neural Networks and Learning Machines*. Third. Pearson Education, Inc., 2009, pp. 48–48.

[26] T. Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: (Oct. 2013).

[27] P. Michaud. “Best-Offset Hardware Prefetching”. In: (2016). DOI: 10.1109/HPCA.2016.7446087Ãŕ. URL:

[28] A. Jain and C. Lin. “Linearizing irregular memory accesses for improved correlated prefetching”. In: ACM Press, 2013, pp. 247–259. ISBN: 9781450326384. DOI: 10.1145/2540708.2540730.

[29] A. Ros. “Berti: A Per-Page Best-Request-Time Delta Prefetcher”. In: 2019.

[30] A. Ros and A. Jimborean. “The entangling instruction prefetcher”. In: May 2021.

[31] P. Zhang et al. “Fine-grained address segmentation for attention-based variable-degree prefetching”. In: ACM, May 2022, pp. 103–112. ISBN: 9781450393386. DOI: 10.1145/3528416.3530236.

[32] A. Vaswani et al. “Attention Is All You Need”. In: (June 2017).

[33] Z. Shi et al. “A hierarchical neural model of data prefetching”. In: Association for Computing Machinery, Apr. 2021, pp. 861–873. ISBN: 9781450383172. DOI: 10.1145/3445814.3446752.

[34] R. Bera et al. “Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning”. In: ACM, Oct. 2021, pp. 1121–1137. ISBN: 9781450385572. DOI: 10.1145/3466752.3480114.

[35] A. Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: (Dec. 2019).

[36] D. Quang. *Modified ChampSim Simulator*. 2021.

[37] C.-K. Luk et al. “Pin”. In: ACM Press, 2005, p. 190. ISBN: 1595930566. DOI: 10.1145/1065010.1065034.

[38] S. P. E. Corporation. *SPEC CPU® 2006*. 2006.

[39] S. P. E. Corporation. *SPEC CPU® 2017*. 2017.

[40] S. Beamer, K. Asanović, and D. Patterson. “The GAP Benchmark Suite”. In: (Aug. 2015).

[41] R. S. Amant, D. A. Jimenez, and D. Burger. "Low-power, high-performance analog neural branch prediction". In: IEEE, Nov. 2008, pp. 447–458. ISBN: 978-1-4244-2836-6. DOI: 10.1109/MICRO.2008.4771812.

[42] E. Garza et al. "Bit-level perceptron prediction for indirect branches". In: ACM, June 2019, pp. 27–38. ISBN: 9781450366694. DOI: 10.1145/3307650.3322217.

[43] S. J. Tarsa et al. "Improving Branch Prediction By Modeling Global History with Convolutional Neural Networks". In: (June 2019).

[44] G. Rjoub et al. "Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems". In: *Concurrency and Computation: Practice and Experience* 33 (23 Dec. 2021). ISSN: 1532-0626. DOI: 10.1002/cpe.5919.

[45] W. Kang and S. Yoo. "Dynamic Management of Key States for Reinforcement Learning-assisted Garbage Collection to Reduce Long Tail Latency in SSD". In: IEEE, June 2018, pp. 1–6. ISBN: 978-1-5386-4114-9. DOI: 10.1109/DAC.2018.8465934.

[46] E. Ipek et al. "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach". In: IEEE, June 2008, pp. 39–50. ISBN: 978-0-7695-3174-8. DOI: 10.1109/ISCA.2008.21.

[47] S. M. P. D. et al. "A Q-Learning Based Self-Adaptive I/O Communication for 2.5D Integrated Many-Core Microprocessor and Memory". In: *IEEE Transactions on Computers* 65 (4 Apr. 2016), pp. 1185–1196. ISSN: 0018-9340. DOI: 10.1109/TC.2015.2439255.

[48] P. Diaz. "Mechanisms to Improve the Efficiency of Hardware Data Prefetchers". University of Edinburgh, 2010. URL:

[49] A. Ros. "BLE: A Timely, IP-based Data Prefetcher". In: 2021.

[50] A. D. Patterson and L. J. Hennessy. *Computer Organization and Design The Hardware/Software Interface*. 5th ed. Morgan Kaufmann, 2014, pp. 427–430.

[51] Z. Hu, M. Martonosi, and S. Kaxiras. "TCP: tag correlating prefetchers". In: IEEE Comput. Soc, 2003, pp. 317–326. ISBN: 0-7695-1871-0. DOI: 10.1109/HPCA.2003.1183549.

[52] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: (Dec. 2014).

[53] G. Hinton, N. Srivastava, and K. Swersky. *Overview of mini--batch gradient descent*. 2012.

[54] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to Sequence Learning with Neural Networks". In: (Sept. 2014).

[55] M. S. Al-Radhi, T. G. Csapó, and G. Németh. "RNN-based speech synthesis using a continuous sinusoidal model". In: (Apr. 2019).

[56] C. Olah. *Understanding LSTMs*. Aug. 2015. URL:

[57] *ML-Based Data Prefetching Competition*. 2021. URL:

[58] S. Lloyd. “Least squares quantization in PCM”. In: *IEEE Transactions on Information Theory* 28 (2 Mar. 1982), pp. 129–137. ISSN: 0018-9448. DOI: 10.1109/TIT.1982.1056489.

[59] C. R. Harris et al. “Array programming with NumPy”. In: *Nature* 585 (7825 Sept. 2020), pp. 357–362. ISSN: 0028-0836. DOI: 10.1038/s41586-020-2649-2.

[60] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL:

[61] W. McKinney. “Data Structures for Statistical Computing in Python”. In: 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.