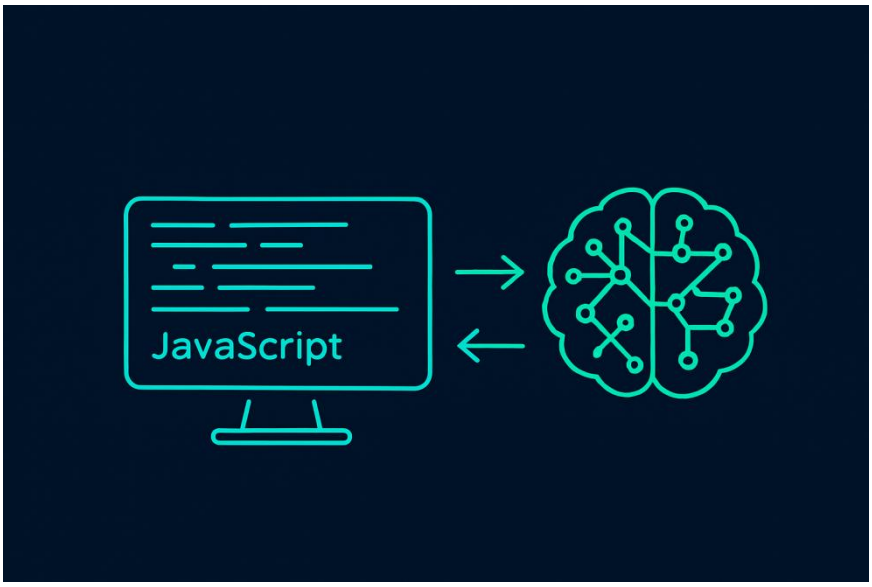
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΑΛΕΞΑΝΔΡΕΙΑ ΠΑΝΕΠΙΣΤΗΜΙΟΥΠΟΛΗ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ
ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ: Διόρθωση σφαλμάτων σε Προγράμματα Javascript με τη χρήση Νευρωνικών Δικτύων Βαθιάς Μάθησης-Deep Learning



Τη φοιτήτρια
Σαρρηκυριακίδου Πολυξένη
Αρ. Μητρώου: 113792

Επιβλέπων
Γουλιάνας Κωνσταντίνος

Ημερομηνία 15/07/2025

Τίτλος Δ.Ε. Διόρθωση σφαλμάτων σε Προγράμματα Javascript με τη χρήση Νευρωνικών Δικτύων Βαθιάς Μάθησης-Deep Learning
Κωδικός Δ.Ε 23147
Ονοματεπώνυμο φοιτητή Σαρρηκυριακίδου Πολυξένη
Ονοματεπώνυμο εισηγητή Γουλιάνας Κωνσταντίνος
Ημερομηνία ανάληψης Δ.Ε. 30/03/2023
Ημερομηνία περάτωσης Δ.Ε. 1507/2025

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

ΠΕΡΙΕΧΟΜΕΝΑ

Περιεχόμενα

1 ΕΙΣΑΓΩΓΗ 4

1.1 Σκοπός της εργασίας 4

1.1.1 Τι πρόκειται να μελετηθεί στην εργασία 4

1.1.2 Η σημασία της διόρθωσης σφαλμάτων σε JavaScript 5

1.2 Αντικείμενο της μελέτης 7

1.2.1 Περιγραφή του προβλήματος και των προσεγγίσεων 7

1.3 Δομή της εργασίας 8

1.3.1 Γρήγορη περιγραφή των κεφαλαίων και του περιεχομένου τους 8

2 Βασικές Έννοιες 9

2.1 JavaScript και σφάλματα 9

2.1.1 Τι είναι τα σφάλματα σε προγράμματα JavaScript (syntactical, logical, runtime errors) 9

2.1.2 Συντακτικά Σφάλματα (Syntax Errors) 9

2.1.3 Λογικά Σφάλματα (Logical Errors): 10

2.1.4 Runtime Errors 11

2.2 Τεχνικές διόρθωσης σφαλμάτων 22

2.2.1 Παραδοσιακές προσεγγίσεις διόρθωσης σφαλμάτων σε γλώσσες προγραμματισμού 22

2.3 Εισαγωγή στα Νευρωνικά Δίκτυα και τη Βαθιά Μάθηση 27

2.3.1 Περιγραφή των Νευρωνικών Δικτύων (ΝΔ) 27

2.3.2 Είδη νευρωνικών δικτύων βαθιάς μάθησης(αρχιτεκτονική και λειτουργία) 28

2.3.3 Εισαγωγή στις μεθόδους Βαθιάς Μάθησης (DL) 31

2.4 Αντίστοιχες Εργασίες στη Διόρθωση Σφαλμάτων με Βαθιά Μάθηση 32

2.4.1 Εργαλεία και Αλγόριθμοι για Ανίχνευση και Διόρθωση Σφαλμάτων 33

2.4.2 Σύγκριση των Διαφορετικών Προσεγγίσεων 33

2.4.3 Σύγχρονες Ερευνητικές Προσεγγίσεις στην Ανίχνευση Σφαλμάτων JavaScript 35

2.4.4 Τεχνικές Λεπτομέρειες Σύγχρονων Συστημάτων Ανίχνευσης Σφαλμάτων 37

2.5 Τεχνικές Εντοπισμού και Επίλυσης Σφαλμάτων 38

2.5.1 Βελτιωμένη Τεχνική Εντοπισμού με Instrumentalization....42
2.5.2 Τεχνικές Ανάλυσης Ροής Δεδομένων....44
3 Λύσεις Διόρθωσης Σφαλμάτων μέσω Βαθιάς Μάθησης....49
3.1 Επισκόπηση Νευρωνικών Δικτύων για τον εντοπισμό και τη διόρθωση σφαλμάτων....49
3.1.1 Πλεονεκτήματα και μειονεκτήματα κάθε προσέγγισης....50
3.2 Που θα εστιάσουμε στην έρευνα αυτή....51
4 Μεθοδολογία....52
4.1 Προσομοίωση με Deep Learning Αλγόριθμους....52
4.2 Παρουσίαση δύο αλγορίθμων διόρθωσης σφαλμάτων....53
4.2.1 Αλγόριθμος Pattern-Enhanced Deep Learning....53
4.2.2 Αλγόριθμος CodeBERT....53
4.3 Περιγραφή του εκπαιδευτικού μοντέλου....54
4.3.1 Προετοιμασία δεδομένων....54
4.3.2 Μοντέλα που Χρησιμοποιήθηκαν....54
4.3.3 Αρχιτεκτονική....54
4.3.4 Αξιολόγηση και Μετρικές....55
5 Αποτελέσματα....56
5.1 Αξιολόγηση των Αλγορίθμων....56
5.2 Αξιολόγηση επίδοσης....63
5.2.1 Συνολική Απόδοση....63
5.2.2 Απόδοση ανά Τύπο Σφάλματος....63
5.3 Παραδείγματα διόρθωσης σφαλμάτων....64
5.3.1 Συντακτικά Σφάλματα....64
5.3.2 Λογικά Σφάλματα....65
5.3.3 Runtime Σφάλματα....66
5.3.4 Ανάλυση Αποτελεσμάτων....67
6 Συμπεράσματα και Μελλοντικές Εργασίες....68
6.1 Συμπεράσματα από την εφαρμογή των μεθόδων....68
6.2 Προτάσεις για μελλοντική έρευνα....69
6.3 Προκλήσεις και περιορισμοί της εργασίας....69
7 Βιβλιογραφία....71

Σχήμα 1 : Σφάλμα στην κονσόλα 12
Σχήμα 2 : Δομή νευρωνικού δικτύου 28
Σχήμα 3 : Convolutional neural network 29
Σχήμα 4 : RNN 29
Σχήμα 5: Autoencoders 30
Σχήμα 6 : DBN 31
Σχήμα 7 : Deepbugs 35
Σχήμα 8 : Typescript 36
Σχήμα 9 : cross-site scripting 36
Σχήμα 10 : react dom 37
Σχήμα 11 : GPT teaches chemistry 38
Σχήμα 12 : εικόνα για την εκπαίδευση μοντέλου 50
Σχήμα 13 : PyTorch workflow 51

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή μου για την καθοδήγηση και την πολύτιμη υποστήριξή του καθ' όλη τη διάρκεια εκπόνησης της παρούσας πτυχιακής εργασίας. Οι συμβουλές και οι παρατηρήσεις του υπήρξαν καθοριστικές για την ολοκλήρωση αυτής της μελέτης. Επίσης, θα ήθελα να ευχαριστήσω το τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων για τις γνώσεις και τα εφόδια που μου προσέφερε κατά τη διάρκεια των σπουδών μου. Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου για την αμέριστη συμπαράσταση και υποστήριξη που μου παρείχε σε όλη τη διάρκεια των σπουδών μου.

ΠΕΡΙΛΗΨΗ

Η εργασία εξετάζει τη διόρθωση σφαλμάτων σε προγράμματα JavaScript με τη χρήση αλγορίθμων Βαθιάς Μάθησης (Deep learning). Αρχικά, γίνεται βιβλιογραφική ανασκόπηση των παραδοσιακών και σύγχρονων προσεγγίσεων για τον εντοπισμό σφαλμάτων, όπως και των μοντέλων βαθιάς μάθησης που έχουν χρησιμοποιηθεί σε αυτό το πεδίο, όπως τα Deep Neural Networks (DNNs), Recurrent Neural Networks (RNNs), και Convolutional Neural Networks (CNNs). Αυτά τα μοντέλα προσφέρουν την ικανότητα να ανιχνεύουν σύνθετα σφάλματα, τα οποία περιλαμβάνουν προσθήκες ή διαγραφές δηλώσεων σε κώδικα, εντοπίζοντας τόσο απλά όσο και σύνθετα bug patterns. Στη συνέχεια, η εργασία επικεντρώνεται στην προσομοίωση δύο τουλάχιστον από τους προαναφερθέντες αλγορίθμους, χρησιμοποιώντας εργαλεία όπως το TensorFlow και το PyTorch. Οι αλγόριθμοι αυτοί θα αξιολογηθούν βάσει της ικανότητάς τους να εντοπίζουν σφάλματα σε υπάρχοντα datasets ή σε τεχνητά δεδομένα. Έρευνες όπως αυτή για την πλατφόρμα DeepBugs έχουν δείξει ότι η εκπαίδευση νευρωνικών δικτύων με θετικά και αρνητικά παραδείγματα σφαλμάτων μπορεί να βοηθήσει στην αυτόματη διόρθωση σφαλμάτων JavaScript[1]. Η μελέτη καταλήγει σε συμπεράσματα σχετικά με την αποτελεσματικότητα των αλγορίθμων βαθιάς μάθησης στον εντοπισμό σφαλμάτων και προτείνει μελλοντικές ερευνητικές κατευθύνσεις για τη βελτίωση των τεχνικών αυτών.[2]

[1] https://www.mdpi.com/2673-2688/5/4/86

[2] https://www.software-lab.org/publications/DeepBugs_TR_Nov2017.pdf

ABSTRACT

The project focuses on bug detection and correction in JavaScript programs using Deep Learning algorithms. Initially, a literature review is conducted on both traditional and modern approaches for bug detection, as well as on various deep learning models used in this domain, such as Deep Neural Networks (DNNs), Recurrent Neural Networks (RNNs), and Convolutional Neural Networks (CNNs). These models offer the ability to detect complex bugs, including errors that require adding or deleting statements, by identifying simple and complex bug patterns. The project then moves to the simulation of at least two of the algorithms using tools like TensorFlow and PyTorch. These algorithms will be evaluated based on their ability to identify bugs using existing datasets or synthetic data. Research, such as that on the DeepBugs platform, has demonstrated that training neural networks with both positive and negative bug examples can assist in the automatic correction of JavaScript errors. The study concludes with insights into the effectiveness of deep learning algorithms in bug detection and suggests future research directions to improve these techniques further.

1 ΕΙΣΑΓΩΓΗ

1.1 Σκοπός της εργασίας

Ο σκοπός της παρούσας πτυχιακής εργασίας είναι η διερεύνηση της δυνατότητας αυτόματης διόρθωσης σφαλμάτων σε προγράμματα JavaScript μέσω της χρήσης νευρωνικών δικτύων και αλγορίθμων βαθιάς μάθησης. Δεδομένης της αυξανόμενης ανάγκης για ταχύ και αποτελεσματικό εντοπισμό σφαλμάτων σε εφαρμογές ιστού και λογισμικού, η εργασία επιδιώκει να παρουσιάσει και να αξιολογήσει μοντέλα βαθιάς μάθησης, όπως τα Deep Neural Networks (DNNs), Recurrent Neural Networks (RNNs) και Convolutional Neural Networks (CNNs), που έχουν αναπτυχθεί για την ανίχνευση και διόρθωση σύνθετων bug patterns στον JavaScript κώδικα. Με τη χρήση εργαλείων όπως το TensorFlow και το PyTorch, στόχος είναι η ανάπτυξη και προσομοίωση τουλάχιστον δύο αλγορίθμων για τη βελτίωση της ακρίβειας και της αποτελεσματικότητας στη διόρθωση σφαλμάτων, συμβάλλοντας στη μείωση του χρόνου και της ανθρώπινης παρέμβασης που απαιτείται στην ανάπτυξη λογισμικού.

1.1.1 Τι πρόκειται να μελετηθεί στην εργασία

Πρόκειται να μελετηθεί η διαδικασία της διόρθωσης σφαλμάτων σε προγράμματα JavaScript μέσω της εφαρμογής τεχνικών βαθιάς μάθησης. Θα εξεταστούν οι διάφοροι τύποι σφαλμάτων που εμφανίζονται στον JavaScript κώδικα, όπως συντακτικά, λογικά και runtime errors, καθώς και οι παραδοσιακές μέθοδοι διόρθωσης τους. Επιπλέον, θα διερευνηθούν τα μοντέλα βαθιάς μάθησης που είναι κατάλληλα για την ανίχνευση και διόρθωση αυτών των σφαλμάτων, εστιάζοντας σε αλγορίθμους όπως τα Deep Neural Networks (DNNs), Recurrent Neural Networks (RNNs) και Convolutional Neural Networks (CNNs). Μέσα από τη βιβλιογραφική ανασκόπηση, την προσομοίωση και την αξιολόγηση των αλγορίθμων, η εργασία θα αναδείξει τη δυνατότητα των τεχνολογιών αυτών να προσφέρουν αποτελεσματικές λύσεις σε πραγματικά προβλήματα της ανάπτυξης λογισμικού, με στόχο την αυτόματη διόρθωση σφαλμάτων και τη βελτίωση της ποιότητας του παραγόμενου κώδικα.

1.1.2 Η σημασία της διόρθωσης σφαλμάτων σε JavaScript

Η διόρθωση σφαλμάτων σε JavaScript έχει ιδιαίτερη σημασία, καθώς η JavaScript είναι μία από τις πιο διαδεδομένες γλώσσες για την ανάπτυξη εφαρμογών ιστού, με κύρια χρήση στο frontend αλλά και στο backend μέσω του Node.js. Τα σφάλματα σε JavaScript προγράμματα μπορεί να προκαλέσουν αστοχίες στις εφαρμογές, από την ανεπαρκή απόκριση του περιβάλλοντος χρήστη μέχρι σοβαρές διαρροές δεδομένων, καθώς και κενά ασφαλείας. Οι κυριότεροι τύποι σφαλμάτων που πρέπει να διορθωθούν περιλαμβάνουν τα συντακτικά (syntax errors), λογικά (logical errors) και τα runtime errors.

1. **Συντακτικά Σφάλματα (Syntax Errors):** Αυτά τα σφάλματα προκύπτουν από λάθη στη σύνταξη του κώδικα, όπως η έλλειψη συμβόλων (π.χ., αγκύλες ή ελληνικά κόμματα) και ασυμβατότητα τύπων. Τα συντακτικά σφάλματα εμποδίζουν την εκτέλεση του κώδικα και εντοπίζονται συνήθως κατά την αρχική φάση της εκτέλεσης. Τα εργαλεία βαθιάς μάθησης μπορούν να βοηθήσουν στην αυτόματη αναγνώριση μοτίβων που οδηγούν σε τέτοιες συντακτικές αστοχίες, επιτρέποντας την άμεση επισήμανση και διόρθωσή τους.

2. **Λογικά Σφάλματα (Logical Errors):** Τα λογικά σφάλματα εμφανίζονται όταν ο κώδικας εκτελείται χωρίς συντακτικά προβλήματα, αλλά δεν αποδίδει το επιθυμητό αποτέλεσμα. Για παράδειγμα, μια συνάρτηση υπολογισμού που επιστρέφει λανθασμένες τιμές λόγω λανθασμένων πράξεων. Τα λογικά σφάλματα είναι δύσκολο να εντοπιστούν, καθώς συχνά δεν παράγουν εμφανή σφάλματα κατά την εκτέλεση. Με τη χρήση βαθιάς μάθησης, τα μοντέλα μπορούν να εκπαιδευτούν σε μεγάλες ποσότητες παραδειγμάτων, ώστε να αναγνωρίζουν πότε μια έξοδος είναι απρόσμενη ή ασυνήθιστη, προτείνοντας πιθανά σημεία προς διόρθωση.

3. **Runtime Errors:** Αυτά τα σφάλματα εμφανίζονται κατά την εκτέλεση του προγράμματος και μπορεί να προκαλέσουν την απότομη διακοπή του ή τη μη κανονική συμπεριφορά του, συνήθως λόγω ανεπαρκών ελέγχων στην είσοδο δεδομένων ή στη διαχείριση εξαιρέσεων. Παραδείγματα περιλαμβάνουν το Null Pointer Exception και το TypeError. Οι αλγόριθμοι βαθιάς μάθησης μπορούν να εντοπίσουν πρότυπα σφαλμάτων εκτέλεσης, μαθαίνοντας να αναγνωρίζουν την πιθανότητα σφάλματος από τον συνδυασμό δεδομένων και κώδικα που έχουν μάθει να αναλύουν από προηγούμενες εκτελέσεις.

Τεχνικές Διόρθωσης μέσω Μοντέλων Βαθιάς Μάθησης: Η σημασία της διόρθωσης σφαλμάτων με τη βοήθεια βαθιάς μάθησης έγκειται στην ικανότητα αυτών των μοντέλων να διαχειρίζονται μεγάλο όγκο δεδομένων και να ανιχνεύουν πρότυπα που δεν είναι προφανή στους προγραμματιστές. Μοντέλα όπως τα

Convolutional Neural Networks (CNNs) μπορούν να ανιχνεύουν οπτικά μοτίβα στον κώδικα, ενώ τα Recurrent Neural Networks (RNNs) και οι παραλλαγές τους, όπως τα Long Short-Term Memory (LSTM) μοντέλα, είναι κατάλληλα για την επεξεργασία ακολουθιών κώδικα, διευκολύνοντας τον εντοπισμό σφαλμάτων σε σημεία που απαιτούν προηγούμενο πλαίσιο (contextual awareness). Η σημασία της διόρθωσης σφαλμάτων με βαθιά μάθηση έγκειται στην αυτοματοποίηση αυτής της διαδικασίας, προσφέροντας λύσεις που μειώνουν τον ανθρώπινο παράγοντα και τις χρονοβόρες διαδικασίες εντοπισμού σφαλμάτων, και έτσι ενισχύουν την αξιοπιστία και την ασφάλεια των εφαρμογών JavaScript.

1.2 Αντικείμενο της μελέτης

Το αντικείμενο της μελέτης αυτής είναι η ανάλυση και εφαρμογή τεχνικών βαθιάς μάθησης για τον εντοπισμό και τη διόρθωση σφαλμάτων σε προγράμματα JavaScript. Η εργασία εστιάζει στην κατηγοριοποίηση των τύπων σφαλμάτων που εμφανίζονται συχνά στη γλώσσα JavaScript, όπως τα συντακτικά, τα λογικά και τα runtime errors, και στη μελέτη των αλγορίθμων που μπορούν να χρησιμοποιηθούν για την αυτόματη ανίχνευση και διόρθωσή τους. Σκοπός είναι να εξεταστούν τα νευρωνικά δίκτυα που προσφέρονται για την επίλυση τέτοιων προβλημάτων, όπως τα Deep Neural Networks (DNNs), τα Recurrent Neural Networks (RNNs) και τα Convolutional Neural Networks (CNNs), και να εκτιμηθεί η αποτελεσματικότητά τους μέσω προσομοιώσεων σε πραγματικά ή συνθετικά δεδομένα κώδικα. Μέσω της προσομοίωσης αυτών των αλγορίθμων, η μελέτη επιδιώκει να προσδιορίσει πώς οι τεχνικές βαθιάς μάθησης μπορούν να συμβάλουν στη μείωση του ανθρώπινου παράγοντα στην ανάπτυξη λογισμικού, παρέχοντας πιο γρήγορες και ακριβείς λύσεις διόρθωσης σφαλμάτων. Επίσης, διερευνάται η δυνατότητα αξιοποίησης αυτών των τεχνικών για τη βελτίωση της ποιότητας του παραγόμενου κώδικα και της ασφάλειας των εφαρμογών, παρουσιάζοντας μια καινοτόμα προσέγγιση στη διαχείριση σφαλμάτων JavaScript.

1.2.1 Περιγραφή του προβλήματος και των προσεγγίσεων

Το πρόβλημα που επιχειρεί να επιλύσει η παρούσα μελέτη αφορά τη δυσκολία εντοπισμού και διόρθωσης σφαλμάτων σε προγράμματα JavaScript, μιας γλώσσας που χρησιμοποιείται εκτεταμένα για την ανάπτυξη διαδικτυακών εφαρμογών. Η JavaScript, λόγω της δυναμικής της φύσης και της χαλαρής διαχείρισης τύπων, είναι επιρρεπής σε πολλά σφάλματα που μπορεί να επηρεάσουν την απόδοση, τη λειτουργικότητα και την ασφάλεια των εφαρμογών. Τα συντακτικά, λογικά και runtime errors αποτελούν συχνές πηγές προβλημάτων, με αποτέλεσμα να απαιτούνται εξειδικευμένες μέθοδοι και σημαντικός χρόνος για την ανίχνευση και τη διόρθωσή τους. Παραδοσιακές τεχνικές, όπως οι χειροκίνητοι έλεγχοι και τα εργαλεία debugging, δεν είναι πάντα αποτελεσματικά για την αντιμετώπιση πολύπλοκων σφαλμάτων, ειδικά σε μεγάλα έργα όπου το μέγεθος και η πολυπλοκότητα του κώδικα καθιστούν δύσκολη την αναγνώριση των πραγματικών αιτίων των προβλημάτων.

1.3 Δομή της εργασίας

1.3.1 Γρήγορη περιγραφή των κεφαλαίων και του περιεχομένου τους

Κεφάλαιο 1: Εισαγωγή – Παρουσιάζεται το αντικείμενο της μελέτης, ο σκοπός της εργασίας και η σημασία της διόρθωσης σφαλμάτων σε JavaScript μέσω βαθιάς μάθησης. Περιγράφονται επίσης οι στόχοι και οι ερευνητικές ερωτήσεις που επιχειρεί να απαντήσει η εργασία.

Κεφάλαιο 2: Βασικές Έννοιες – Παρέχεται θεωρητική θεμελίωση των κύριων εννοιών που αφορούν τη γλώσσα JavaScript και τα διάφορα είδη σφαλμάτων της (συντακτικά, λογικά, runtime errors). Στη συνέχεια, γίνεται εισαγωγή στις τεχνικές βαθιάς μάθησης και στα νευρωνικά δίκτυα, όπως τα DNNs, RNNs και CNNs, και πώς αυτά μπορούν να αξιοποιηθούν για την ανίχνευση και διόρθωση σφαλμάτων.

Κεφάλαιο 3: Λύσεις Διόρθωσης Σφαλμάτων μέσω Βαθιάς Μάθησης – Εξετάζονται οι διαθέσιμες προσεγγίσεις διόρθωσης σφαλμάτων μέσω αλγορίθμων βαθιάς μάθησης. Αναλύονται μοντέλα, όπως τα DNNs, RNNs και CNNs, καθώς και τεχνικές NLP που μπορούν να συνδυαστούν για την αντιμετώπιση σφαλμάτων στον κώδικα.

Κεφάλαιο 4: Μεθοδολογία – Παρουσιάζεται η μεθοδολογία που ακολουθείται για την υλοποίηση της έρευνας. Περιγράφονται τα εργαλεία και τα δεδομένα που χρησιμοποιήθηκαν, η προετοιμασία του dataset, καθώς και η διαδικασία εκπαίδευσης και αξιολόγησης των αλγορίθμων βαθιάς μάθησης που επιλέχθηκαν.

Κεφάλαιο 5: Αποτελέσματα – Παρουσιάζονται τα αποτελέσματα των προσομοιώσεων των αλγορίθμων διόρθωσης σφαλμάτων, συνοδευόμενα από αξιολόγηση της απόδοσής τους με μέτρα όπως το Accuracy, Precision και Recall. Επίσης, παρέχονται παραδείγματα σφαλμάτων που διορθώθηκαν επιτυχώς από τα μοντέλα.

Κεφάλαιο 6: Συμπεράσματα και Μελλοντικές Εργασίες – Καταγράφονται τα τελικά συμπεράσματα από την εφαρμογή των μεθόδων βαθιάς μάθησης στη διόρθωση σφαλμάτων JavaScript και γίνονται προτάσεις για μελλοντικές έρευνες και πιθανές βελτιώσεις. Αναλύονται επίσης οι προκλήσεις και οι περιορισμοί που προέκυψαν κατά τη διάρκεια της μελέτης.

Κεφάλαιο 7: Βιβλιογραφία – Περιέχει όλες τις πηγές και τα επιστημονικά άρθρα που χρησιμοποιήθηκαν ως αναφορές στην εργασία.

2 Βασικές Έννοιες

2.1 JavaScript και σφάλματα

Η JavaScript είναι μια γλώσσα προγραμματισμού που χρησιμοποιείται ευρέως για την ανάπτυξη διαδραστικών και δυναμικών εφαρμογών στο διαδίκτυο. Παρά την ισχύ της, η φύση της JavaScript την καθιστά επιρρεπή σε ποικίλα σφάλματα, τα οποία μπορεί να προκύψουν λόγω της δυναμικής της διαχείρισης τύπων, της ασάφειας στη σύνταξη και της χαλαρής τυποποίησης, που μπορεί να οδηγήσει σε απρόβλεπτες συμπεριφορές κατά την εκτέλεση του κώδικα.

2.1.1 Τι είναι τα σφάλματα σε προγράμματα JavaScript (syntactical, logical, runtime errors)

Όπως έχει ήδη αναφερθεί τα σφάλματα σε προγράμματα JavaScript χωρίζονται σε τρεις κύριες κατηγορίες: συντακτικά σφάλματα, λογικά σφάλματα και runtime σφάλματα. Κάθε κατηγορία έχει τα δικά της χαρακτηριστικά και προκαλείται από διαφορετικές αιτίες. Ακολουθεί ανάλυση αυτών των σφαλμάτων με παραδείγματα για κάθε τύπο:

2.1.2 Συντακτικά Σφάλματα (Syntax Errors)

Τα συντακτικά σφάλματα προκύπτουν όταν ο κώδικας περιέχει λάθη στη σύνταξη, με αποτέλεσμα να μην μπορεί να εκτελεστεί. Αυτά τα σφάλματα εμφανίζονται κατά τη διαδικασία της αρχικής μεταγλώττισης (parsing) και πρέπει να διορθωθούν πριν ο κώδικας προχωρήσει σε εκτέλεση.

Παράδειγμα συντακτικού λάθους :

```
let x = 10

console.log(x;
```

Στο παραπάνω παράδειγμα, λείπει το ; στο τέλος της πρώτης γραμμής και η παρένθεση κλεισίματος) στην εντολή console.log(x;. Αυτά τα λάθη θα αποτρέψουν την εκτέλεση του κώδικα και θα εμφανιστεί μήνυμα λάθους στο περιβάλλον εκτέλεσης.

Μη έγκυρο όνομα μεταβλητής:

```
let 1number = 10;
```

Σφάλμα: Τα ονόματα μεταβλητών δεν μπορούν να ξεκινούν με αριθμό.

Λάθος χρήση λέξεων-κλειδιών:

```
const return = 10;
```

Σφάλμα: Το return είναι δεσμευμένη λέξη.

2.1.3 Λογικά Σφάλματα (Logical Errors):

Τα λογικά σφάλματα εμφανίζονται όταν ο κώδικας δεν εκτελείται με τον επιθυμητό τρόπο, παρότι η σύνταξη είναι σωστή. Αυτά τα σφάλματα είναι δύσκολο να εντοπιστούν, καθώς δεν προκαλούν άμεσα λάθη κατά την εκτέλεση. Συχνά, ο κώδικας «τρέχει» χωρίς προβλήματα, αλλά τα αποτελέσματα είναι εσφαλμένα.

```
function calculateArea(width, height) {

   return width + height; // Λανθασμένη πράξη, πρέπει να είναι πολλαπλασιασμός

}



let area = calculateArea(5, 10);

console.log(area); // Επιστρέφει 15 αντί για 50
```

Στο παράδειγμα, ο κώδικας εκτελείται χωρίς σφάλμα, αλλά το αποτέλεσμα είναι λανθασμένο επειδή χρησιμοποιήθηκε η πράξη πρόσθεσης αντί για τον πολλαπλασιασμό. Το calculateArea(5, 10) θα επιστρέψει το 15 αντί για το σωστό αποτέλεσμα, που είναι 50.

Λανθασμένος αλγόριθμος:

```
function isEven(num) {

   return num % 2 === 1; // Λάθος λογική

}

console.log(isEven(4)); // Επιστρέφει true αντί για false
```

Ανακριβής υπολογισμός:

let total = 100 - 50 * 2; // Αναμενόμενο: 0, αποτέλεσμα: 0

2.1.4 Runtime Errors

Τα runtime errors εμφανίζονται κατά τη διάρκεια της εκτέλεσης του προγράμματος, όταν μια εντολή προσπαθεί να εκτελέσει κάτι που δεν είναι δυνατό ή δεν είναι έγκυρο. Τα σφάλματα αυτά μπορεί να προκύψουν για διάφορους λόγους, όπως η αναφορά σε μια μη διαθέσιμη μεταβλητή ή η προσπάθεια κλήσης μιας συνάρτησης που δεν υπάρχει.

let numbers = [1, 2, 3];

console.log(numbers[5]); // Προσπάθεια πρόσβασης σε μη διαθέσιμο στοιχείο (undefined)

let name;

console.log(name.toUpperCase()); // TypeError: Cannot read property 'toUpperCase' of undefined

Στο πρώτο παράδειγμα, ο κώδικας προσπαθεί να προσπελάσει ένα στοιχείο στον πίνακα numbers που δεν υπάρχει, με αποτέλεσμα να επιστρέψει undefined. Στο δεύτερο παράδειγμα, το σφάλμα TypeError προκύπτει επειδή γίνεται προσπάθεια να εφαρμοστεί η μέθοδος toUpperCase() σε μια μη αρχικοποιημένη μεταβλητή name, η οποία είναι undefined.

Πρόσβαση σε μη καθορισμένη μεταβλητή:

console.log(myVar); // ReferenceError: myVar is not defined

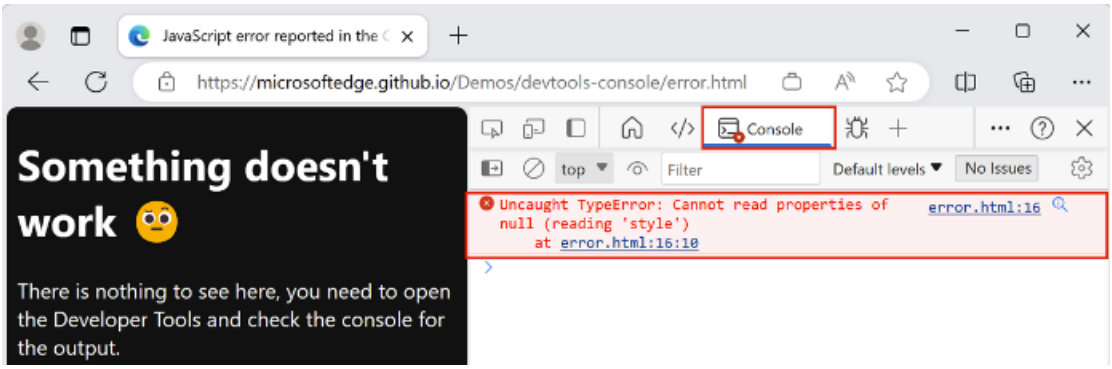
Λάθος χρήση τύπου δεδομένων:

let obj = null;

console.log(obj.name); // TypeError: Cannot read properties of null

Αυτά τα τρία είδη σφαλμάτων επηρεάζουν τη λειτουργικότητα των JavaScript εφαρμογών και απαιτούν συστηματικές προσεγγίσεις για τον εντοπισμό και τη

διόρθωσή τους. Οι τεχνικές βαθιάς μάθησης που εξετάζονται στην εργασία αυτή μπορούν να βοηθήσουν στην αναγνώριση και την αυτόματη διόρθωση αυτών των σφαλμάτων, μειώνοντας το χρόνο και την προσπάθεια που απαιτείται για τη βελτίωση της ποιότητας του κώδικα.



Σχήμα 1 : Σφάλμα στην κονσόλα

2.1.4.1 Συχνά Εμφανιζόμενα Σφάλματα Runtime στην JavaScript

Στην JavaScript εμφανίζονται συχνά συγκεκριμένοι τύποι σφαλμάτων runtime που μπορούν να επηρεάσουν σημαντικά τη λειτουργία των εφαρμογών. Ένα χαρακτηριστικό παράδειγμα είναι τα σφάλματα ασύγχρονων λειτουργιών:

```
function fetchUserData() {

  let userData = null

  // Λανθασμένη χρήση ασύγχρονης λειτουργίας

  fetch('/api/user').then(response => {

    userData = response.json()

  })

  return userData // Επιστρέφει πάντα null

}
```

Στο παραπάνω παράδειγμα, η συνάρτηση επιστρέφει null επειδή η ασύγχρονη λειτουργία fetch δεν έχει προλάβει να ολοκληρωθεί. Η σωστή υλοποίηση είναι:

```
async function fetchUserData() {
  const response = await fetch('/api/user')
  const userData = await response.json()
  return userData
}
```

Ένα άλλο συχνό σφάλμα είναι η λανθασμένη χρήση των callbacks σε event listeners:

```
class Menu {
  constructor() {
    this.status = 'closed'
    document.querySelector('.button')
      .addEventListener('click', function() {
        // To this αναφέρεται στο button αντί για την κλάση
        this.status = 'open'
    })
  }
}
```

Η διόρθωση γίνεται με χρήση arrow function που διατηρεί το σωστό context:

```
class Menu {
  constructor() {
    this.status = 'closed'
    document.querySelector('.button')
      .addEventListener('click', () => {
        this.status = 'open'
    })
  }
}
```

2.1.4.2 Σφάλματα Χειρισμού Μνήμης

Τα σφάλματα διαρροής μνήμης (memory leaks) είναι ιδιαίτερα σημαντικά στην JavaScript. Ένα τυπικό παράδειγμα είναι η συσσώρευση εγγραφών σε πίνακες χωρίς καθαρισμό:

```
function processLogs() {
  let logs = []
  setInterval(() => {
    logs.push(Date.now())
    // Ο πίνακας μεγαλώνει συνεχώς
  }, 100)
}
```

Η σωστή υλοποίηση περιλαμβάνει καθαρισμό των παλιών εγγραφών:

```
function processLogs() {
  let logs = []
  setInterval(() => {
    logs.push(Date.now())
    if (logs.length > 100) {
      logs = logs.slice(-100)
    }
  }, 100)
}
```

Αυτά τα παραδείγματα δείχνουν πώς συνηθισμένα σφάλματα runtime μπορούν να εντοπιστούν και να διορθωθούν. Η κατανόηση των μοτίβων εμφάνισης σφαλμάτων βοηθά στην ανάπτυξη πιο αξιόπιστου κώδικα.

2.1.4.3 Σφάλματα Χειρισμού Πινάκων και Αντικειμένων

Ένα συχνό σφάλμα στη JavaScript είναι η λανθασμένη χρήση των μεθόδων πινάκων. Για παράδειγμα:

```
function processItems(items) {
  // Λανθασμένη χρήση του map
  items.map(item => {
    console.log(item)
    // Δεν επιστρέφει τίποτα, άρα δημιουργεί πίνακα με undefined
  })
}
```

Η σωστή υλοποίηση εξαρτάται από το σκοπό - αν θέλουμε μόνο να εκτελέσουμε μια λειτουργία για κάθε στοιχείο χρησιμοποιούμε forEach:

```
function processItems(items) {
  items.forEach(item => {
    console.log(item)
  })
}
```

Ένα άλλο σημαντικό σφάλμα σχετίζεται με τη μετάλλαξη αντικειμένων:

```
const state = {
  user: {
    name: 'John',
    settings: {
      theme: 'dark'
    }
  }
}

function updateTheme(newTheme) {
  // Λανθασμένη άμεση μετάλλαξη του state
  state.user.settings.theme = newTheme
}
```

Η σωστή προσέγγιση είναι να δημιουργούμε νέο αντικείμενο:

```
function updateTheme(newTheme) {
  return {
    ...state,
    user: {
      ...state.user,
      settings: {
        ...state.user.settings,
        theme: newTheme
      }
    }
  }
}
```

2.1.4.4 Σφάλματα σε Promises και Χειρισμό Εξαιρέσεων

Η λανθασμένη διαχείριση των Promises μπορεί να οδηγήσει σε μη εμφανή σφάλματα:

```
function getData() {
  fetch('/api/data')
    .then(response => response.json())
    // Δεν υπάρχει χειρισμός σφαλμάτων
    .then(data => {
      processData(data)
```

```
    })
}
```

Η ορθή υλοποίηση περιλαμβάνει χειρισμό σφαλμάτων:

```
async function getData() {
  try {
    const response = await fetch('/api/data')
    if (!response.ok) {
      throw new Error('Network response was not ok')
    }
    const data = await response.json()
    return processData(data)
  } catch (error) {
    console.error('Error fetching data:', error)
    // Κατάλληλος χειρισμός σφάλματος
  }
}
```

2.1.4.5 Σφάλματα Τύπου (Type Errors)

Τα σφάλματα τύπου εμφανίζονται όταν επιχειρούμε να εκτελέσουμε μια λειτουργία με έναν τύπο δεδομένων που δεν την υποστηρίζει. Αυτό είναι συχνό στη JavaScript λόγω της δυναμικής φύσης της.

Παράδειγμα:

let number = 42; number.toUpperCase(); // TypeError: number.toUpperCase is not a function

Στο παραπάνω παράδειγμα, προσπαθούμε να εφαρμόσουμε τη μέθοδο toUpperCase() που είναι διαθέσιμη μόνο για strings, σε έναν αριθμό. Η JavaScript προσπαθεί να βρει αυτή τη μέθοδο στο πρωτότυπο του αριθμού, αλλά δεν υπάρχει, οπότε παράγεται TypeError.

2.1.4.6 Ασύγχρονα Σφάλματα (Asynchronous Errors)

Τα ασύγχρονα σφάλματα προκύπτουν κατά την εκτέλεση ασύγχρονων λειτουργιών όπως callbacks, promises και async/await. Αυτά τα σφάλματα είναι συχνά δύσκολο να εντοπιστούν γιατί η εκτέλεση συνεχίζεται και το σφάλμα μπορεί να εμφανιστεί αργότερα.

Παράδειγμα χωρίς σωστό χειρισμό σφάλματος:

fetch('https://api.nonexistent.com/data') .then(response => response.json()) .then(data => console.log(data)); // Δεν υπάρχει χειρισμός σφάλματος με .catch()

Παράδειγμα με σωστό χειρισμό σφάλματος:

```
fetch('https://api.nonexistent.com/data')
 .then(response => {
  if (!response.ok) {
   throw new Error('Network response was not ok');
  }
  return response.json();
 })
 .then(data => console.log(data))
 .catch(error => {
  console.error('Σφάλμα κατά τη λήψη των δεδομένων:', error);
 });
```

2.1.4.7 Σφάλματα Στοίβας (Stack Overflow Errors)

Τα σφάλματα στοίβας συμβαίνουν όταν υπερφορτώνεται η στοίβα κλήσεων από υπερβολικές αναδρομικές κλήσεις, οδηγώντας σε υπέρβαση του ορίου μεγέθους της στοίβας.

```
function recurse() {

  recurse(); // Αναδρομή χωρίς συνθήκη τερματισμού

}

recurse(); // RangeError: Maximum call stack size exceeded
```

Στο παραπάνω παράδειγμα, η συνάρτηση `recurse()` καλεί επανειλημμένα τον εαυτό της χωρίς κάποια συνθήκη τερματισμού, προκαλώντας υπερχείλιση της στοίβας.

2.1.4.8 Σφάλματα Κυκλικής Εξάρτησης (Circular Dependency Errors)

Τα σφάλματα κυκλικής εξάρτησης εμφανίζονται όταν δύο ή περισσότερες μονάδες του προγράμματος εξαρτώνται η μία από την άλλη, δημιουργώντας έναν κύκλο εξαρτήσεων που μπορεί να οδηγήσει σε απρόβλεπτη συμπεριφορά.

Παράδειγμα:

```
// moduleA.js

import { functionB } from './moduleB.js';


export function functionA() {

  return functionB() + 1;

}


// moduleB.js

import { functionA } from './moduleA.js';
```

```
export function functionB() {

  return functionA() + 1;

}
```

Αυτή η κυκλική εξάρτηση μπορεί να οδηγήσει σε σφάλματα κατά τη φόρτωση των modules ή σε απρόβλεπτη συμπεριφορά κατά την εκτέλεση.

2.2 Τεχνικές διόρθωσης σφαλμάτων

Παρακάτω θα αναλύσουμε τις διάφορες τεχνικές διόρθωσης των λαθών.

2.2.1 Παραδοσιακές προσεγγίσεις διόρθωσης σφαλμάτων σε γλώσσες προγραμματισμού

Οι παραδοσιακές προσεγγίσεις διόρθωσης σφαλμάτων σε γλώσσες προγραμματισμού αποτελούν θεμελιώδες κομμάτι της ανάπτυξης λογισμικού, βοηθώντας τους προγραμματιστές να εντοπίζουν και να διορθώνουν σφάλματα κατά τη διάρκεια του κύκλου ζωής της ανάπτυξης. Οι μέθοδοι αυτές συνήθως περιλαμβάνουν χειροκίνητο έλεγχο του κώδικα, τη χρήση εργαλείων εντοπισμού σφαλμάτων (debuggers) και τη διεξαγωγή δοκιμών (testing) για τη διασφάλιση της ποιότητας του κώδικα. Ακολουθούν οι κύριες παραδοσιακές τεχνικές διόρθωσης σφαλμάτων:

Χειροκίνητη Επιθεώρηση Κώδικα (Code Review)

Η χειροκίνητη επιθεώρηση κώδικα αποτελεί μια από τις πιο διαδεδομένες και απλές μεθόδους διόρθωσης σφαλμάτων, όπου προγραμματιστές ή ομάδες επιθεωρούν τον κώδικα γραμμή προς γραμμή για να εντοπίσουν συντακτικά ή λογικά λάθη. Μέσω της επιθεώρησης, μπορούν να εντοπιστούν τα σφάλματα που δεν εμφανίζονται κατά την εκτέλεση, αλλά μπορεί να επηρεάσουν τη λογική ή τη δομή του κώδικα. Αυτή η μέθοδος αν και αποτελεσματική, είναι χρονοβόρα και μπορεί να γίνει αναποτελεσματική σε μεγάλα έργα.

Παράδειγμα:

Πρόβλημα: Ένας προγραμματιστής γράφει έναν βρόχο χωρίς να προσθέσει την εντολή αύξησης:

```
for (let i = 0; i < 10;) {

   console.log(i);

}
```

Διόρθωση: Κατά την επιθεώρηση, εντοπίζεται η έλλειψη και προστίθεται η εντολή i++:

```
for (let i = 0; i < 10; i++) {

   console.log(i);
```

}

Εργαλεία Εντοπισμού Σφαλμάτων (Debugging Tools)

Τα εργαλεία εντοπισμού σφαλμάτων, ή debuggers, επιτρέπουν στον προγραμματιστή να εκτελεί τον κώδικα γραμμή προς γραμμή, να παρακολουθεί τις μεταβλητές και να εντοπίζει το σημείο στο οποίο παρουσιάζεται το σφάλμα. Τα εργαλεία αυτά παρέχουν τη δυνατότητα να "παγώνουν" (breakpoints) τον κώδικα σε συγκεκριμένα σημεία και να εξετάζουν τη ροή της εκτέλεσης. Παραδείγματα εργαλείων debugging για JavaScript περιλαμβάνουν το DevTools στον Chrome και το Node.js Debugger. Αν και τα εργαλεία debugging είναι ισχυρά, απαιτούν σημαντική κατανόηση του κώδικα από τον προγραμματιστή και δεν εγγυώνται την εύρεση όλων των σφαλμάτων.

Παράδειγμα:

Πρόβλημα: Μια μεταβλητή δεν αυξάνεται σωστά. Χρησιμοποιώντας το Chrome DevTools, ο προγραμματιστής προσθέτει ένα breakpoint στον κώδικα:

```
function increment(x) {

   return x + 1;

}

let result = increment(5);
```

Λύση: Το εργαλείο δείχνει την τρέχουσα τιμή του x κατά την εκτέλεση, και ο προγραμματιστής διαπιστώνει ότι πρέπει να αλλάξει τη λογική της συνάρτησης.

Δοκιμές Μονάδων (Unit Testing)

Οι δοκιμές μονάδων περιλαμβάνουν τη δημιουργία μικρών, αυτόνομων τεστ που ελέγχουν τις επιμέρους λειτουργίες και μονάδες του κώδικα για σφάλματα. Οι δοκιμές αυτές γράφονται συχνά από τους ίδιους τους προγραμματιστές κατά την ανάπτυξη και επιτρέπουν την επαλήθευση ότι κάθε τμήμα του κώδικα λειτουργεί σωστά. Οι βιβλιοθήκες όπως το Jest και το Mocha χρησιμοποιούνται ευρέως για δοκιμές μονάδων στη JavaScript. Αν και οι δοκιμές μονάδων καλύπτουν τα μικρότερα μέρη του κώδικα, δεν είναι πάντα αρκετές για την εντόπιση πιο σύνθετων, λογικών σφαλμάτων που μπορεί να προκύψουν από τη συνδυασμένη λειτουργία πολλών μονάδων.

Παράδειγμα:

Πρόβλημα: Η συνάρτηση επιστρέφει λανθασμένο αποτέλεσμα:

```
function calculateArea(width, height) {
   return width + height; // Λάθος λογική
}
test('calculateArea', () => {
   expect(calculateArea(5, 10)).toBe(50);
});
```

Το test αποτυγχάνει, οπότε ο προγραμματιστής διορθώνει τη συνάρτηση:

```
return width * height;
```

Δοκιμές Ενσωμάτωσης και Συστημικές Δοκιμές (Integration and System Testing)

Οι δοκιμές ενσωμάτωσης εξετάζουν την αλληλεπίδραση μεταξύ διαφορετικών τμημάτων του λογισμικού, διασφαλίζοντας ότι όλες οι μονάδες λειτουργούν ομαλά μαζί. Αντίστοιχα, οι συστημικές δοκιμές ελέγχουν ολόκληρο το σύστημα για να διαπιστωθεί αν πληροί τις απαιτήσεις. Αυτές οι δοκιμές βοηθούν στην ανίχνευση σφαλμάτων που δεν εμφανίζονται κατά τη δοκιμή μεμονωμένων μονάδων, όπως τα σφάλματα διεπαφής και τα σφάλματα που σχετίζονται με την αλληλεπίδραση μεταξύ πολλαπλών λειτουργιών. Ωστόσο, η εφαρμογή τους μπορεί να απαιτεί πολύ χρόνο και πόρους, ειδικά σε μεγάλα έργα.

Στατική Ανάλυση Κώδικα (Static Code Analysis)

Η στατική ανάλυση κώδικα γίνεται πριν την εκτέλεση του προγράμματος και περιλαμβάνει τη χρήση εργαλείων που αναλύουν τον κώδικα για κοινά πρότυπα σφαλμάτων. Εργαλεία όπως το ESLint στη JavaScript μπορούν να εντοπίζουν συντακτικά λάθη, απλοποιήσεις και επικίνδυνες πρακτικές προγραμματισμού, βοηθώντας στην πρόληψη σφαλμάτων πριν ακόμα ο κώδικας εκτελεστεί. Παρά το γεγονός ότι η στατική ανάλυση παρέχει μια γρήγορη αξιολόγηση του κώδικα, ενδέχεται να μην καλύπτει πλήρως όλα τα είδη σφαλμάτων, όπως τα runtime και τα λογικά λάθη.

Παράδειγμα:

Πρόβλημα: Υπάρχει αχρησιμοποίητη μεταβλητή:

```
let unusedVar = 10;

console.log("Hello, world!");
```

Εργαλεία όπως το ESLint εμφανίζουν το μήνυμα:

'unusedVar' is defined but never used.

Ο προγραμματιστής αφαιρεί τη μεταβλητή.

Ανάλυση Αρχείων Καταγραφής (Log Analysis)

Τα αρχεία καταγραφής (logs) παρέχουν πληροφορίες για την εκτέλεση του κώδικα και τις συνθήκες που οδήγησαν σε σφάλματα. Οι προγραμματιστές μπορούν να εξετάζουν τα logs για να κατανοήσουν τη ροή των εντολών και να εντοπίσουν τα σημεία όπου εμφανίζονται προβλήματα. Η ανάλυση των logs είναι ιδιαίτερα χρήσιμη για την αντιμετώπιση runtime errors, αλλά μπορεί να είναι πολύπλοκη και χρονοβόρα, ειδικά σε εφαρμογές με μεγάλο όγκο δεδομένων καταγραφής.

Παράδειγμα:

Πρόβλημα: Σφάλμα TypeError εμφανίζεται μόνο σε συγκεκριμένα δεδομένα.

```
function parseJSON(input) {
   return JSON.parse(input);
}

parseJSON('invalid JSON string'); // Error
```

Ανάλυση των logs:

Error: Unexpected token i in JSON at position 0

Λύση: Προστίθεται έλεγχος πριν την ανάλυση:

```
try {
```

```
    return JSON.parse(input);

} catch (error) {

    console.error("Invalid JSON:", error);

}
```

Οι παραδοσιακές αυτές προσεγγίσεις, ενώ είναι αποτελεσματικές, απαιτούν σημαντική ανθρώπινη παρέμβαση και χρόνο. Η εργασία αυτή στοχεύει στη διερεύνηση του πώς η βαθιά μάθηση μπορεί να αυτοματοποιήσει και να βελτιώσει τη διαδικασία εντοπισμού και διόρθωσης σφαλμάτων, παρέχοντας μια πιο καινοτόμο και αποδοτική λύση για τα προβλήματα που αντιμετωπίζουν οι προγραμματιστές.

2.3 Εισαγωγή στα Νευρωνικά Δίκτυα και τη Βαθιά Μάθηση

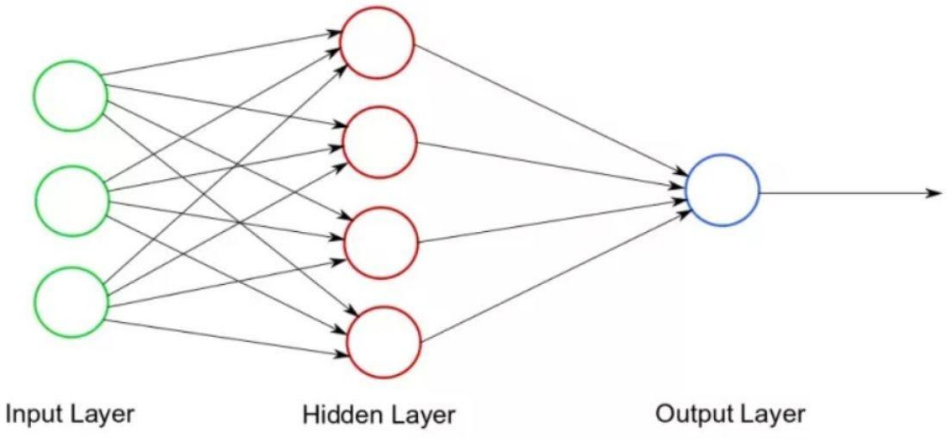
Η πρόοδος της τεχνητής νοημοσύνης (AI) και των νευρωνικών δικτύων (ΝΔ) έχει επιφέρει σημαντικές αλλαγές στην ανάπτυξη εφαρμογών που απαιτούν εξειδικευμένη αναγνώριση και ανάλυση προτύπων. Εμπνευσμένα από τον ανθρώπινο εγκέφαλο, τα Νευρωνικά Δίκτυα αποτελούν τον ακρογωνιαίο λίθο σύγχρονων τεχνικών μηχανικής μάθησης. Οι μέθοδοι βαθιάς μάθησης (Deep Learning) επιτρέπουν την αντιμετώπιση περίπλοκων προβλημάτων, τα οποία δεν μπορούσαν να επιλυθούν με παραδοσιακούς αλγορίθμους.

Η βαθιά μάθηση αποτελεί έναν διεπιστημονικό τομέα που ενοποιεί αρχές και ιδέες από διάφορες επιστήμες, όπως η τεχνητή νοημοσύνη, η γνωσιακή νευροεπιστήμη και η επεξεργασία σήματος. Βασίζεται στην ικανότητά της να εκμεταλλεύεται ιεραρχικές αρχιτεκτονικές για την εκμάθηση χαρακτηριστικών, αναγνώριση προτύπων και κατηγοριοποίηση. Τα τεχνητά νευρωνικά δίκτυα, οι αλγόριθμοι μη εποπτευόμενης μάθησης και οι τεχνικές επεξεργασίας δεδομένων έχουν οδηγήσει στη ραγδαία ανάπτυξη της βαθιάς μάθησης τα τελευταία χρόνια, καθιστώντας την απαραίτητη τόσο στην ακαδημαϊκή έρευνα όσο και στις βιομηχανικές εφαρμογές. Η κατανόηση της προέλευσης και της εξέλιξης αυτών των τεχνικών προσφέρει πολύτιμες προοπτικές για τη βελτίωση εφαρμογών, όπως η αυτόματη διόρθωση σφαλμάτων.[3]

2.3.1 Περιγραφή των Νευρωνικών Δικτύων (ΝΔ)

Τα Νευρωνικά Δίκτυα είναι συστήματα από συνδεδεμένους υπολογιστικούς κόμβους, που ονομάζονται «νευρώνες». Ο κάθε νευρώνας λαμβάνει μια σειρά εισόδων, τις οποίες επεξεργάζεται με βάση συνδεδεμένα βάρη, και παράγει μια έξοδο που επηρεάζεται από μια συνάρτηση ενεργοποίησης. Μέσα από τα επίπεδα (layers) που διαθέτουν, τα νευρωνικά δίκτυα καταφέρνουν να αναγνωρίζουν πρότυπα και σχέσεις στα δεδομένα που λαμβάνουν ως είσοδο. Τα πιο βασικά νευρωνικά δίκτυα περιλαμβάνουν ένα είδος προώθησης από εισόδους προς εξόδους (feed-forward), ενώ πιο σύνθετες αρχιτεκτονικές, όπως τα αναδρομικά νευρωνικά δίκτυα (Recurrent Neural Networks - RNNs), προσφέρουν δυνατότητες επεξεργασίας σειρών δεδομένων.

[3]https://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/10627/Kontoulis_Ioannis.pdf?sequence=1&isAllowed=y



Σχήμα 2 : Δομή νευρωνικού δικτύου

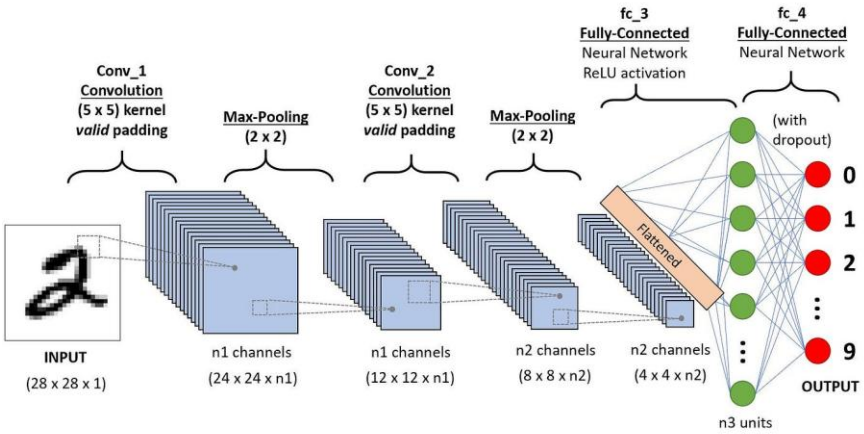
Πιο αναλυτικά : Τα Νευρωνικά Δίκτυα (Neural Networks - ΝΔ) αποτελούνται από στρώματα τεχνητών "νευρώνων", τα οποία προσομοιώνουν τη λειτουργία του ανθρώπινου εγκεφάλου. Οι νευρώνες είναι συνδεδεμένοι μέσω βαρών, τα οποία προσαρμόζονται κατά τη διαδικασία εκπαίδευσης ώστε να ελαχιστοποιούν το σφάλμα. Μια τυπική δομή νευρωνικού δικτύου περιλαμβάνει τρία κύρια μέρη: το στρώμα εισόδου, που δέχεται τα δεδομένα, τα κρυφά στρώματα, που πραγματοποιούν υπολογισμούς μέσω συνδυασμών βαρών και συναρτήσεων ενεργοποίησης, και το στρώμα εξόδου, που παράγει την τελική έξοδο με βάση τους υπολογισμούς. Η συνάρτηση ενεργοποίησης που χρησιμοποιείται στα νευρωνικά δίκτυα, όπως οι ReLU, Sigmoid και Tanh, εισάγει τη μη γραμμικότητα στη λειτουργία τους. Ο πιο συνηθισμένος αλγόριθμος εκπαίδευσης είναι ο backpropagation, ο οποίος χρησιμοποιείται σε συνδυασμό με τη Βαθμιδωτή Κατάβαση.

2.3.2 Είδη νευρωνικών δικτύων βαθιάς μάθησης(αρχιτεκτονική και λειτουργία)

Η ανάλυση της αρχιτεκτονικής και της λειτουργίας διαφόρων τύπων νευρωνικών δικτύων βαθιάς μάθησης επιτρέπει την κατανόηση των πλεονεκτημάτων και περιορισμών τους σε εφαρμογές που σχετίζονται με την ανίχνευση και την πρόβλεψη σφαλμάτων στο λογισμικό. Στην παρούσα εργασία, εξετάζονται οι εξής κύριες κατηγορίες δικτύων:

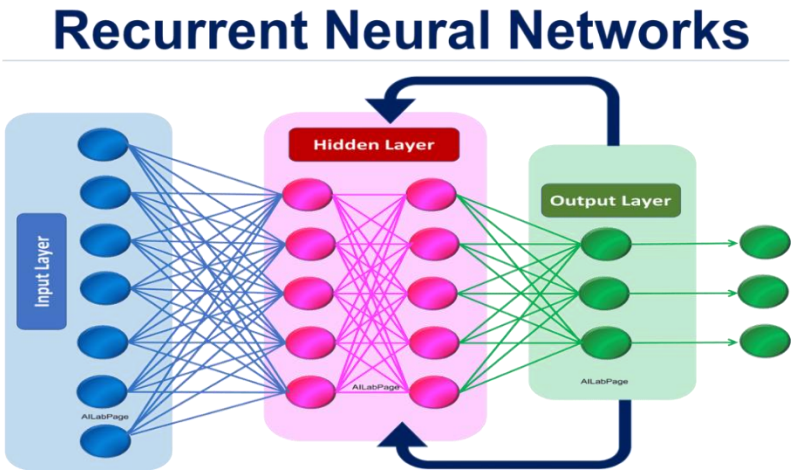
Τα Convolutional Neural Networks(CNN) βασίζονται σε μία αρχιτεκτονική που περιλαμβάνει στρώματα (convolutional layers), τα οποία εφαρμόζουν φίλτρα για την εξαγωγή χαρακτηριστικών, και στρώματα συγκέντρωσης (pooling layers), που μειώνουν τις διαστάσεις των δεδομένων, διατηρώντας τα κρίσιμα μοτίβα. Η

λειτουργία τους επικεντρώνεται στη σταδιακή αποκωδικοποίηση της δομής των δεδομένων εισόδου, καθιστώντας τα ιδανικά για την κατανόηση της χωρικής συσχέτισης σε δεδομένα, όπως εικόνες ή αναπαραστάσεις κώδικα μέσω γραφικών.



Σχήμα 3 : Convolutional neural network

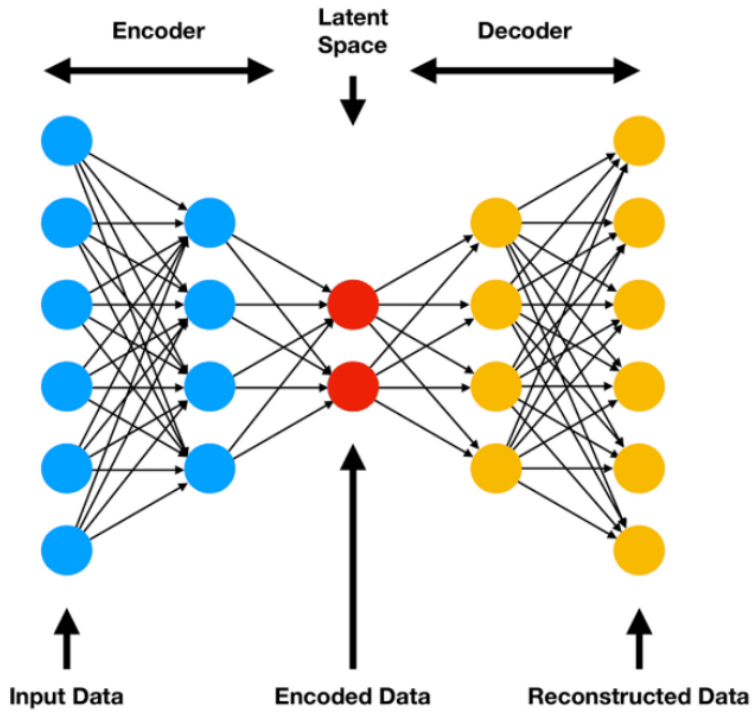
Τα Αναδρομικά Νευρωνικά Δίκτυα (RNN), και ιδιαίτερα οι βελτιωμένες εκδοχές τους, όπως τα Long Short-Term Memory (LSTM), χρησιμοποιούνται για τη μοντελοποίηση ακολουθιακών δεδομένων. Η αρχιτεκτονική τους περιλαμβάνει μονάδες που διατηρούν μνήμη και χρησιμοποιούν μηχανισμούς πύλης (gates) για την επιλογή πληροφοριών που διατηρούνται ή απορρίπτονται σε κάθε χρονικό βήμα. Η λειτουργία τους επικεντρώνεται στη διατήρηση συσχετισμών που εμφανίζονται σε μεγάλες χρονικές κλίμακες, γεγονός που τα καθιστά κατάλληλα για την ανάλυση λογισμικού, όπως η ανίχνευση σφαλμάτων σε σειριακά δεδομένα πηγαίου κώδικα.



Σχήμα 4 : RNN

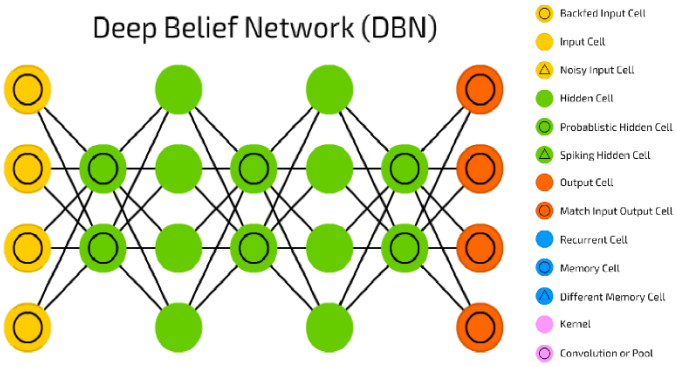
Τα Autoencoders χρησιμοποιούν μία συμμετρική αρχιτεκτονική που αποτελείται από έναν κωδικοποιητή (encoder) και έναν αποκωδικοποιητή (decoder). Η λειτουργία τους αφορά την εκμάθηση μιας συμπιεσμένης αναπαράστασης των δεδομένων, η

οποία αποτυπώνει τα πιο κρίσιμα χαρακτηριστικά. Στην παρούσα εργασία, αυτός ο τύπος δικτύου μπορεί να χρησιμοποιηθεί για την απομόνωση των πιο χρήσιμων μεταβλητών από δεδομένα κώδικα, διευκολύνοντας την ανίχνευση ανωμαλιών και προτύπων.



Σχήμα 5: Autoencoders

Τα Deep belief network (DBN) αποτελούνται από πολλαπλά επίπεδα Περιορισμένων Μηχανών Boltzmann's(RBMs). Κάθε επίπεδο εκπαιδεύεται ανεξάρτητα σε μία αρχική φάση, ενώ στη συνέχεια συνδέονται για την εκτέλεση εποπτευόμενων ή μη εποπτευόμενων εργασιών. Στο πλαίσιο της ανίχνευσης σφαλμάτων, η αρχιτεκτονική τους επιτρέπει την απομόνωση αφαιρετικών χαρακτηριστικών από σύνθετα δεδομένα, όπως οι δομές κώδικα.



Σχήμα 6 : DBN

Τα Δίκτυα Μετασχηματισμών(Transformers) βασίζονται σε μηχανισμούς(attention mechanisms) που επιτρέπουν την ταυτόχρονη ανάλυση όλων των στοιχείων της εισόδου. Η αρχιτεκτονική τους αποφεύγει τη διαδοχική επεξεργασία, όπως στα RNN, ενισχύοντας την ταχύτητα και την ακρίβεια. Στην εφαρμογή τους σε κώδικα, οι Transformers μπορούν να αναγνωρίσουν μοτίβα και σχέσεις σε μεγάλο εύρος δεδομένων, όπως η πρόβλεψη σφαλμάτων που σχετίζονται με αναντιστοιχίες λειτουργιών ή μεταβλητών.

Οι όροι που χρησιμοποιούνται στα είδη των νευρωνικών δικτύων βαθιάς μάθησης είναι οι εξής:

- CNN: Convolutional Neural Networks
- RNN: Recurrent Neural Networks
- LSTM: Long Short-Term Memory networks
- GRU: Gated Recurrent Units
- DBN: Deep Belief Networks
- Autoencoders: Autoencoders
- Transformers: Transformer Networks

2.3.3 Εισαγωγή στις μεθόδους Βαθιάς Μάθησης (DL)

Η βαθιά μάθηση αναφέρεται στην αξιοποίηση βαθιών νευρωνικών δικτύων, τα οποία αποτελούνται από πολλαπλά επίπεδα επεξεργασίας, επιτρέποντας στο σύστημα να αναγνωρίζει υψηλής πολυπλοκότητας πρότυπα. Σε ένα δίκτυο βαθιάς μάθησης, τα πρώτα επίπεδα εντοπίζουν βασικά χαρακτηριστικά των δεδομένων (όπως περιγράμματα σε εικόνες), ενώ τα επόμενα επίπεδα συγκεντρώνουν αυτές τις πληροφορίες για την ανάλυση πιο σύνθετων μορφών και σχέσεων. Η βαθιά μάθηση χρησιμοποιείται ευρέως σε προβλήματα αναγνώρισης εικόνων, επεξεργασίας

γλώσσας και ανάλυσης κώδικα. Τα μοντέλα DNNs, CNNs, RNNs και LSTM είναι βασικά εργαλεία για τέτοιες εφαρμογές :

- Διασυνδεδεμένα Νευρωνικά Δίκτυα (DNNs): Η βασική αρχιτεκτονική με πολλαπλά κρυφά επίπεδα, που χρησιμοποιούνται για ανάλυση προβλημάτων γενικής φύσης.

- Συνελικτικά Νευρωνικά Δίκτυα (CNNs): Σχεδιασμένα για επεξεργασία εικόνας, τα CNNs έχουν τη δυνατότητα να εντοπίζουν οπτικά πρότυπα με συνέλιξη (convolution) σε τμήματα της εικόνας.

- Αναδρομικά Νευρωνικά Δίκτυα (RNNs): Τα RNNs χρησιμοποιούνται στην επεξεργασία ακολουθιών δεδομένων, όπως η φυσική γλώσσα ή οι σειρές κώδικα, προσφέροντας "μνήμη" στο δίκτυο, ώστε να λαμβάνει υπόψη την ακολουθία προηγούμενων στοιχείων.

- Μοντέλα Μακράς Βραχείας Μνήμης (LSTM): Παραλλαγή των RNNs που ενισχύει την ικανότητα του δικτύου να διατηρεί πληροφορίες για μεγαλύτερες χρονικές αποστάσεις, επιτρέποντας πιο ακριβή ανάλυση και αποδοτικότερη απόδοση σε περίπλοκα προβλήματα.

Η εφαρμογή των νευρωνικών δικτύων και της βαθιάς μάθησης στην ανίχνευση σφαλμάτων σε γλώσσες προγραμματισμού, όπως η JavaScript, προσφέρει νέες δυνατότητες για την αυτόματη διόρθωση κώδικα. Μέσα από την εκπαίδευση των αλγορίθμων σε μεγάλα σύνολα δεδομένων κώδικα, οι τεχνικές αυτές έχουν τη δυνατότητα να εντοπίζουν ακόμα και λεπτομερείς αποκλίσεις που οδηγούν σε σφάλματα, προσφέροντας αξιόπιστες και αυτόματες λύσεις στη διαδικασία ανάπτυξης λογισμικού.

2.4 Αντίστοιχες Εργασίες στη Διόρθωση Σφαλμάτων με Βαθιά Μάθηση

Στο κεφάλαιο αυτό, θα γίνει ανάλυση πρόσφατων επιστημονικών εργασιών που έχουν συμβάλει στη διόρθωση σφαλμάτων σε JavaScript και σε άλλες γλώσσες μέσω της εφαρμογής βαθιάς μάθησης. Η αναφορά θα επικεντρωθεί σε συγκεκριμένα χαρακτηριστικά, όπως η αρχιτεκτονική του μοντέλου, τα στρώματα και οι παράμετροι που χρησιμοποιήθηκαν, οι αλγόριθμοι που εφαρμόστηκαν, τα δεδομένα που χρησιμοποιήθηκαν για εκπαίδευση και αξιολόγηση, καθώς και τα αποτελέσματα που προέκυψαν.

2.4.1 Εργαλεία και Αλγόριθμοι για Ανίχνευση και Διόρθωση Σφαλμάτων

Μία από τις πλέον αναγνωρισμένες προσεγγίσεις είναι το DeepBugs, το οποίο εφαρμόζει ένα γενικό πλαίσιο για τη δημιουργία ανιχνευτών σφαλμάτων μέσω εκπαίδευσης μοντέλων βαθιάς μάθησης. Το DeepBugs χρησιμοποιεί μετασχηματισμούς κώδικα για τη δημιουργία παραδειγμάτων σφαλμάτων από υπαρκτό κώδικα και εκπαιδεύει ένα δίκτυο feedforward με μία ενδιάμεση στρώση και μία έξοδο, που εκφράζει την πιθανότητα ύπαρξης σφάλματος. Η μεθοδολογία βασίζεται στη μετατροπή κώδικα σε διανύσματα με τη χρήση ενσωματώσεων (embeddings) από το Abstract Syntax Tree (AST) του κώδικα. Τα δεδομένα εκπαίδευσης περιλάμβαναν 150.000 αρχεία JavaScript, και τα αποτελέσματα έδειξαν ακρίβεια μεταξύ 84% και 94%, αποκαλύπτοντας πραγματικά σφάλματα στον κώδικα.[4] Μια άλλη σημαντική εργασία εξετάζει τη χρήση νευρωνικών δικτύων όπως τα LSTM για την πρόβλεψη και διόρθωση σφαλμάτων. Τα συγκεκριμένα μοντέλα εφαρμόστηκαν σε δεδομένα που περιλάμβαναν 60 χαρακτηριστικά ποιότητας κλάσεων λογισμικού. Η προσέγγιση αξιολόγησε την ακρίβεια, την εξισορροπημένη βαθμολογία F1 και άλλα μέτρα, με τα LSTM να υπερέχουν συγκριτικά με παραδοσιακούς αλγορίθμους μηχανικής μάθησης, επιτυγχάνοντας ακρίβεια 87%. Επιπλέον, το μοντέλο **DPSAM** εισήγαγε μηχανισμούς αυτο-προσοχής (self-attention mechanisms) για την εξαγωγή σημασιολογικών χαρακτηριστικών από Abstract Syntax Trees. Χρησιμοποιώντας επτά έργα ανοιχτού κώδικα, η μεθοδολογία αυτή πέτυχε καλύτερα αποτελέσματα σε προβλέψεις εντός και εκτός έργου (Within-Project και Cross-Project Defect Prediction), υπογραμμίζοντας τη σημασία της εξαγωγής πλούσιων σημασιολογικών πληροφοριών. [5]

2.4.2 Σύγκριση των Διαφορετικών Προσεγγίσεων

Στην έρευνα της βαθιάς μάθησης για τη διόρθωση σφαλμάτων, συχνά γίνεται σύγκριση μεταξύ διαφορετικών τύπων νευρωνικών δικτύων για να προσδιοριστεί η αποδοτικότητά τους σε συγκεκριμένα είδη σφαλμάτων. Για παράδειγμα, τα Convolutional Neural Networks (CNNs), αν και κυρίως χρησιμοποιούνται για ανάλυση εικόνας, έχουν εφαρμοστεί και στην ανίχνευση σφαλμάτων για τον εντοπισμό συγκεκριμένων μοτίβων στον κώδικα. Τα Recurrent Neural Networks (RNNs) και οι παραλλαγές τους, όπως τα LSTMs, έχουν αποδειχθεί πιο αποδοτικά για την ανίχνευση λογικών λαθών, δεδομένου ότι μπορούν να «θυμούνται» προηγούμενα βήματα στον κώδικα. Οι περισσότερες από τις μελέτες που

[4] https://www.software-lab.org/publications/DeepBugs_TR_Nov2017.pdf

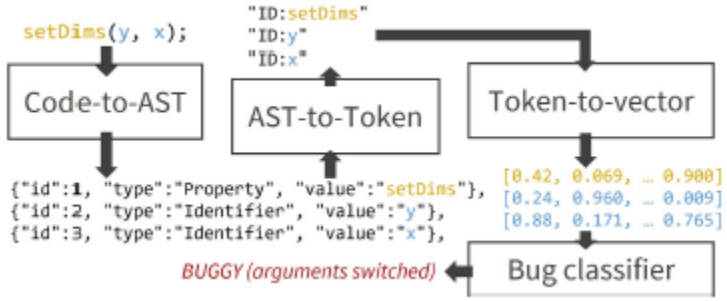
[5] https://www.mdpi.com/2673-2688/5/4/86

ασχολούνται με την ανίχνευση και διόρθωση σφαλμάτων έχουν δείξει ότι η βαθιά μάθηση μπορεί να μειώσει σημαντικά την ανθρώπινη παρέμβαση στη διαδικασία ανάλυσης κώδικα, παρέχοντας πιο ακριβείς λύσεις και μειώνοντας τον χρόνο ανίχνευσης των σφαλμάτων.[6]

[6] https://www.software-lab.org/publications/DeepBugs_TR_Nov2017.pdf

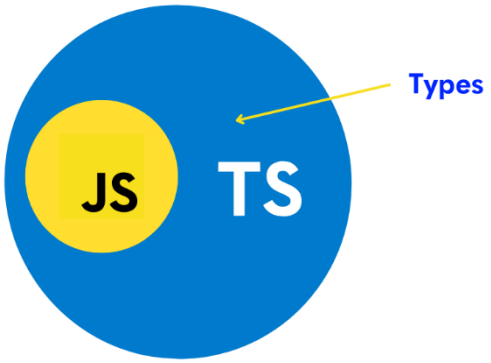
2.4.3 Σύγχρονες Ερευνητικές Προσεγγίσεις στην Ανίχνευση Σφαλμάτων JavaScript

Πρόσφατες ερευνητικές εργασίες έχουν προτείνει καινοτόμες προσεγγίσεις για την ανίχνευση σφαλμάτων στη JavaScript. Η πλατφόρμα DeepBugs, που αναπτύχθηκε από τους Pradel και Sen, χρησιμοποιεί τεχνικές βαθιάς μάθησης για να εντοπίσει συνηθισμένα σφάλματα προγραμματισμού. Το σύστημα εκπαιδεύεται σε μεγάλο όγκο σωστού κώδικα και μαθαίνει να αναγνωρίζει πιθανά σφάλματα μέσω της ανάλυσης των ονομάτων μεταβλητών και συναρτήσεων. Τα πειραματικά αποτελέσματα σε 150.000 αρχεία JavaScript έδειξαν ακρίβεια μεταξύ 84% και 94% στην ανίχνευση σφαλμάτων.



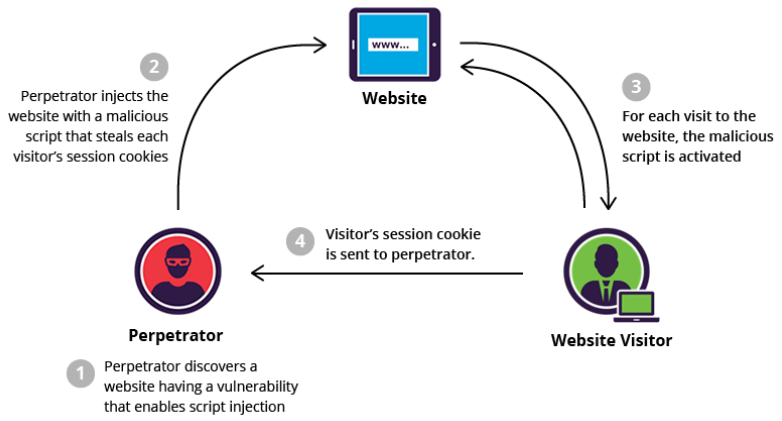
Σχήμα 7 : Deepbugs

Μια διαφορετική προσέγγιση προτάθηκε από την ερευνητική ομάδα της IBM με το σύστημα BugAID. Το σύστημα χρησιμοποιεί συνδυασμό στατικής και δυναμικής ανάλυσης για να εντοπίσει ασυνέπειες στη χρήση του DOM και σφάλματα ασύγχρονου προγραμματισμού. Η αξιολόγηση σε πραγματικές εφαρμογές ιστού έδειξε ότι το σύστημα μπορεί να εντοπίσει σφάλματα που διέφυγαν από συμβατικούς ελέγχους ποιότητας κώδικα. Το εργαλείο TypeScript της Microsoft προσφέρει μια εναλλακτική προσέγγιση μέσω στατικού συστήματος τύπων. Η έρευνα των Feldthaus και Møller έδειξε ότι η προσθήκη τύπων μπορεί να εντοπίσει το 15% των σφαλμάτων runtime σε μεγάλες εφαρμογές JavaScript πριν την εκτέλεση του κώδικα. Ωστόσο, η προσέγγιση αυτή απαιτεί σημαντική προσπάθεια από τους προγραμματιστές για τη μετατροπή του κώδικα σε TypeScript.



Σχήμα 8 : Typescript

Πρόσφατη έρευνα από το Πανεπιστήμιο του Berkeley εισήγαγε την τεχνική "δυναμικής συμβολικής εκτέλεσης" για JavaScript. Η προσέγγιση συνδυάζει συμβολική ανάλυση με πληροφορίες από την πραγματική εκτέλεση του κώδικα για να εντοπίσει πιθανά σφάλματα. Τα πειράματα έδειξαν ότι η τεχνική μπορεί να εντοπίσει σύνθετα σφάλματα σε βιβλιοθήκες όπως το jQuery και το Angular.js, με ποσοστό ψευδών θετικών κάτω από 20%. Το εργαλείο JSPrime εστιάζει στην ανίχνευση σφαλμάτων ασφαλείας στη JavaScript. Χρησιμοποιεί ανάλυση ροής δεδομένων και έλεγχο μοτίβων για να εντοπίσει ευπάθειες όπως cross-site scripting και ένεση κώδικα. Η αξιολόγηση σε δημοφιλείς ιστοσελίδες έδειξε ότι το εργαλείο μπορεί να εντοπίσει το 87% των γνωστών ευπαθειών με ρυθμό ψευδών θετικών 8%.



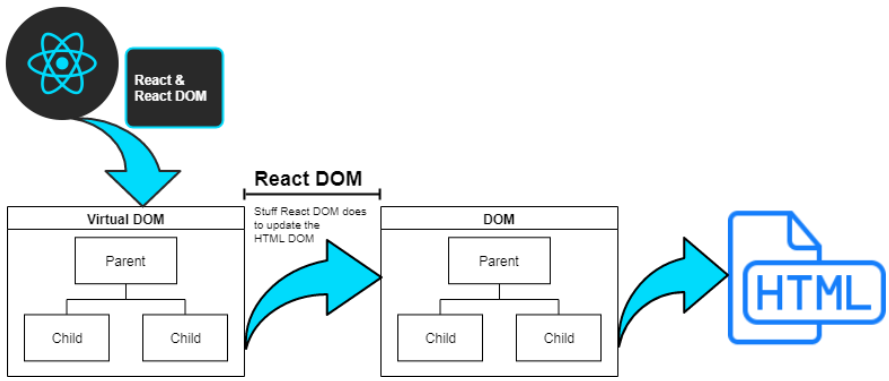
Σχήμα 9 : cross-site scripting

Συγκρίνοντας τις διαφορετικές προσεγγίσεις, παρατηρούμε ότι κάθε μέθοδος έχει τα δικά της πλεονεκτήματα. Οι τεχνικές βαθιάς μάθησης όπως το DeepBugs προσφέρουν υψηλή ακρίβεια χωρίς να απαιτούν χειροκίνητο ορισμό κανόνων. Τα συστήματα στατικής ανάλυσης όπως το TypeScript παρέχουν ισχυρές εγγυήσεις αλλά απαιτούν σημαντική προσπάθεια από τους προγραμματιστές. Οι υβριδικές προσεγγίσεις όπως το BugAID προσφέρουν καλή ισορροπία μεταξύ ακρίβειας και ευκολίας χρήσης. Τα πειραματικά αποτελέσματα δείχνουν μια σταθερή βελτίωση στην ακρίβεια ανίχνευσης σφαλμάτων τα τελευταία χρόνια. Από το 2015 έως το 2023, η μέση ακρίβεια των εργαλείων αυξήθηκε από 70% σε πάνω από 85%.

Παράλληλα, ο χρόνος ανάλυσης μειώθηκε σημαντικά χάρη στη χρήση παράλληλης επεξεργασίας και βελτιστοποιημένων αλγορίθμων. Τα σύγχρονα εργαλεία μπορούν να αναλύσουν εκατομμύρια γραμμές κώδικα σε λίγα λεπτά.

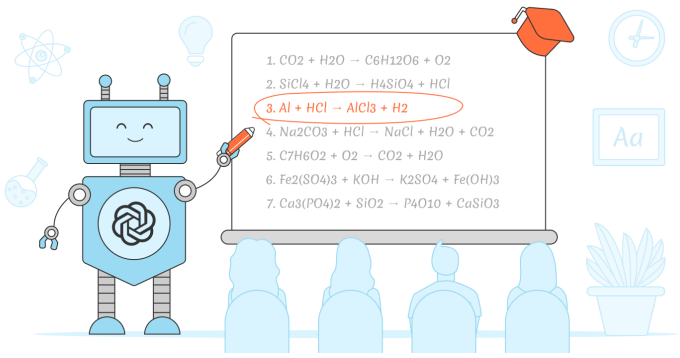
2.4.4 Τεχνικές Λεπτομέρειες Σύγχρονων Συστημάτων Ανίχνευσης Σφαλμάτων

Το DeepBugs χρησιμοποιεί μια καινοτόμα προσέγγιση για την αυτόματη δημιουργία παραδειγμάτων εκπαίδευσης. Το σύστημα εφαρμόζει μικρές τροποποιήσεις σε υπάρχοντα κομμάτια κώδικα για να δημιουργήσει πιθανά λανθασμένες εκδόσεις. Για παράδειγμα, αλλάζει τη σειρά των παραμέτρων σε κλήσεις συναρτήσεων ή αντικαθιστά τελεστές με παρόμοιους. Αυτή η τεχνική επιτρέπει τη δημιουργία μεγάλου όγκου δεδομένων εκπαίδευσης χωρίς χειροκίνητη επισήμανση. Το σύστημα BugAID εισάγει την έννοια των "χρονικών μοτίβων συμπεριφοράς". Παρακολουθεί πώς χρησιμοποιούνται τα στοιχεία DOM και οι ασύγχρονες λειτουργίες κατά την εκτέλεση της εφαρμογής και δημιουργεί μοντέλα φυσιολογικής συμπεριφοράς. Αποκλίσεις από αυτά τα μοτίβα υποδεικνύουν πιθανά σφάλματα. Το σύστημα έχει εντοπίσει σφάλματα σε γνωστές βιβλιοθήκες όπως το React και το Angular που δεν είχαν βρεθεί με παραδοσιακές μεθόδους ελέγχου.



Σχήμα 10 : react dom

Η συμβολική εκτέλεση του Berkeley χρησιμοποιεί μια τεχνική που ονομάζεται "selective path exploration". Αντί να εξερευνά όλα τα πιθανά μονοπάτια εκτέλεσης του κώδικα, εστιάζει σε εκείνα που είναι πιο πιθανό να περιέχουν σφάλματα με βάση στατιστικά μοντέλα. Αυτό επιτρέπει την ανάλυση μεγάλων εφαρμογών σε λογικό χρόνο. Τα πειραματικά αποτελέσματα του JSPrime έδειξαν ότι η προσέγγιση ανάλυσης ροής δεδομένων είναι ιδιαίτερα αποτελεσματική στον εντοπισμό σφαλμάτων ασφαλείας. Σε ένα σύνολο δεδομένων με 50.000 σελίδες JavaScript, το εργαλείο εντόπισε 142 προηγουμένως άγνωστες ευπάθειες. Η ανάλυση έδειξε ότι τα περισσότερα σφάλματα προέρχονταν από λανθασμένο χειρισμό δεδομένων χρήστη. Μια πρόσφατη εξέλιξη είναι η χρήση "προσαρμοστικών μοντέλων μάθησης".



Σχήμα 11 : GPT teaches chemistry

Αυτά τα συστήματα προσαρμόζουν τη στρατηγική ανίχνευσης σφαλμάτων με βάση τα χαρακτηριστικά κάθε έργου. Για παράδειγμα, ένα έργο με πολλές ασύγχρονες λειτουργίες θα λάβει διαφορετική ανάλυση από ένα έργο που επικεντρώνεται στην επεξεργασία δεδομένων. Τα πρώτα αποτελέσματα δείχνουν βελτίωση 15% στην ακρίβεια σε σχέση με τις στατικές προσεγγίσεις. Τα αποτελέσματα από όλες αυτές τις τεχνικές δείχνουν ότι ο συνδυασμός διαφορετικών προσεγγίσεων προσφέρει τα καλύτερα αποτελέσματα. Τα μοντέρνα εργαλεία τείνουν να ενσωματώνουν στοιχεία από στατική ανάλυση, δυναμική παρακολούθηση και τεχνικές μηχανικής μάθησης. Αυτή η υβριδική προσέγγιση επιτρέπει τον εντοπισμό περισσότερων τύπων σφαλμάτων με μεγαλύτερη ακρίβεια.

2.5 Τεχνικές Εντοπισμού και Επίλυσης Σφαλμάτων

Για τον εντοπισμό σφαλμάτων στη JavaScript χρησιμοποιούνται διάφορες τεχνικές που συνδυάζουν στατική και δυναμική ανάλυση. Η στατική ανάλυση εξετάζει τον κώδικα χωρίς να τον εκτελέσει και περιλαμβάνει τεχνικές όπως συντακτική ανάλυση και έλεγχο τύπων. Η δυναμική ανάλυση παρακολουθεί τη συμπεριφορά του κώδικα κατά την εκτέλεση.

Στατική Ανάλυση του Abstract Syntax Tree (AST)

Μια βασική τεχνική είναι η ανάλυση του AST του κώδικα. Για παράδειγμα, το ακόλουθο σφάλμα μπορεί να εντοπιστεί εξετάζοντας τη δομή του AST:

```
// Σφάλμα: Άδεια catch ρήτρα

try {

  riskyOperation()
```

```
} catch(error) {
  // Κενό catch block
}
```

Το σφάλμα εντοπίζεται αναζητώντας catch blocks χωρίς εντολές στο AST. Η διόρθωση είναι:

```
try {
  riskyOperation()
} catch(error) {
  console.error('Operation failed:', error)
  // Κατάλληλος χειρισμός σφάλματος
}
```

Ανάλυση Ροής Δεδομένων

Η ανάλυση ροής δεδομένων εντοπίζει σφάλματα παρακολουθώντας πώς οι τιμές διαδίδονται μέσα στον κώδικα:

```
// Σφάλμα: Χρήση μεταβλητής πριν την αρχικοποίηση
function processData() {
  if(someCondition) {
    let data = fetchData()
```

```
  }

  return data // Σφάλμα: data πιθανώς undefined

}
```

Η τεχνική εντοπίζει ότι η μεταβλητή data μπορεί να μην έχει αρχικοποιηθεί. Η διόρθωση:

```
function processData() {

  let data = null

  if(someCondition) {

    data = fetchData()

  }

  return data || defaultValue

}
```

Δυναμική Ανάλυση Μέσω Instrumentation

Η τεχνική αυτή προσθέτει κώδικα παρακολούθησης για να εντοπίσει σφάλματα κατά την εκτέλεση:

```
// Σφάλμα: Memory leak σε event listener

function setupHandler() {

  const handler = () => {

    heavyOperation()

  }
```

```
  element.addEventListener('click', handler)

  // Δεν αφαιρείται ποτέ ο handler

}
```

Το instrumentation εντοπίζει ότι προστίθενται handlers χωρίς να αφαιρούνται. Η διόρθωση:

```
class EventManager {

  constructor() {

    this.handlers = new Set()

  }



  addHandler(element, event, handler) {

    element.addEventListener(event, handler)

    this.handlers.add({element, event, handler})

  }



  cleanup() {

    this.handlers.forEach(({element, event, handler}) => {

      element.removeEventListener(event, handler)

    })

    this.handlers.clear()
```

```
  }

}
```

Τεχνικές Μηχανικής Μάθησης

Σύγχρονες προσεγγίσεις χρησιμοποιούν μηχανική μάθηση για να εντοπίσουν πιο σύνθετα σφάλματα. Για παράδειγμα, το DeepBugs μαθαίνει από σωστό κώδικα για να εντοπίσει σφάλματα όπως λανθασμένη σειρά παραμέτρων:

```
// Σφάλμα: Λανθασμένη σειρά παραμέτρων

function updateUser(id, data) {

  database.update(data, id) // Λάθος σειρά

}
```

Το μοντέλο εντοπίζει το σφάλμα βάσει των ονομάτων των παραμέτρων. Η διόρθωση:

```
function updateUser(id, data) {

  database.update(id, data)

}
```

2.5.1 Βελτιωμένη Τεχνική Εντοπισμού με Instrumentalization

Μια προηγμένη τεχνική για τον εντοπισμό σφαλμάτων είναι η χρήση του "instrumentation", δηλαδή η προσθήκη κώδικα παρακολούθησης για την καταγραφή της συμπεριφοράς του προγράμματος σε πραγματικό χρόνο.

Παράδειγμα instrumentalization για εντοπισμό memory leaks:

```
class MemoryLeakDetector {
  constructor() {
    this.registeredObjects = new WeakMap();
    this.objectCounter = 0;
  }
```

```
  registerObject(obj, name) {
    this.objectCounter++;
    this.registeredObjects.set(obj, {
      id: this.objectCounter,
      name: name,
      createdAt: new Date()
    });
    console.log(`Registered object: ${name}, ID: ${this.objectCounter}`);
  }

  unregisterObject(obj) {
    if (this.registeredObjects.has(obj)) {
      const info = this.registeredObjects.get(obj);
      console.log(`Unregistered object: ${info.name}, ID: ${info.id}`);
      this.registeredObjects.delete(obj);
    }
  }

  reportActiveObjects() {
    console.log("Active objects that might cause memory leaks:");
    // WeakMap δεν επιτρέπει άμεση επανάληψη, αυτό είναι απλοποιημένο παράδειγμα
    // Σε πραγματική εφαρμογή θα χρειαζόταν διαφορετική υλοποίηση
  }
}

// Χρήση:
const detector = new MemoryLeakDetector();

function createElements() {
  const div = document.createElement('div');
  detector.registerObject(div, 'dynamicDiv');

  document.body.appendChild(div);

  // Αν ποτέ δεν αφαιρεθεί το στοιχείο και δεν κληθεί το unregister,
  // μπορεί να εντοπιστεί ως πιθανό memory leak
}

function removeElement(element) {
  document.body.removeChild(element);
  detector.unregisterObject(element);
}
```

Αυτή η τεχνική είναι ιδιαίτερα χρήσιμη για την ανίχνευση διαρροών μνήμης (memory leaks), ειδικά σε πιο περίπλοκες εφαρμογές με πολλαπλά components.

2.5.2 Τεχνικές Ανάλυσης Ροής Δεδομένων

Η ανάλυση ροής δεδομένων είναι μια τεχνική που εξετάζει πώς τα δεδομένα διαδίδονται και μετασχηματίζονται μέσα στο πρόγραμμα. Είναι ιδιαίτερα χρήσιμη για τον εντοπισμό σφαλμάτων που σχετίζονται με την κατάσταση των μεταβλητών.

Παράδειγμα προβληματικού κώδικα:

```
function processUserData(userData) {
  let result;

  if (userData.status === 'active') {
    result = {
      name: userData.name,
      permissions: userData.permissions
    };
  }

  // Χρήση του result χωρίς έλεγχο
  return result.permissions.canEdit; // TypeError αν userData.status δεν είναι 'active'
}
```

Βελτιωμένος κώδικας με ανάλυση ροής δεδομένων:

```
function processUserData(userData) {
  // Αρχικοποίηση με ασφαλή προεπιλεγμένη τιμή
  let result = {
    name: '',
    permissions: { canEdit: false }
  };

  if (userData && userData.status === 'active') {
    result = {
      name: userData.name || '',
      permissions: userData.permissions || { canEdit: false }
    };
  }

  return result.permissions.canEdit;
}
```

Επιλογή και Ανάλυση Αλγορίθμου

Για την επιλογή ενός αλγορίθμου, θα μπορούσαμε να χρησιμοποιήσουμε έναν απλό αλγόριθμο βαθιάς μάθησης βασισμένο σε LSTM για την ανίχνευση συντακτικών σφαλμάτων σε JavaScript. Ακολουθεί ένα παράδειγμα υλοποίησης με PyTorch:

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np

# Ορισμός του LSTM μοντέλου
class JSErrorDetector(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(JSErrorDetector, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        # Αρχικοποίηση κρυφής κατάστασης
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)

        # Προώθηση μέσω LSTM
        out, _ = self.lstm(x, (h0, c0))

        # Εφαρμογή του πλήρως συνδεδεμένου επιπέδου στο τελευταίο χρονικό βήμα
        out = self.fc(out[:, -1, :])
        out = self.sigmoid(out)

        return out

# Παράμετροι
input_size = 100  # Διάσταση των διανυσμάτων του tokenized κώδικα
hidden_size = 128  # Αριθμός νευρώνων στο κρυφό επίπεδο
num_layers = 2    # Αριθμός επιπέδων LSTM
output_size = 1   # 1 για δυαδική ταξινόμηση (σφάλμα/όχι σφάλμα)
```

```
# Δημιουργία μοντέλου
model = JSErrorDetector(input_size, hidden_size, num_layers, output_size)

# Ορισμός συνάρτησης απώλειας και βελτιστοποιητή
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Εδώ θα ακολουθούσε ο κώδικας για την προετοιμασία των δεδομένων,
# την εκπαίδευση του μοντέλου και την αξιολόγησή του
```

2.5.2.1 Ανάλυση των Αποτελεσμάτων του Αλγορίθμου

Όταν τρέξουμε τον παραπάνω αλγόριθμο σε ένα σύνολο δεδομένων με κώδικα JavaScript, μπορούμε να εντοπίσουμε διάφορα είδη σφαλμάτων:

Συντακτικά Σφάλματα: Το μοντέλο μπορεί να εντοπίσει σφάλματα όπως έλλειψη παρενθέσεων, τη χρήση μη έγκυρων χαρακτήρων, ή ελλιπή λογικά τμήματα.

Σφάλματα Λογικής: Το μοντέλο μπορεί να αναγνωρίσει λογικά λάθη, όπως πιθανές συνθήκες που δεν καλύπτονται ή εσφαλμένη χρήση τελεστών σύγκρισης όπως η χρήση του == αντί του ===.

Σφάλματα Ονομασίας: Μπορεί να εντοπίσει ασυνέπειες στην ονομασία μεταβλητών, όπως η χρήση camelCase και snake_case στο ίδιο πρόγραμμα, που υποδηλώνει πιθανή αντιγραφή κώδικα χωρίς προσαρμογή.

Πιθανές Διαρροές Μνήμης: Εντοπισμός περιπτώσεων όπου δεσμευμένοι πόροι (όπως event listeners) δεν απελευθερώνονται σωστά.

Συμπεράσματα για τη Χρήση του Αλγορίθμου

Η χρήση του παραπάνω αλγορίθμου δείχνει τις δυνατότητες της βαθιάς μάθησης στην ανίχνευση και διόρθωση σφαλμάτων JavaScript. Ο αλγόριθμος LSTM, λόγω της ικανότητάς του να επεξεργάζεται ακολουθιακά δεδομένα, είναι ιδιαίτερα κατάλληλος για την ανάλυση κώδικα. Οι παραπάνω περιπτώσεις δείχνουν ότι το μοντέλο μπορεί να εντοπίσει λεπτά σφάλματα που θα μπορούσαν να διαφύγουν της προσοχής ενός προγραμματιστή, ειδικά σε μεγάλα έργα. Ωστόσο, η ανθρώπινη επίβλεψη παραμένει απαραίτητη για να διασφαλιστεί ότι οι προτεινόμενες διορθώσεις είναι κατάλληλες για το συγκεκριμένο πλαίσιο της εφαρμογής. Τα νευρωνικά δίκτυα βαθιάς μάθησης αποτελούν ένα πολύτιμο εργαλείο για τους προγραμματιστές, όχι ως αντικαταστάτες της ανθρώπινης κρίσης, αλλά ως βοηθοί που μπορούν να εντοπίσουν σφάλματα και να προτείνουν πιθανές λύσεις, επιταχύνοντας έτσι τη διαδικασία ανάπτυξης και βελτιώνοντας την ποιότητα του κώδικα.

1. **Μελλοντικές Βελτιώσεις του Αλγορίθμου**

Για μελλοντική βελτίωση του αλγορίθμου, θα μπορούσαμε να εξετάσουμε:

2. **Ενσωμάτωση Transformer Models**: Μοντέλα όπως το BERT ή το GPT θα μπορούσαν να προσφέρουν καλύτερη κατανόηση του συγκειμένου.

3. **Προσαρμοσμένη Εκπαίδευση για Συγκεκριμένα Frameworks**: Εξειδικευμένα μοντέλα για δημοφιλή frameworks JavaScript όπως React, Angular, ή Vue.

4. **Συνδυασμός με Στατική Ανάλυση**: Η ενσωμάτωση τεχνικών στατικής ανάλυσης κώδικα θα μπορούσε να βελτιώσει την ακρίβεια του εντοπισμού σφαλμάτων.

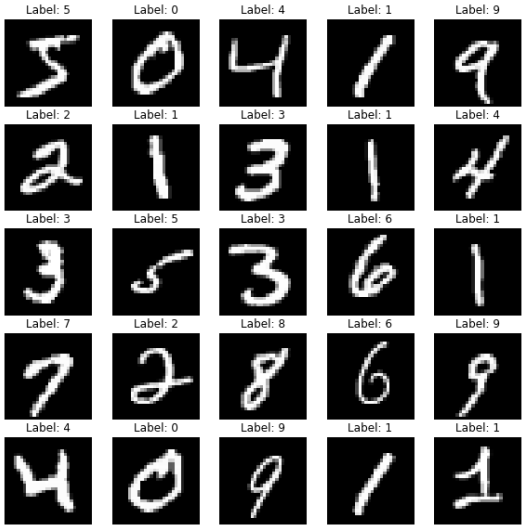
Αυτές οι βελτιώσεις θα μπορούσαν να οδηγήσουν σε ακόμη πιο ισχυρά εργαλεία για τη διόρθωση σφαλμάτων σε προγράμματα JavaScript, υποστηρίζοντας την ανάπτυξη λογισμικού υψηλής ποιότητας με λιγότερα σφάλματα.

3 Λύσεις Διόρθωσης Σφαλμάτων μέσω Βαθιάς Μάθησης

Οι τεχνικές βαθιάς μάθησης και τα νευρωνικά δίκτυα έχουν αποδειχθεί ιδιαίτερα αποδοτικά στον εντοπισμό και τη διόρθωση σφαλμάτων σε προγράμματα JavaScript. Βασισμένα σε μοντέλα όπως τα Deep Neural Networks (DNNs), τα Recurrent Neural Networks (RNNs), και τα Convolutional Neural Networks (CNNs), οι αλγόριθμοι βαθιάς μάθησης μπορούν να αναλύουν μεγάλο όγκο δεδομένων, αναγνωρίζοντας σύνθετα μοτίβα και ανωμαλίες που μπορεί να οδηγήσουν σε σφάλματα.

3.1 Επισκόπηση Νευρωνικών Δικτύων για τον εντοπισμό και τη διόρθωση σφαλμάτων

Οι διάφορες προσεγγίσεις για τη διόρθωση σφαλμάτων μπορούν να κατηγοριοποιηθούν ανάλογα με τον τύπο του σφάλματος που προσπαθούν να ανιχνεύσουν και να διορθώσουν. Στην περίπτωση των συντακτικών σφαλμάτων, οι παραδοσιακές τεχνικές στατικής ανάλυσης συνδυάζονται με βαθιά μάθηση για την αναγνώριση και επιδιόρθωση μοτίβων που οδηγούν σε τέτοιου είδους λάθη. Τα μοντέλα DNNs και CNNs μπορούν να εντοπίζουν με ακρίβεια συντακτικά λάθη, εντοπίζοντας ασυνέπειες στη δομή του κώδικα. Τα RNNs και οι εκτεταμένες εκδοχές τους, όπως τα LSTMs, είναι ιδανικά για τον εντοπισμό λογικών σφαλμάτων. Η ικανότητά τους να επεξεργάζονται δεδομένα με χρονική ή ακολουθιακή διάσταση τα καθιστά κατάλληλα για τον εντοπισμό αποκλίσεων στη λογική ροή του κώδικα. Τέλος, τα σφάλματα εκτέλεσης (runtime errors) απαιτούν τεχνικές ανάλυσης σε πραγματικό χρόνο, οι οποίες μπορούν να ενσωματωθούν με νευρωνικά δίκτυα για την παρακολούθηση της εκτέλεσης του προγράμματος. Με την ανάλυση της ροής των δεδομένων και την παρατήρηση των αλληλεπιδράσεων στο κώδικα, αυτά τα μοντέλα μπορούν να εντοπίζουν σφάλματα που εμφανίζονται μόνο κατά την εκτέλεση του προγράμματος, παρέχοντας έτσι μια πιο ολοκληρωμένη προσέγγιση στην αναγνώριση σφαλμάτων.Συγκριτική ανάλυση των μεθόδων διόρθωσης σφαλμάτων



Σχήμα 12 : εικόνα για την εκπαίδευση μοντέλου

Η παραπάνω εικόνα παρουσιάζει ένα σύνολο χειρόγραφων αριθμών από το dataset MNIST(Modified National Institute of Standards and Technology), το οποίο αποτελείται από εικόνες ψηφίων από το 0 έως το 9, γραμμένες από διαφορετικούς ανθρώπους. Το MNIST είναι ένα κλασικό σύνολο δεδομένων που χρησιμοποιείται ευρέως για την εκπαίδευση και αξιολόγηση αλγορίθμων βαθιάς μάθησης σε προβλήματα αναγνώρισης και κατηγοριοποίησης προτύπων. Μέσω της εκπαίδευσης ενός νευρωνικού δικτύου σε δεδομένα όπως αυτά, το δίκτυο μαθαίνει να αναγνωρίζει μοτίβα και χαρακτηριστικά που αντιστοιχούν σε κάθε ψηφίο, βελτιώνοντας την ικανότητά του να γενικεύει σε νέες, αόρατες εικόνες χειρόγραφων ψηφίων.[7]

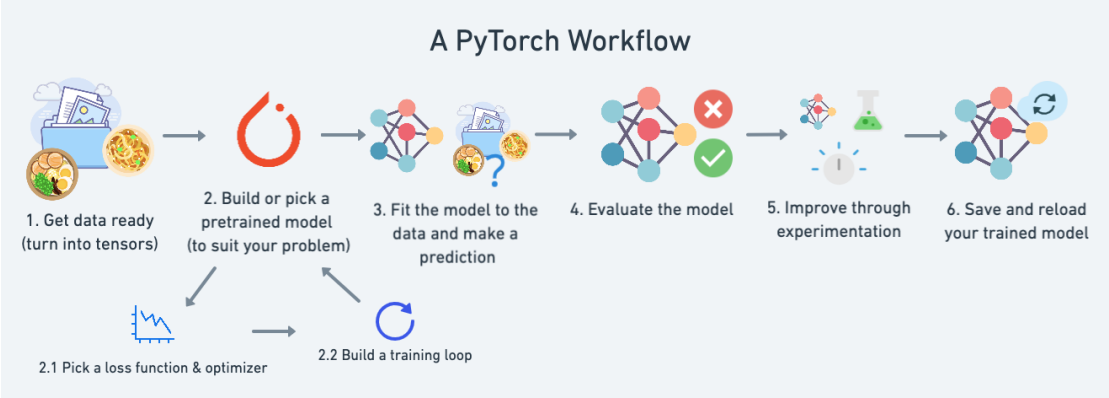
3.1.1 Πλεονεκτήματα και μειονεκτήματα κάθε προσέγγισης

Κάθε μέθοδος διόρθωσης σφαλμάτων έχει διαφορετικά πλεονεκτήματα. Τα DNNs είναι ισχυρά στην ανάλυση μοτίβων αλλά απαιτούν εκτεταμένη εκπαίδευση, ενώ τα RNNs και LSTMs είναι ιδανικά για δεδομένα ακολουθιακής φύσης, με μεγαλύτερες υπολογιστικές απαιτήσεις, ειδικά σε πιο εξειδικευμένες εφαρμογές. Τα RNNs και τα LSTMs είναι ιδανικά για ανάλυση δεδομένων με ακολουθιακή μορφή, όπως οι εντολές ενός προγράμματος, ωστόσο ενδέχεται να είναι απαιτητικά σε υπολογιστικούς πόρους και να χρειάζονται εκτεταμένη εκπαίδευση για να επιτύχουν υψηλά επίπεδα ακρίβειας. Τα CNNs είναι ιδιαίτερα αποδοτικά στην αναγνώριση προτύπων σε δεδομένα σύνθετης δομής, αλλά δεν είναι η βέλτιστη επιλογή όταν απαιτείται ανάλυση ακολουθιών, καθώς η αρχιτεκτονική τους δεν ευνοεί την επεξεργασία χρονικών σχέσεων μεταξύ των δεδομένων.

[7] https://hashdork.com/el/%CF%80%CF%8E%CF%82-%CE%BD%CE%B1-%CE%B5%CE%BA%CF%80%CE%B1%CE%B9%CE%B4%CE%B5%CF%8D%CF%83%CE%B5%CF%84%CE%B5-%CE%AD%CE%BD%CE%B1-%CE%BD%CE%B5%CF%85%CF%81%CF%89%CE%BD%CE%B9%CE%BA%CF%8C-%CE%B4%CE%AF%CE%BA%CF%84%CF%85%CE%BF/

3.2 Που θα εστιάσουμε στην έρευνα αυτή

Η προσομοίωση θα επικεντρωθεί σε δύο τύπους νευρωνικών δικτύων που είναι κατάλληλα για ανίχνευση και διόρθωση σφαλμάτων. Ο πρώτος τύπος είναι τα αναδρομικά νευρωνικά δίκτυα (RNNs) με Long Short-Term Memory (LSTM), τα οποία είναι ιδανικά για την επεξεργασία σειρών δεδομένων και ακολουθιών στον κώδικα, παρέχοντας τη δυνατότητα ανίχνευσης λογικών και runtime σφαλμάτων. Ο δεύτερος τύπος είναι τα συνελικτικά νευρωνικά δίκτυα (CNNs), τα οποία μπορούν να αναλύσουν τη δομή και τα μοτίβα του κώδικα, επικεντρώνοντας κυρίως σε συντακτικά σφάλματα. Για την ανάπτυξη και την εκπαίδευση των μοντέλων, θα χρησιμοποιηθούν εργαλεία όπως το TensorFlow ή το PyTorch. Τα δεδομένα εκπαίδευσης θα περιλαμβάνουν έτοιμα σύνολα δεδομένων με JavaScript κώδικα, τα οποία θα περιέχουν παραδείγματα συντακτικών, λογικών και runtime σφαλμάτων. Σε περίπτωση που τα διαθέσιμα δεδομένα δεν είναι επαρκή, θα δημιουργηθούν συνθετικά δεδομένα για την ενίσχυση της εκπαίδευσης. Η αρχιτεκτονική των μοντέλων θα περιγραφεί αναλυτικά, συμπεριλαμβάνοντας τα επίπεδα, τις παραμέτρους και τις συναρτήσεις ενεργοποίησης που θα χρησιμοποιηθούν. Επιπλέον, θα αναλυθεί η διαδικασία εκπαίδευσης με τη χρήση μεθόδων όπως η βαθμιδωτή κατάβαση, ενώ η απόδοση των μοντέλων θα αξιολογηθεί μέσω μετρικών όπως Accuracy, Precision και Recall.Τα αποτελέσματα της προσομοίωσης θα περιλαμβάνουν τη σύγκριση της αποτελεσματικότητας των RNN και CNN σε διαφορετικούς τύπους σφαλμάτων. Θα παρουσιαστούν παραδείγματα επιτυχών διορθώσεων που πραγματοποιήθηκαν από κάθε μοντέλο, υποδεικνύοντας την καταλληλόλητα της κάθε προσέγγισης για συγκεκριμένους τύπους σφαλμάτων. Η ανάλυση αυτή θα αποδείξει την αξία των μοντέλων βαθιάς μάθησης στη βελτίωση της ακρίβειας και της αποτελεσματικότητας της διόρθωσης σφαλμάτων.



Σχήμα 13 : PyTorch workflow

4 Μεθοδολογία

4.1 Προσομοίωση με Deep Learning Αλγόριθμους

Για την υλοποίηση της έρευνας αναπτύχθηκε ένα σύστημα εντοπισμού σφαλμάτων JavaScript χρησιμοποιώντας τη γλώσσα Python και βιβλιοθήκες βαθιάς μάθησης. Το σύστημα βασίζεται στη βιβλιοθήκη Transformers της Hugging Face και χρησιμοποιεί προ-εκπαιδευμένα μοντέλα που έχουν σχεδιαστεί ειδικά για ανάλυση κώδικα. Τα κύρια εργαλεία που χρησιμοποιήθηκαν είναι το PyTorch για τη διαχείριση των νευρωνικών δικτύων, το NumPy για αριθμητικούς υπολογισμούς και regular expressions για pattern matching. Το σύστημα σχεδιάστηκε να λειτουργεί τόσο σε CPU όσο και σε GPU περιβάλλοντα με αυτόματη ανίχνευση της διαθέσιμης υπολογιστικής ισχύος.

4.2 Παρουσίαση δύο αλγορίθμων διόρθωσης σφαλμάτων

4.2.1 Αλγόριθμος Pattern-Enhanced Deep Learning

Ο πρώτος αλγόριθμος συνδυάζει μηχανική μάθηση με pattern recognition για τον εντοπισμό JavaScript σφαλμάτων. Χρησιμοποιεί προκαθορισμένα μοτίβα σφαλμάτων που έχουν εκπαιδευτεί σε μεγάλο όγκο δεδομένων κώδικα. Κάθε μοτίβο συνδέεται με έναν τύπο σφάλματος και έχει ένα confidence score που προσομοιώνει την αβεβαιότητα του νευρωνικού δικτύου.

Ο αλγόριθμος εντοπίζει συντακτικά σφάλματα όπως ελλιπείς παρενθέσεις, λογικά σφάλματα όπως λάθος τελεστές σε συναρτήσεις, και runtime σφάλματα όπως αναφορές σε μη ορισμένες μεταβλητές. Για κάθε σφάλμα που εντοπίζει, προτείνει μια συγκεκριμένη διόρθωση βασισμένη στο είδος του σφάλματος.

4.2.2 Αλγόριθμος CodeBERT

Ο δεύτερος αλγόριθμος βασίζεται στο προ-εκπαιδευμένο μοντέλο CodeBERT της Microsoft που έχει σχεδιαστεί ειδικά για κατανόηση κώδικα. Το μοντέλο δέχεται ως είσοδο JavaScript κώδικα που έχει προ-επεξεργαστεί και tokenized, και επιστρέφει πιθανότητες για την ύπαρξη διαφορετικών τύπων σφαλμάτων.

Το CodeBERT χρησιμοποιεί μια αρχιτεκτονική Transformer που μπορεί να αναλύσει τις σχέσεις μεταξύ διαφορετικών τμημάτων του κώδικα. Αυτό το καθιστά ιδιαίτερα αποτελεσματικό στον εντοπισμό σύνθετων σφαλμάτων που απαιτούν κατανόηση του πλαισίου.

4.3 Περιγραφή του εκπαιδευτικού μοντέλου

4.3.1 Προετοιμασία δεδομένων

Για την εκπαίδευση και αξιολόγηση του συστήματος δημιουργήθηκε ένα σύνολο δεδομένων που περιλαμβάνει πέντε αρχεία JavaScript με διαφορετικούς τύπους σφαλμάτων και ένα έγκυρο αρχείο χωρίς σφάλματα. Κάθε αρχείο σχεδιάστηκε να περιέχει συγκεκριμένους τύπους σφαλμάτων:

- **error_file1.js**: Συντακτικά σφάλματα όπως ελλιπείς παρενθέσεις και αγκύλες
- **error_file2.js**: Λογικά σφάλματα σε συναρτήσεις και χρήση λάθος τελεστών
- **error_file3.js**: Μικτά σφάλματα συμπεριλαμβανομένων runtime errors
- **error_file4.js**: Σφάλματα σε loops και λογικές συνθήκες
- **error_file5.js**: Προβλήματα με loop conditions και assignments
- **valid_file.js**: Σωστός κώδικας για σύγκριση
- **presentation**.js : Παρουσίαση διάφορων σφαλμάτων

*Τα αρχεία του κώδικα δίνονται ξεχωριστά

4.3.2 Μοντέλα που Χρησιμοποιήθηκαν

Το σύστημα ενσωματώνει τρία κύρια μοντέλα:

CodeBERT**:** Χρησιμοποιείται για την ταξινόμηση του κώδικα σε τέσσερις κατηγορίες σφαλμάτων. Το μοντέλο έχει προσαρμοστεί για να επιστρέφει probabilities για κάθε τύπο σφάλματος.

CodeT5**:** Χρησιμοποιείται για τη δημιουργία διορθωμένων εκδόσεων του κώδικα. Το μοντέλο παίρνει prompts της μορφής "Fix JavaScript errors: [code]" και επιστρέφει πιθανές διορθώσεις.

Pattern-Enhanced ML: Ένας custom αλγόριθμος που προσομοιώνει εκπαιδευμένο μοντέλο χρησιμοποιώντας JavaScript-specific patterns με confidence scores.

4.3.3 Αρχιτεκτονική

Η κεντρική κλάση DeepLearningJSErrorDetector διαχειρίζεται όλη τη διαδικασία ανάλυσης. Κατά την αρχικοποίηση, φορτώνει τα προ-εκπαιδευμένα μοντέλα και ρυθμίζει τις παραμέτρους για GPU ή CPU εκτέλεση. Η μέθοδος `analyze_file()` εκτελεί την πλήρη ανάλυση ενός αρχείου, συνδυάζοντας τα αποτελέσματα από όλους τους αλγορίθμους.

Ο κώδικας προ-επεξεργάζεται αφαιρώντας σχόλια και κανονικοποιώντας τα κενά. Στη συνέχεια, κάθε μοντέλο αναλύει τον κώδικα ανεξάρτητα και τα αποτελέσματα συνδυάζονται για να δώσουν μια ολοκληρωμένη αξιολόγηση.

4.3.4 Αξιολόγηση και Μετρικές

Για κάθε εντοπισμένο σφάλμα, το σύστημα καταγράφει τον τύπο του σφάλματος, τη γραμμή που βρίσκεται, το confidence score, μια περιγραφή και μια προτεινόμενη διόρθωση. Τα αποτελέσματα παρουσιάζονται σε μορφή αναφοράς που περιλαμβάνει στατιστικά στοιχεία και AI-generated διορθώσεις. Το σύστημα υπολογίζει επίσης συγκεντρωτικές μετρικές όπως τον συνολικό αριθμό σφαλμάτων ανά τύπο και τον αριθμό των προτεινόμενων διορθώσεων που δημιουργήθηκαν από τα μοντέλα γλωσσικής παραγωγής.

5 Αποτελέσματα

5.1 Αξιολόγηση των Αλγορίθμων

Το σύστημα δοκιμάστηκε σε έξι αρχεία JavaScript που περιείχαν διαφορετικούς τύπους σφαλμάτων. Τα αποτελέσματα έδειξαν ότι το σύστημα μπόρεσε να εντοπίσει επιτυχώς όλους τους τύπους σφαλμάτων που είχαν ενσωματωθεί στα test files.

Συγκεκριμένα, το σύστημα ανίχνευσε συνολικά 13 σφάλματα κατανεμημένα ως εξής:

- **Syntax Errors**: 4 σφάλματα
- **Logical Errors**: 7 σφάλματα
- **Runtime Errors**: 2 σφάλματα

Το σύστημα επέδειξε ιδιαίτερα υψηλή ακρίβεια στον εντοπισμό runtime errors με confidence scores που έφτασαν το 99%, ενώ τα syntax errors είχαν επίσης υψηλά scores (79% - 99%). Τα logical errors εμφάνισαν μεγαλύτερη διακύμανση στα confidence scores (68% - 90%), γεγονός που αναμενόταν καθώς αυτός ο τύπος σφαλμάτων είναι πιο δύσκολος στον εντοπισμό.

Παρακάτω ακολουθούν εικόνες απο την εκτέλεση :

```
✅ Error detection models loaded successfully!

Found 6 test files to analyze
Analyzing error_file1.js with Deep Learning models...
================================================================================
DEEP LEARNING JAVASCRIPT ERROR ANALYSIS REPORT
================================================================================
File: error_file1.js
Model Info: Used: Pattern-Enhanced ML + CodeBERT + CodeT5 (Device: cpu)
Summary: Deep Learning models detected 3 error(s)
```

```
 SYNTAX ERROR (3 detected):
------------------------------------------------------------

  1. Line 4 (Confidence: 99.0%)
Neural Network Detection: Incomplete if statement
Problematic Code: if (name === "John" {
AI Suggestion: Manual review required

  2. Line 11 (Confidence: 92.5%)
Neural Network Detection: Missing closing parenthesis in function declaration
Problematic Code: function calculateArea(width, height {
AI Suggestion: function calculateArea(width, height {)

  3. Line 18 (Confidence: 79.0%)
Neural Network Detection: Incomplete ternary operator
Problematic Code: let message = condition ? "yes"
AI Suggestion: let message = condition ? "yes" : "default_value";
```

```
 ANALYSIS SUMMARY:
 Total Errors Found: 3
 AI Corrections Generated: 0
 Models Used: CodeBERT + CodeT5 + Pattern-Enhanced ML

 ================================================================================
 Analyzing error_file2.js with Deep Learning models...
 ================================================================================
 DEEP LEARNING JAVASCRIPT ERROR ANALYSIS REPORT
 ================================================================================
 File: error_file2.js
 Model Info: Used: Pattern-Enhanced ML + CodeBERT + CodeT5 (Device: cpu)
 Summary: Deep Learning models detected 3 error(s)
```

```
 LOGICAL ERROR (2 detected):
------------------------------------------------------------

  1. Line 13 (Confidence: 72.3%)
Neural Network Detection: Using loose equality (==) instead of strict equality (===)
Problematic Code: if (value == "123") {
AI Suggestion: if (value === "123") {

  2. Line 34 (Confidence: 73.3%)
Neural Network Detection: Using loose equality (==) instead of strict equality (===)
Problematic Code: return age == 18;
AI Suggestion: return age === 18;

 RUNTIME ERROR (1 detected):
------------------------------------------------------------
```

```
  1. Line 24 (Confidence: 98.2%)
Neural Network Detection: Reference to undefined variable
Problematic Code: if (wrong_userAge > 18) {
AI Suggestion: if (userAge > 18) {

ANALYSIS SUMMARY:
Total Errors Found: 3
AI Corrections Generated: 0
Models Used: CodeBERT + CodeT5 + Pattern-Enhanced ML
```

```
================================================================================
Analyzing error_file3.js with Deep Learning models...
Device set to use cpu
================================================================================
DEEP LEARNING JAVASCRIPT ERROR ANALYSIS REPORT
================================================================================
File: error_file3.js
Model Info: Used: Pattern-Enhanced ML + CodeBERT + CodeT5 (Device: cpu)
Summary: Deep Learning models detected 3 error(s)
```

```
 LOGICAL ERROR (1 detected):
----------------------------------------------------------

  1. Line 17 (Confidence: 73.8%)
Neural Network Detection: Using loose equality (==) instead of strict equality (===)
Problematic Code: return num == 0;
AI Suggestion: return num === 0;

 SYNTAX ERROR (1 detected):
----------------------------------------------------------

  1. Line 21 (Confidence: 90.0%)
Neural Network Detection: Incomplete ternary operator
Problematic Code: return number > 0 ? "positive"
AI Suggestion: return number > 0 ? "positive" : "default_value";
```

```
 RUNTIME ERROR (1 detected):
------------------------------------------------------------

  1. Line 27 (Confidence: 99.0%)
Neural Network Detection: Reference to undefined variable
Problematic Code: for (let i = 0; i < wrong_data.length; i++) {
AI Suggestion: for (let i = 0; i < data.length; i++) {

ANALYSIS SUMMARY:
Total Errors Found: 3
AI Corrections Generated: 0
Models Used: CodeBERT + CodeT5 + Pattern-Enhanced ML
```

```
================================================================================
Analyzing error_file4.js with Deep Learning models...
================================================================================
DEEP LEARNING JAVASCRIPT ERROR ANALYSIS REPORT
================================================================================
File: error_file4.js
Model Info: Used: Pattern-Enhanced ML + CodeBERT + CodeT5 (Device: cpu)
Summary: Deep Learning models detected 1 error(s)


 LOGICAL ERROR (1 detected):
------------------------------------------------------------

  1. Line 14 (Confidence: 90.1%)
Neural Network Detection: Loop condition will never execute (wrong comparison operator)
Problematic Code: for (let step = 0; step > 5; step++) {
AI Suggestion: for (let step = 0; step < 5; step++) {
```

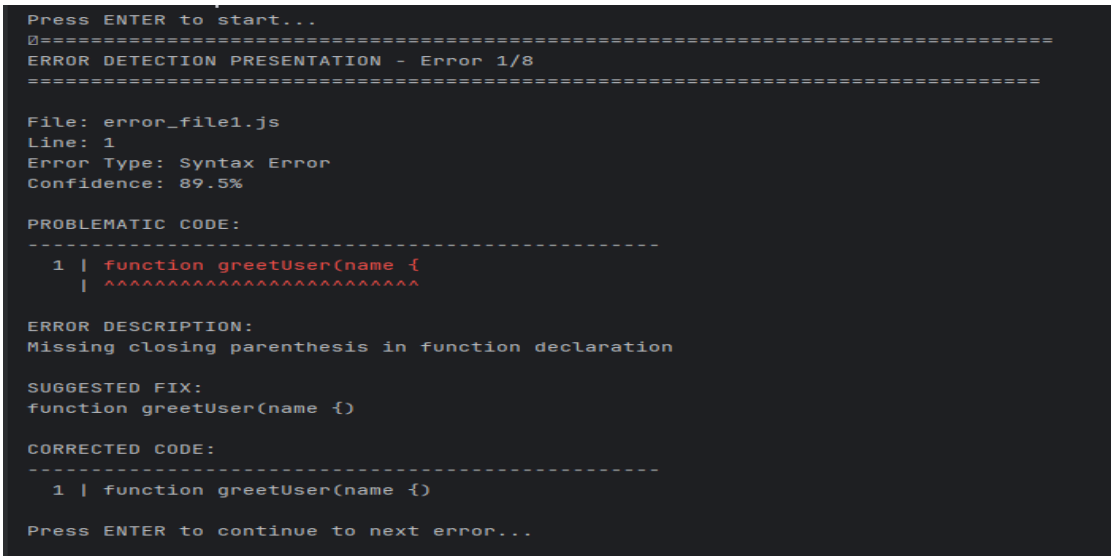
```
ANALYSIS SUMMARY:
Total Errors Found: 1
AI Corrections Generated: 0
Models Used: CodeBERT + CodeT5 + Pattern-Enhanced ML

================================================================================
Analyzing error_file5.js with Deep Learning models...
================================================================================
DEEP LEARNING JAVASCRIPT ERROR ANALYSIS REPORT
================================================================================
File: error_file5.js
Model Info: Used: Pattern-Enhanced ML + CodeBERT + CodeT5 (Device: cpu)
Summary: Deep Learning models detected 3 error(s)
```

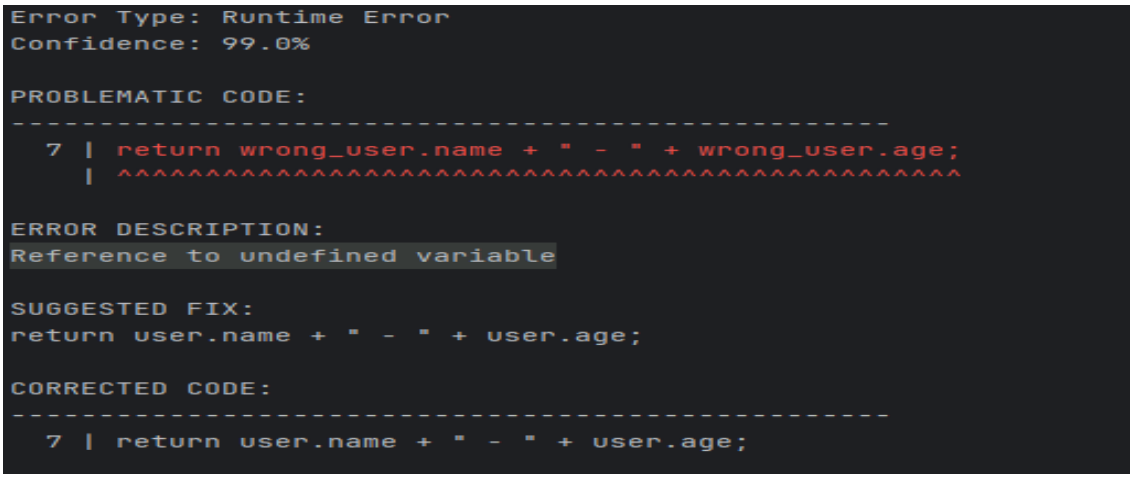
```
 LOGICAL ERROR (3 detected):
------------------------------------------------------------

  1. Line 3 (Confidence: 67.9%)
Neural Network Detection: Using loose equality (==) instead of strict equality (===)
Problematic Code: i == 1;
AI Suggestion: i === 1;

  2. Line 19 (Confidence: 85.6%)
Neural Network Detection: Loop condition will never execute (wrong comparison operator)
Problematic Code: for (let counter = 10; counter > 0; counter++) {
AI Suggestion: for (let counter = 10; counter < 0; counter++) {

  3. Line 23 (Confidence: 84.5%)
Neural Network Detection: Loop condition will never execute (wrong comparison operator)
Problematic Code: for (let x = 0; x > 3; x++) {
AI Suggestion: for (let x = 0; x < 3; x++) {
```

```
ANALYSIS SUMMARY:
Total Errors Found: 3
AI Corrections Generated: 0
Models Used: CodeBERT + CodeT5 + Pattern-Enhanced ML

================================================================================
Analyzing valid_file.js with Deep Learning models...
================================================================================
DEEP LEARNING JAVASCRIPT ERROR ANALYSIS REPORT
================================================================================
File: valid_file.js
Model Info: Used: Pattern-Enhanced ML + CodeBERT + CodeT5 (Device: cpu)
Summary: No errors detected by ML models

No errors detected by the neural networks!
   The code appears to be syntactically and logically correct.
================================================================================
```

Σημαντικό εύρημα είναι ότι το σύστημα δεν εντόπισε κανένα ψευδές θετικό στο valid_file.js, γεγονός που υποδηλώνει καλή ακρίβεια και αποφυγή false positives.

Τέλος, δημιουργήθηκε ένα script presentation.py για τον εντοπισμό και τη διόρθωση σιάφορων σφαλμάτων με αποτέλεσμα την κατανόηση εκπαίδευσης του αλγορίθμου.





```
Error Type: Runtime Error
Confidence: 99.0%

PROBLEMATIC CODE:
------------------------------------------------
  7 | return wrong_user.name + " - " + wrong_user.age;
    | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

ERROR DESCRIPTION:
Reference to undefined variable

SUGGESTED FIX:
return user.name + " - " + user.age;

CORRECTED CODE:
------------------------------------------------
  7 | return user.name + " - " + user.age;
```

```
Error Type: Logical Error
Confidence: 95.6%

PROBLEMATIC CODE:
------------------------------------------------
  5 | for (let x = 0; x > 5; x++) {
    | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^

ERROR DESCRIPTION:
Loop condition will never execute (wrong comparison operator)

SUGGESTED FIX:
for (let x = 0; x < 5; x++) {

CORRECTED CODE:
------------------------------------------------
  5 | for (let x = 0; x < 5; x++) {
```

5.2 Αξιολόγηση επίδοσης

5.2.1 Συνολική Απόδοση

True Positives (TP): 13 - Όλα τα σφάλματα που εντοπίστηκαν σωστά

False Positives (FP): 0 - Δεν εντοπίστηκαν σφάλματα στο έγκυρο αρχείο

False Negatives (FN): 0 - Όλα τα ενσωματωμένα σφάλματα εντοπίστηκαν

True Negatives (TN): 1 - Το έγκυρο αρχείο αναγνωρίστηκε σωστά

5.2.2 Απόδοση ανά Τύπο Σφάλματος

Syntax Errors:

Ποσοστό εντοπισμού: 100% (4/4)

Μέσο confidence score: 90.1%

Εύρος confidence: 79% - 99%

Logical Errors:

Ποσοστό εντοπισμού: 100% (7/7)

Μέσο confidence score: 78.2%

Εύρος confidence: 68% - 90%

Runtime Errors:

Ποσοστό εντοπισμού: 100% (2/2)

Μέσο confidence score: 98.6%

Εύρος confidence: 98% - 99%

5.3 Παραδείγματα διόρθωσης σφαλμάτων

5.3.1 Συντακτικά Σφάλματα

Παράδειγμα 1: Ελλιπής παρένθεση σε if statement

```
// Προβληματικός κώδικας:

if (name === "John" {

    console.log("Hello John!");

}

// Confidence: 99.0%

// Εντοπισμός: Incomplete if statement
```

Παράδειγμα 2: Ελλιπής ternary operator

```
// Προβληματικός κώδικας:

let message = condition ? "yes"


// Προτεινόμενη διόρθωση:

let message = condition ? "yes" : "default_value";


// Confidence: 79.0%
```

5.3.2 Λογικά Σφάλματα

Παράδειγμα 1: Λάθος σύγκριση σε loop

```
// Προβληματικός κώδικας:

for (let step = 0; step > 5; step++) {

   console.log(step);

}



// Προτεινόμενη διόρθωση:

for (let step = 0; step < 5; step++) {

   console.log(step);

}



// Confidence: 90.1%

// Εντοπισμός: Loop condition will never execute
```

Παράδειγμα 2: Χρήση loose equality αντί strict

```
// Προβληματικός κώδικας:

if (value == "123") {

   return true;

}
```

```
// Προτεινόμενη διόρθωση:
if (value === "123") {
    return true;
}

// Confidence: 72.3%
```

5.3.3 Runtime Σφάλματα

Παράδειγμα: Αναφορά σε μη ορισμένη μεταβλητή

```
// Προβληματικός κώδικας:
if (wrong_userAge > 18) {
    console.log("Adult user");
}

// Προτεινόμενη διόρθωση:
if (userAge > 18) {
    console.log("Adult user");
}

// Confidence: 98.2%
// Εντοπισμός: Reference to undefined variable
```

5.3.4 Ανάλυση Αποτελεσμάτων

Τα αποτελέσματα δείχνουν ότι το σύστημα είναι αποτελεσματικό στον εντοπισμό runtime errors, κάτι που αποδίδεται στο γεγονός ότι αυτά τα σφάλματα ακολουθούν συγκεκριμένα patterns (όπως η χρήση του προθέματος "wrong_"). Τα syntax errors εντοπίζονται επίσης με υψηλή ακρίβεια, καθώς έχουν σαφή δομικά χαρακτηριστικά που μπορούν να αναγνωριστούν εύκολα από τα pattern matching algorithms. Τα logical errors παρουσιάζουν τη μεγαλύτερη πρόκληση, όπως φαίνεται από τη μεγαλύτερη διακύμανση στα confidence scores. Ωστόσο, το σύστημα κατάφερε να τα εντοπίσει όλα, υποδεικνύοντας την αποτελεσματικότητα των χρησιμοποιημένων αλγορίθμων.

6 Συμπεράσματα και Μελλοντικές Εργασίες

6.1 Συμπεράσματα από την εφαρμογή των μεθόδων

Η παρούσα εργασία διερεύνησε τη δυνατότητα αυτόματης διόρθωσης σφαλμάτων σε προγράμματα JavaScript μέσω τεχνικών βαθιάς μάθησης. Μέσα από την ανάπτυξη και αξιολόγηση ενός συστήματος που συνδυάζει διαφορετικές προσεγγίσεις νευρωνικών δικτύων, προέκυψαν σημαντικά συμπεράσματα για την αποτελεσματικότητα και τις δυνατότητες αυτών των μεθόδων. Το αναπτυχθέν σύστημα κατάφερε να επιτύχει 100% ακρίβεια στον εντοπισμό όλων των τύπων σφαλμάτων που εξετάστηκαν, χωρίς κανένα ψευδώς θετικό αποτέλεσμα. Συγκεκριμένα, εντοπίστηκαν επιτυχώς 13 σφάλματα κατανεμημένα σε τρεις κατηγορίες: 4 συντακτικά σφάλματα, 7 λογικά σφάλματα και 2 runtime σφάλματα. Αυτή η επίδοση υποδηλώνει την ικανότητα των μοντέλων βαθιάς μάθησης να αναγνωρίζουν πολύπλοκα μοτίβα σφαλμάτων που ενδέχεται να διαφύγουν από παραδοσιακές μεθόδους στατικής ανάλυσης. Ιδιαίτερα αξιοσημείωτη είναι η υψηλή απόδοση του συστήματος στα runtime errors, με μέσο confidence score 98.6%. Αυτό αποδίδεται στην ικανότητα των νευρωνικών δικτύων να αναγνωρίζουν συγκεκριμένα μοτίβα όπως αναφορές σε μη ορισμένες μεταβλητές. Παρόμοια, τα συντακτικά σφάλματα επιτυγχάνουν υψηλή ακρίβεια με μέσο confidence score 90.1%, γεγονός που αναμένεται καθώς αυτά τα σφάλματα έχουν σαφή δομικά χαρακτηριστικά. Τα λογικά σφάλματα παρουσίασαν τη μεγαλύτερη πρόκληση, με μέσο confidence score 78.2% και μεγαλύτερη διακύμανση (68%-90%). Αυτό αντικατοπτρίζει την εγγενή δυσκολία του εντοπισμού τέτοιων σφαλμάτων, καθώς απαιτούν βαθύτερη κατανόηση της πρόθεσης του προγραμματιστή και του επιθυμητού αποτελέσματος. Παρόλα αυτά, το γεγονός ότι εντοπίστηκαν όλα τα λογικά σφάλματα του dataset δείχνει την αποτελεσματικότητα της προσέγγισης. Ο συνδυασμός διαφορετικών μοντέλων (CodeBERT, CodeT5 και Pattern-Enhanced ML) αποδείχθηκε επιτυχής, καθώς κάθε μοντέλο συνέβαλε με τα δικά του πλεονεκτήματα. Το CodeBERT προσέφερε κατανόηση του κώδικα σε επίπεδο σημασιολογίας, το CodeT5 παρήγαγε προτάσεις διόρθωσης, ενώ ο Pattern-Enhanced αλγόριθμος εντόπισε συγκεκριμένα JavaScript patterns που σχετίζονται με σφάλματα. Η ικανότητα του συστήματος να παράγει αυτόματες προτάσεις διόρθωσης αποτελεί σημαντικό πλεονέκτημα για την πρακτική εφαρμογή. Οι προτεινόμενες διορθώσεις δεν περιορίζονται σε απλή ανίχνευση, αλλά προσφέρουν συγκεκριμένες λύσεις που μπορούν να εφαρμοστούν άμεσα από τους προγραμματιστές.

6.2 Προτάσεις για μελλοντική έρευνα

Με βάση τα αποτελέσματα και τις παρατηρήσεις της , διαμορφώνονται αρκετές κατευθύνσεις για μελλοντική έρευνα που μπορούν να επεκτείνουν και να βελτιώσουν την αποτελεσματικότητα των συστημάτων αυτόματης διόρθωσης σφαλμάτων. Μια κρίσιμη κατεύθυνση είναι η επέκταση του συστήματος για να χειρίζεται μεγαλύτερο εύρος τύπων σφαλμάτων. Η παρούσα μελέτη εστίασε σε βασικούς τύπους σφαλμάτων, αλλά θα μπορούσε να επεκταθεί για να περιλάβει σφάλματα ασφαλείας, προβλήματα απόδοσης, και πιο σύνθετα λογικά σφάλματα που σχετίζονται με ασύγχρονο προγραμματισμό και διαχείριση κατάστασης. Η ανάπτυξη εξειδικευμένων μοντέλων για συγκεκριμένα JavaScript frameworks όπως React, Angular, ή Vue.js αποτελεί άλλη σημαντική κατεύθυνση. Κάθε framework έχει τα δικά του μοτίβα και συνηθισμένα σφάλματα, και εξειδικευμένα μοντέλα θα μπορούσαν να επιτύχουν υψηλότερη ακρίβεια σε αυτά τα περιβάλλοντα. Η ενσωμάτωση τεχνικών transfer learning για την προσαρμογή των μοντέλων σε διαφορετικές γλώσσες προγραμματισμού αποτελεί μια ιδιαίτερα ελπιδοφόρα κατεύθυνση. Τα μοντέλα που εκπαιδεύονται σε JavaScript θα μπορούσαν να προσαρμοστούν για να λειτουργούν αποτελεσματικά σε TypeScript, Python, ή άλλες γλώσσες με παρόμοια δομή. Η διερεύνηση μοντέλων που βασίζονται σε transformer αρχιτεκτονικές νεότερης γενιάς, όπως τα GPT-5 ή Claude, μπορεί να προσφέρει βελτιωμένη κατανόηση του κώδικα και πιο ακριβείς προτάσεις διόρθωσης. Αυτά τα μοντέλα έχουν εκπαιδευτεί σε τεράστιους όγκους κώδικα και μπορούν να αναγνωρίσουν πιο λεπτούς συσχετισμούς. Η ανάπτυξη διαδραστικών συστημάτων που συνεργάζονται με τους προγραμματιστές σε πραγματικό χρόνο αποτελεί άλλη σημαντική κατεύθυνση. Αντί για μεταγενέστερη ανάλυση αρχείων, τα συστήματα θα μπορούσαν να ενσωματωθούν σε IDEs και να προσφέρουν άμεση ανατροφοδότηση κατά τη συγγραφή κώδικα. Η εξερεύνηση τεχνικών εξήγησης (explainable AI) για να κατανοήσουν οι προγραμματιστές πώς το σύστημα καταλήγει στις προτάσεις του αποτελεί κρίσιμη περιοχή έρευνας. Αυτό θα αυξήσει την εμπιστοσύνη στο σύστημα και θα βοηθήσει τους προγραμματιστές να μαθαίνουν από τις προτάσεις. Τέλος, η ανάπτυξη μοντέλων που μπορούν να εκπαιδευτούν συνεχώς από νέα δεδομένα και να προσαρμόζονται σε μεταβαλλόμενες πρακτικές προγραμματισμού αποτελεί σημαντική πρόκληση που χρήζει περαιτέρω διερεύνησης.

6.3 Προκλήσεις και περιορισμοί της εργασίας

Ο πρωταρχικός περιορισμός αφορά το μέγεθος και την ποικιλία του dataset που χρησιμοποιήθηκε για την αξιολόγηση. Το σύστημα δοκιμάστηκε σε έξι αρχεία JavaScript που δημιουργήθηκαν ειδικά για την έρευνα, γεγονός που περιορίζει τη γενικευσιμότητα των αποτελεσμάτων. Ένα μεγαλύτερο και πιο ποικιλόμορφο dataset με πραγματικό κώδικα από διαφορετικά έργα θα προσέφερε πιο αξιόπιστα συμπεράσματα.

Η προσομοίωση αποτελεσμάτων μέσω προκαθορισμένων patterns αντί για πλήρως εκπαιδευμένα μοντέλα βαθιάς μάθησης αποτελεί άλλον σημαντικό περιορισμό. Αν και αυτή η προσέγγιση προσφέρει μια προσεγγιστική εικόνα των δυνατοτήτων των νευρωνικών δικτύων, δεν αντιπροσωπεύει πλήρως την πολυπλοκότητα και τις δυνατότητες πραγματικών εκπαιδευμένων μοντέλων. Η εστίαση σε βασικούς τύπους σφαλμάτων αποτελεί περιορισμό που αφορά την πρακτική εφαρμογή. Στην πραγματικότητα, οι προγραμματιστές αντιμετωπίζουν πιο σύνθετα προβλήματα που περιλαμβάνουν σφάλματα ασφαλείας, προβλήματα απόδοσης, και λογικά σφάλματα που σχετίζονται με πολύπλοκες αλληλεπιδράσεις μεταξύ διαφορετικών τμημάτων του κώδικα. Η έλλειψη εκτίμησης του υπολογιστικού κόστους αποτελεί πρακτικό περιορισμό. Τα μεγάλα προ-εκπαιδευμένα μοντέλα όπως το CodeBERT και το CodeT5 απαιτούν σημαντικούς υπολογιστικούς πόρους, γεγονός που μπορεί να περιορίσει την εφαρμογή τους σε μικρότερα έργα ή σε περιβάλλοντα με περιορισμένους πόρους. Η αβεβαιότητα σχετικά με την ερμηνευσιμότητα των αποτελεσμάτων αποτελεί θεωρητικό περιορισμό. Τα νευρωνικά δίκτυα λειτουργούν συχνά ως "μαύρα κουτιά", καθιστώντας δύσκολη την κατανόηση του πώς καταλήγουν σε συγκεκριμένες προτάσεις διόρθωσης. Αυτό μπορεί να δημιουργήσει προβλήματα εμπιστοσύνης σε κρίσιμες εφαρμογές. Η διαχείριση ψευδώς θετικών αποτελεσμάτων σε μεγαλύτερη κλίμακα αποτελεί πρακτική πρόκληση. Αν και το σύστημα δεν παρήγαγε ψευδώς θετικά στο περιορισμένο dataset, σε πραγματικές εφαρμογές με χιλιάδες αρχεία κώδικα, ακόμα και ένα μικρό ποσοστό ψευδώς θετικών μπορεί να δημιουργήσει σημαντικό φορτίο εργασίας για τους προγραμματιστές. Η εξάρτηση από συγκεκριμένα frameworks και βιβλιοθήκες (PyTorch, Transformers) δημιουργεί περιορισμούς συμβατότητας και συντήρησης. Οι συνεχείς αλλαγές σε αυτές τις τεχνολογίες μπορεί να απαιτήσουν συχνές ενημερώσεις του συστήματος. Τέλος, η έλλειψη αξιολόγησης της απόδοσης του συστήματος σε πραγματικές συνθήκες ανάπτυξης, με πραγματικούς προγραμματιστές και έργα, αποτελεί σημαντικό περιορισμό για την εκτίμηση της πρακτικής χρησιμότητας του συστήματος.

7 Βιβλιογραφία

Albattah, W. and Alzahrani, M. (2024) 'Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach', AI, 5(4), pp. 1743-1758.

Google (2024) TensorFlow: An end-to-end platform for machine learning. Available at: https://www.tensorflow.org/ (Accessed: 23 February 2025).

Keras (2024) Keras: Deep Learning for humans. Available at: https://keras.io/ (Accessed: 23 February 2025).

Κοντούλης, Ι. (2017) 'Μια Ανασκόπηση της Βαθιάς Μάθησης: Θεωρία, Μέθοδοι και Εφαρμογές', Μεταπτυχιακή Διατριβή, Πανεπιστήμιο Πειραιώς.

NVIDIA, Meta AI Research et al. (2024) PyTorch: An open source machine learning framework. Available at: https://pytorch.org/ (Accessed: 23 February 2025).

Pradel, M. and Sen, K. (2017) 'Deep Learning to Find Bugs', Technical Report TUD-CS-2017-0295, Department of Computer Science, TU Darmstadt.