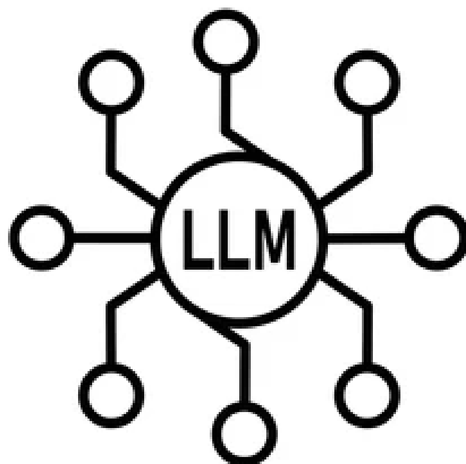
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

Διεθνές Πανεπιστήμιο Ελλάδος

Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

ΠΤΥΧΙΑΚΉ ΕΡΓΑΣΊΑ

Διόρθωση σφαλμάτων σε Προγράμματα JavaScript με Transformer Neural Networks



Φοιτητής:
Ραδής Μάριος Τουμπαλίδης
Αριθμός Μητρώου: 185289

Επιβλέπων:
Κωνσταντίνος Γουλίανας

25 Ιανουαρίου 2025

Τίτλος Δ.Ε Διόρθωση σφαλμάτων σε Προγράμματα Javascript με Transformer Neural Networks
Κωδικός Δ.Ε 23146
Ονοματεπώνυμο φοιτητή Ραδής Μάριος Τουμπαλίδης
Ονοματεπώνυμο εισηγητή Κωνσταντίνος Γουλίανας
Ημερομηνία ανάληψης Δ.Ε. 15-03-2023
Ημερομηνία περάτωσης Δ.Ε. 25-01-2025

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 διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Ραδή Μάριου Τουμπαλίδη που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ’ οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα-/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Αφιέρωση

Το αποτέλεσμα αυτής της διπλωματικής εργασίας το αφιερώνω στους γονείς μου για την στήριξη που μου παρείχαν καθόλα την διάρκεια των σπουδών μου. Επιπλέον, οφείλω την ευγνωμοσύνη μου στον επιβλέποντα καθηγητή Κωνσταντίνο Γουλιάνα για την επιστημονική καθοδήγησή και υποστήριξη που μου παρείχε για την έρευνα και υλοποίηση αυτής της εργασίας.

Πρόλογος

Το αντικείμενο της επεξεργασίας φυσικής γλώσσας και συγκεκριμένα των δικτύων transformer έχει υποστεί ραγδαία ανάπτυξη τα τελευταία χρόνια, γεγονός που έχει προσελκύσει το ερευνητικό ενδιαφέρον μεγάλης μερίδας της επιστημονικής κοινότητας, καθώς και πολλών νέων ανθρώπων να ασχοληθούν με την επιστήμη της πληροφορικής. Το θέμα της επιδιόρθωσης σφαλμάτων σε προγράμματα με την χρήση δικτύων transformer μου προσέλκυσε κατευθείαν το ενδιαφέρον καθώς συνδυάζει δύο αντικείμενα με τα οποία ασχολήθηκα ενεργά καθόλη την διάρκεια των σπουδών μου και πιστεύω πως θα συνεχίσω να ασχολούμαι και στην επαγγελματική μου σταδιοδρομία.

Περίληψη

Η Javascript παρουσιάστηκε στο κοινό το 1995 ως μια απλή ”scripting” γλώσσα και πλέον αποτελεί την πιο διαδεδομένη γλώσσα για διαδικτυακές εφαρμογές, χάρη στην εκφραστικότητα, τη φορητότητά και την ευελιξία της. Η καθιέρωση της οφείλεται σε συγκεκριμένα χαρακτηριστικά που παρουσιάζει, τα οποία δεν εμφανίζονται σε άλλες γλώσσες προγραμματισμού. Κάποια από αυτά είναι η δυνατότητα ενσωμάτωσης της σε έγγραφα HTML, η δυνατότητα να προσθέσει/αφαιρέσει ιδιότητες σε αντικείμενα καθώς αποτελεί αντικειμενοστραφή γλώσσα με βάση τα πρότυπα και όχι τις κλάσεις, καθώς και η δυνατότητα να χρησιμοποιεί εξωτερικές πηγές (π.χ. η ανάκτηση μιας CSS βιβλιοθήκης). Αυτά τα χαρακτηριστικά όμως εκτός από το γεγονός πως δίνουν την ευκαιρία στους προγραμματιστές της να δημιουργήσουν εφαρμογές με δυναμικό περιεχόμενο και όμορφο σχεδιασμό, επιφέρουν τον κίνδυνο εμφάνισης σφαλμάτων ειδικότερα σε εφαρμογές μεγάλης κλίμακας και αυξημένης πολυπλοκότητας. Οι παραπάνω λόγοι οδήγησαν στην ανάπτυξη εργαλείων που εφαρμόζουν ανάλυση στατικά ή δυναμικά σε εφαρμογές JavaScript για την πρόβλεψη και αντιμετώπιση παρόμοιων προβλημάτων. Ωστόσο, τα τελευταία χρόνια, με τη ραγδαία ανάπτυξη της επιστήμης της μηχανικής μάθησης και των τεχνικών επεξεργασίας φυσικής γλώσσας, και συγκεκριμένα της αρχιτεκτονικής *transformer* πολλοί ερευνητές εργάστηκαν, [1] στην ανάπτυξη μεθόδων και εργαλείων για την ανάλυση-/αποσφαλμάτωση λογισμικού με βάση τις αρχιτεκτονικές *transformers*. Σε αντίστοιχο επίπεδο, εφαρμόστηκαν δύο διαφορετικά προ εκπαιδευμένα μοντέλα transformer στην κατανόηση και παραγωγή κώδικα μοντέλα *transformer* με διαφορετικές αρχιτεκτονικές τα οποία συνδυάστηκαν με ένα κλασσικό νευρωνικό δίκτυο ταξινόμησης πολλαπλών κλάσεων, με σκοπό την επιδιόρθωση σφαλμάτων και την ανίχνευση του τύπου σφάλματος προς επιδιόρθωση. Συγκεκριμένα εκπαιδεύτηκαν τα CodeT5[2] και CodeBert[3], όπου το κάθε ένα συνδυάστηκε με ένα δίκτυο ταξινόμησης πολλαπλών κλάσεων δημιουργώντας μία συνδυαστική συνάρτηση απώλειας, αρχικά στην διόρθωση του ελαττωματικού κώδικα και έπειτα στην ανίχνευση τύπων σφαλμάτων προς διόρθωση ανάμεσα από πέντε τύπους σφαλμάτων. Τα αποτελέσματα αυτής της υλοποίησης απέδειξαν πως τα μοντέλα transformer είναι ικανά να εφαρμοστούν στην ανάλυση και διόρθωση προγραμμάτων είτε σε απομονωμένα σημεία, όπως CodeBert που απέδωσε καλύτερα στην διόρθωση μεμονωμένων στοιχείων σε ένα μπλοκ κώδικα, είτε να επιδιορθώσουν ένα ολόκληρο πρόγραμμα κρατώντας την συνοχή του, όπως το CodeT5 που απέδωσε καλύτερα στην επαναδιατύπωση ολόκληρων script που περιείχαν σφάλματα.

Περιεχόμενα

1 Εισαγωγή **2**

1.1 Η γλώσσα προγραμματισμού JavaScript 2

1.1.1 Ιστορική αναδρομή 2

1.1.2 Τα χαρακτηριστικά της JavaScript 3

1.1.3 Προκλήσεις που προκύπτουν στην ανάπτυξη εφαρμογών 7

1.1.4 Αντιμετώπιση τρωτών σημείων σε εφαρμογές JavaScript 8

1.2 Επεξεργασία Φυσικής Γλώσσας 11

1.2.1 Επεξεργασία φυσικής γλώσσας - NLP 11

1.2.2 Παραγωγή Κειμένου 12

1.2.3 Δίκτυα Transformers 13

2 Βιβλιογραφική Ανασκόπηση **16**

2.1 Τεχνικές αποσφαλμάτωσης σε εφαρμογές Javascript 17

2.2 Εφαρμογές παραγωγής και επιδιόρθωσης κώδικα με χρήση μεθόδων μηχανικής μάθησης 20

3 Εύρεση πηγών δεδομένων **24**

3.1 OctoPack & COMMITPACK Σύνολο Δεδομένων 24

3.1.1 COMMITPACK 24

3.1.2 Εξαγωγή δειγμάτων που αφορούν αποσφαλμάτωση & Δημιουργία του συνόλου εκπαίδευσης και αξιολόγησης 26

3.1.3 HumanEvalPack 28

4 Επιλογή & Ανάκτηση Μοντέλων 29

4.1 Επιλογή, ανάκτηση και παραμετροποίηση του Tokenizer 29

4.1.1 Ανασκόπηση των διεργασιών που εφαρμόζει ένας tokenizer 29

4.1.2 Χρησιμοποίηση του Roberta Tokenizer 31

4.2 CodeT5 32

4.3 CodeBert 34

4.4 Ταξινομητής Σφαλμάτων 36

5 Μοντελοποίηση 40

5.1 Προ-επεξεργασία των δεδομένων 40

5.2 Παραγωγή Διορθωμένου Κώδικα 41

5.3 Συνδυασμός Απώλειας Transformer και Ταξινομητή 43

5.4 Περιβάλλον Εκπαίδευσης & Καθορισμός Παραμέτρων 44

5.5 Ανάκληση 45

6 Αποτελέσματα 49

6.1 Αξιολόγηση 49

6.1.1 ROUGE 49

6.1.2 Υλοποίηση 51

6.2 Αποτελέσματα CodeT5 51

6.3 Αποτελέσματα CodeBert 52

6.4 Αποτελέσματα Ταξινομητή 54

7 Θέματα Συζήτησης 57

7.1 Προκλήσεις 57

7.2 Τρόποι επέκτασης της υλοποίησης 59

8 Συμπεράσματα 60

8.1 Θέματα Ασφάλειας & προστασίας δεδομένων 60

Βιβλιογραφία **62**

Κεφάλαιο 1

Εισαγωγή

1.1 Η γλώσσα προγραμματισμού JavaScript

Η γλώσσα προγραμματισμού JavaScript στα πρώτα χρόνια από την κυκλοφορία της χρησιμοποιούνταν για την υλοποίηση ιστοσελίδων με δυναμικό περιεχόμενο, καθώς αυτό θεωρούνταν αδύνατο στα προγράμματα περιήγησης εκείνης της εποχής. Επίσης, είχε διαπιστωθεί πως η ίδια ιστοσελίδα ενδεχομένως να παρουσιάζει διαφορετική συμπεριφορά αναλόγως του προγράμματος περιήγησης στο οποίο εκτελούνταν.

1.1.1 Ιστορική αναδρομή

Η πρώτη έκδοση της γλώσσας, παρουσιάστηκε το 1996 από την εταιρεία Netscape και χρησιμοποιούσε μέθοδο σύνταξης παρόμοια με αυτήν της γλώσσας **C** (ένα script αποτελεί μια ακολουθία από δηλώσεις και αξιώματα), υποστήριζε επτά τύπους δεδομένων (αριθμητικές, συμβολοσειρές και boolean, αντικείμενα και συναρτήσεις, undefined, symbol) καθώς και ένα σύνολο από προ-εγκατεστημένες συναρτήσεις, έτοιμες για χρήση.

Οι εφαρμογές JavaScript μπορούν να ενσωματωθούν σε μια ιστοσελίδα μέσω του HTML στοιχείου *<script>*, [4] όπου κατά την φόρτωση της σελίδας δημιουργείται ένα νέο περιβάλλον εκτέλεσης JavaScript το οποίο περιέχει ένα καθολικό αντικείμενο με ιδιότητες τις προ - εγκατεστημένες συναρτήσεις και μεταβλητές που παρέχει η γλώσσα και τις δηλωμένες καθολικές μεταβλητές από κάθε script. Σε μεταγενέστερες εκδόσεις της γλώσσας οι συντάκτες της αποφάσισαν να εισάγουν το στοιχείο εκτέλεσης με βάση τα συμβάντα (δηλαδή η εμφάνιση ενός συμβάντος ενεργοποιεί την εκτέλεση ενός script ή συνάρτησης), γεγονός που διεύρυνε τις δυνατότητες ανάπτυξης εφαρμογών με δυναμικό περιεχόμενο.

Με την ενασχόληση όλο και περισσότερων προγραμματιστών στον τομέα της ανάπτυξης διαδικτυακών εφαρμογών και τον ανταγωνισμό που είχε δημιουργηθεί με αντίστοιχες γλώσσες προγραμματισμού για το διαδίκτυο αλλά και προγράμματα περιήγησης, ο οργανισμός υπεύ-

θυνος για τη συντήρηση της γλώσσας εργάστηκε στην ενσωμάτωση προτύπων σύνταξης στην γλώσσα διότι η ανάπτυξη της είχε πραγματοποιηθεί με γρήγορους ρυθμούς γεγονός που οδήγησε στην εμφάνιση προβλημάτων όσον αφορά την διαχείριση των τύπων της γλώσσας και το περιβάλλον εκτέλεσης της. Πολλές εφαρμογές παρουσίαζαν διαφορετική συμπεριφορά, αναλόγως του προγράμματος περιήγησης στο οποίο εκτελούνταν, το οποίο οφείλεται στις διαφορές που παρουσίαζαν τα προγράμματα περιήγησης και οι scripting γλώσσες που χρησιμοποιούνταν εσωτερικά τους. Επομένως θεωρήθηκε αναγκαία η ανάπτυξη ενός *standard* το οποίο θα λειτουργούσε ως ένα σύνολο κανόνων σύνταξης για τις scripting γλώσσες σε διαδικτυακά περιβάλλοντα (μέχρι τότε δεν είχε εδραιωθεί η JavaScript ως η πιο διαδεδομένη γλώσσα για το διαδίκτυο). Τον Νοέμβριο του 1996, εκδόθηκε το πρότυπο **EcmaScript**, το οποίο μαζί με τους οργανισμούς που το σύνταξαν αποτελεί μέχρι σήμερα το βασικό πρότυπο σύνταξης εφαρμογών JavaScript.

Η JavaScript, από τη δημοσίευση του πρώτου προτύπου EcmaScript το 1996, έχει υποστεί τεράστιες αλλαγές και εξέλιξη, που την έχουν μετατρέψει από μια απλή γλώσσα scripting σε μια από τις πιο ισχυρές και ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού. Στα πρώτα της βήματα, η JavaScript είχε περιορισμένες δυνατότητες, επικεντρωμένες κυρίως στην αλληλεπίδραση με το περιβάλλον του χρήστη στις ιστοσελίδες. Ωστόσο, η υιοθέτηση του προτύπου EcmaScript οδήγησε σε μια σταθερή βάση για την περαιτέρω ανάπτυξη της γλώσσας.

Με τις επόμενες εκδόσεις του EcmaScript, όπως το ES6 το 2015, εισήχθησαν χαρακτηριστικά όπως οι συναρτήσεις βέλους, οι σταθερές και μεταβλητές με let/const, οι κλάσεις και οι υποσχέσεις (Promises), τα οποία απλοποίησαν και ενίσχυσαν τον τρόπο γραφής του κώδικα. Παράλληλα, η ανάπτυξη εργαλείων όπως το Node.js [5] επέτρεψε τη χρήση της JavaScript και στον server-side[1] προγραμματισμό, επεκτείνοντας τις δυνατότητές της πέρα από το πρόγραμμα περιήγησης.

Σήμερα, με τη συνεχή υποστήριξη της κοινότητας και την ενσωμάτωση νεότερων τεχνολογιών, η JavaScript είναι απαραίτητη για την ανάπτυξη σύγχρονων εφαρμογών, αποδεικνύοντας την ευελιξία και την προσαρμοστικότητά της σε ένα συνεχώς εξελισσόμενο τεχνολογικό τοπίο.

1.1.2 Τα χαρακτηριστικά της JavaScript

Η JavaScript από τη στιγμή της δημιουργίας της, κατάφερε να ξεχωρίσει όχι μόνο ως γλώσσα προγραμματισμού αλλά και ως θεμέλιο της διαδικτυακής εμπειρίας. Η ευελιξία της, σε συνδυασμό με τη συνεχή εξέλιξη των προτύπων και την υποστήριξη μιας ζωντανής κοινότητας, την κατέστησαν μοναδική στον κόσμο της τεχνολογίας.Ωστόσο, αυτό που την έκανε πραγματικά να ξεχωρίσει στην ανάπτυξη διαδικτυακών εφαρμογών είναι κάποια δυναμικά χαρακτηριστικά τα οποία άλλαξαν τον τρόπο με τον οποίο οι προγραμματιστές αλληλεπιδρούν με τις εφαρμογές και τους χρήστες. Αυτά τα χαρακτηριστικά αξίζει να εξεταστούν περαιτέρω.

Αρχικά, η ενσωμάτωση του DOM (**D**ocument **O**bject **M**odel) διευκόλυνε σε μεγάλο βαθμό την ανάπτυξη εφαρμογών. Το DOM αποτελεί μια αναπαράσταση της ιεραρχίας των στοιχείων σε

[1] Η εφαρμογές που εκτελούνται σε εξυπηρετητές και διαμοιράζονται δεδομένα με εφαρμογές πελάτη

ένα HTML έγγραφο όπου κάθε κόμβος είναι ένα HTML στοιχείο (το οποίο μεταφράζεται στην JavaScript ως ένα αντικείμενο), οι ακμές του οποίου οδηγούν στους κόμβους παιδιά του αρχικού κόμβου. Ο συγκεκριμένος τρόπος αναπαράστασης ενός HTML εγγράφου έδωσε την ευκαιρία στους προγραμματιστές JavaScript να εργαστούν εύκολα με τα περιεχόμενα κάθε στοιχείου HTML, καθώς και τη μορφοποίησης τους. Ειδικότερα, οι προγραμματιστές μέσω του DOM και το χαρακτηριστικό της εκτέλεσης με βάση τα συμβάντα, μπορούν να προσθέσουν, να αφαιρέσουν αλλά και να επεξεργαστούν στοιχεία HTML με βάση τις δράσεις του χρήστη.

Στα πρώτα χρόνια χρησιμοποίησης της, οι εφαρμογές JavaScript ενσωματώνονταν αποκλειστικά στα προγράμματα περιήγησης μέσω των οποίων απέστελλαν δεδομένα που εισήγαγαν οι χρήστες μέσω μιας HTML φόρμας σε κάποιον **server**, ο οποίος με την σειρά του επέστρεφε ένα ανανεωμένο έγγραφο HTML προς προβολή στο πρόγραμμα περιήγησης, το οποίο έπρεπε να προσπελαστεί κάθε φορά που γινόταν μια επικοινωνία του πελάτη (εφαρμογή JavaScript) με έναν server. Ο συγκεκριμένος τρόπος επικοινωνίας της εφαρμογής με τον server ωστόσο, καθιστούσε δύσκολη την ανάπτυξη διαδραστικών εφαρμογών με μεγάλο βαθμό πολυπλοκότητας, καθώς αύξανε και τον χρόνο εκτέλεσης, με συμπέρασμα να υποβαθμίζεται η εμπειρία του χρήστη. Το εξής πρόβλημα έσπευσαν να επιλύσουν εταιρείες που είχαν την ανάγκη δημιουργίας εφαρμογών τέτοιας κλίμακας (π.χ. τα Gmail και GoogleMaps από την Google). Σύμφωνα με τους συντάκτες του [4] αυτές οι προσπάθειες οδήγησαν στην δημιουργία εργαλείων όπως η διεπαφή ανάπτυξης εφαρμογών (API) XMLHTTP και του DHTML (Dynamic HTML) τα οποία έδωσαν την δυνατότητα στα προγράμματα περιήγησης (δηλ. τους διερμηνευτές της JavaScript) να μεταφέρουν και να ανακτούν δεδομένα με κάποιον server χωρίς την επαναπροσπέλαση του ανανεωμένου εγγράφου HTML, δηλαδή με ασύγχρονο τρόπο. Έπειτα από περαιτέρω επεκτάσεις των λειτουργιών των παραπάνω εργαλείων από την κοινότητα των προγραμματιστών JavaScript, η ασύγχρονη λειτουργία της JavaScript ονομάστηκε AJAX (Asynchronous JavaScript and XML). Αυτή η καινοτομία επέτρεψε στους προγραμματιστές να δημιουργήσουν απρόσκοπτες και ανταποκρινόμενες διεπαφές χρήστη. Οι εφαρμογές θα μπορούσαν πλέον να ανακτούν δεδομένα από server, να ενημερώνουν δυναμικά το περιεχόμενο και να αλληλεπιδρούν με τους χρήστες σε πραγματικό χρόνο, μιμούμενοι τη λειτουργικότητα των εφαρμογών εκτός του διαδικτύου. Το AJAX κατάφερε να αποσυνδέσει την ανάκτηση δεδομένων από την διαδικασία εκφόρτωσης μιας ιστοσελίδας (rendering), γεγονός το οποίο εκτός από την δημιουργία ποιοτικότερων εφαρμογών και βελτίωση της εμπιρείας τους χρήστη οδήγησε και στην εδραίωση της JavaScript ως την βασικότερη τεχνολογία για την ανάπτυξη διαδικτυακών εφαρμογών.

Όσο το εύρος χρησιμοποίησης της JavaScript μεγάλωνε μέλη της κοινότητας της άρχισαν να επεκτείνουν τις λειτουργικότητες και τα χαρακτηριστικά της, τα οποία ενσωματώνονταν στις βασικές λειτουργίες της γλώσσας μέσω του προτύπου Ecmascript, το οποίο ενημέρωνε τους κανόνες του ανά διαστήματα. Εκτός από τα βασικά χαρακτηριστικά που αναλύθηκαν παραπάνω , τα οποία εδραίωσαν την γλώσσα σε παγκόσμιο επίπεδο, στο αποτέλεσμα αυτό συνείσφεραν και κάποιες περισσότερο τεχνικές λειτρουγικότητες και χαρακτηριστικά τα οποία δύναται να διακριθούν σε γλωσσικά, δυναμικά, και χαρακτηριστικά περιβάλλοντος.

Γλωσσικά Χαρακτηριστικά

Σύμφωνα με την ερμηνεία των συντακτών του [6] η JavaScript διαθέτει συνολικά επτά τύπους δεδομένων, οι οποίοι είναι:

- Undefined: O τύπος που έχουν όλες οι μεταβλητές πριν αρχικοποιηθούν
- Boolean, Number, String: Για δυαδικές, αριθμητικές και συμβολοσειρές μεταβλητές αναλόγως
- Symbol
- Object

Ένα ακόμη χαρακτηριστικό το οποίο συνδέεται με τους τύπους δεδομένων στην JavaScript αφορά την δυνατότητα των μεταβλητών να αλλάζουν "σιωπηλά" τύπο ή/και τιμή κατά την εκτέλεση (implicit type casting) σύμφωνα με τη σημασιολογία μετατροπής τύπου που ορίστηκε στις προδιαγραφές της γλώσσας από το πρότυπο EcmaScript το 2015 (βλ. σχήμα 1.1) . Επιπλέον, η γλώσσα υποστηρίζει και συναρτησιακό προγραμματισμό, το οποίο συνήθως εκφράζεται με την εκτενή χρήση ανώνυμων συναρτήσεων (βλ. σχήμα 1.2), καθώς υπάρχει και η δυνατότητα δυναμικής παραγωγής κώδικα μέσω της προ εγκατεστημένης συνάρτησης *eval* η οποία μπορεί να δεχτεί μία συμβολοσειρά που αναπαριστά κώδικα JavaScript ως είσοδο και να τον εκτελέσει.

Τα τρία παραπάνω χαρακτηριστικά προσφέρουν στους προγραμματιστές μεγαλύτερη εκφραστικότητα και ευελιξία στην ανάπτυξη εφαρμογών καθώς και την ευκολότερη διαχείριση των πολύπλοκων αλληλεπιδράσεων με τον χρήστη που ενδέχεται να έχει μία διαδικτυακή εφαρμογή.

```javascript
let count = "5"; // A string
// Implicit type casting converts "5" to a number
let result = count * 2;
console.log(result); // Output: 10
```

Σχήμα 1.1: Παράδειγμα "σιωπηλής" αλλαγής τύπου σε μεταβλητή

```javascript
setTimeout(() => {
  console.log("This message is displayed after 2 seconds.");
}, 2000);
```

Σχήμα 1.2: Χρησιμοποιείται η προ εγκατεστημένη συνάρτηση *setTimeout* η οποία δέχεται ως όρισμα μια ανώνυμη συνάρτηση η οποία υλοποιείται εσωτερικά των ορισμάτων της *setTimeout*

Δυναμικά Χαρακτηριστικά

- Μπορεί να ενσωματωθεί σε αρχεία HTML
- Είναι prototype-based αντικειμενοστραφής γλώσσα και όχι class-based. Δηλαδή ένα αντικείμενο μπορεί να προσθέσει/αφαιρέσει ιδιότητες δυναμικά (βλ. σχήμα 1.3). Ωστόσο αυτό μπορεί να οδηγήσει σε μη εμφανείς μετατροπές τύπου μεταβλητών και σφάλματα τύπου.
- Μπορεί να χρησιμοποιήσει μη έμπιστες τρίτες πηγές κατά την εκτέλεση οι οποίες δεν εμφανίζονται πριν την εκτέλεση (σχήμα 1.4 η ανάκτηση μιας CSS βιβλιοθήκης, πολυμέσα που δεν βρίσκονται στον κώδικα της εφαρμογή πελάτη)

```
function User(name) {
  this.name = name;
}

User.prototype.greet = function () {
  return `Hello, ${this.name}!`;
};

const user1 = new User("Alice");
console.log(user1.greet()); // Output: "Hello, Alice!"

User.prototype.farewell = function () {
  return `Goodbye, ${this.name}!`;
};

console.log(user1.farewell()); // Output: "Goodbye, Alice!"
```

Σχήμα 1.3: Παράδειγμα που επιδεικνύει την δυνατότητα επεξεργασίας των ιδιοτήτων ενός αντικειμένου σε οποιοδήποτε σημείο.

```
const link = document.createElement("link");
link.rel = "stylesheet";
link.href = "https://cdnjs.cloudflare.com/ajax/libs/font-awesome.css";
document.head.appendChild(link);

link.onload = () => {
  const content = document.createElement("div");
  content.innerHTML = '<i class="fas fa-smile"></i> Hello, styled world!';
  document.body.appendChild(content);
};
```

Σχήμα 1.4: Παράδειγμα χρήσης μη έμπιστων τρίτων πηγών που γίνονται προσβάσιμες κατά την εκτέλεση

Χαρακτηριστικά Web Περιβάλλοντος

Όπως αναφέρθηκε και παραπάνω τα DOM και AJAX αποτελούν τα βασικά στοιχεία για την εδραίωση της JavaScript ως η *de facto* γλώσσα για διαδικτυακές εφαρμογές, σημαντικό είναι να εξεταστούν και οι τύποι σφαλμάτων και εξαιρέσεων της γλώσσας. Σύμφωνα με τον ορισμό από τους συντάκτες του [6] οι βασικότεροι τύποι σφαλμάτων στην Javascript είναι το *TypeError* το οποίο είναι πολύ σημαντικό λόγου της δυνατότητας της γλώσσας να αλλάζει ”σιωπηλά” τύπο σε μεταβλητές, χαρακτηριστικό το οποίο μπορεί να οδηγήσει σε απρόβλεπτα αποτελέσματα (π.χ. διαίρεση με μεταβλητή που έχει τύπο String) και το *ReferenceError* το οποίο υφίσταται για να διαχειριστεί τις αναφορές σε μεταβλητές που δεν έχουν δηλωθεί σε μια δεδομένη στιγμή της εκτέλεσης του προγράμματος.

Ανεξαρτήτως από την ευελιξία που προσφέρουν τα ξεχωριστά χαρακτηριστικά της JavaScript, επιπλέον την καθιστούν και ιδανική γλώσσα για αρχάριους προγραμματιστές με περιορισμένο επιστημονικό επίπεδο στην ανάπτυξη ολοκληρωμένων συστημάτων και εφαρμογών. Συγκεκριμένα, τα δυναμικά χαρακτηριστικά όπως η ”σιωπηλή” αλλαγή τύπου, η ανεξέλεγκτη επεξεργασία ιδιοτήτων σε αντικείμενα και πολλά ακόμη χαρακτηριστικά της γλώσσας αφαιρούν την ανάγκη ορθής διαχείρισης των δεδομένων και των περιπτώσεων αιχμής που ενδέχεται να εμφανιστούν σε ένα σύστημα,μειώνοντας σημαντικά τον χρονικό κύκλο ανάπτυξης εφαρμογών JavaScript καθώς και το αναγκαίο επίπεδο γνώσης για την υλοποίηση τους.

1.1.3 Προκλήσεις που προκύπτουν στην ανάπτυξη εφαρμογών

Ωστόσο, η ευελιξία και η ευκολία ανάπτυξης εφαρμογών JavaScript δημιουργεί και κάποιες ευπάθειες, ειδικότερα όταν αναπτύσσονται από αρχάριους προγραμματιστές, οι οποίες ενδέχεται να οδηγήσουν σε ανεξήγητες συμπεριφορές της εφαρμογής όταν δεν διαχειρίζονται με τον ορθό τρόπο (όπως γίνεται αντιληπτό και στο παράδειγμα παρακάτω).

```
<form id="myForm">
  <label for="age">Enter your age:</label>
  <input type="text" id="age" name="age">
  <button type="submit">Submit</button>
</form>
```

Σχήμα 1.5: HTML φόρμα που ο χρήστης πληκτρολογεί μία είσοδο και πυροδοτείτε το συμβάν *onSubmit*

```
document.getElementById("myForm").
addEventListener("submit", function (event) {
  event.preventDefault();
  const ageInput = document.getElementById("age").value;
  const doubledAge = ageInput * 2;
  ...
});
```

Σχήμα 1.6: Αν ο χρήστης εισάγει μη αριθμητική τιμή (π.χ. Είκοσι) τότε η JavaScript θα επιχειρήσει να το πολλαπλασιάσει , οδηγώντας σε ανεξήγητα αποτελέσματα

1.1.4 Αντιμετώπιση τρωτών σημείων σε εφαρμογές JavaScript

Η κοινότητα των προγραμματιστών προσπάθησε να αντιμετωπίσει τα προβλήματα που προκύπτουν από τα χαρακτηριστικά της JavaScript μέσω διαφόρων μεθόδων και εργαλείων. Ένα από τα μεγαλύτερα ζητήματα ήταν η σιωπηλή αλλαγή τύπου, το οποίο συχνά οδηγούσε σε απρόβλεπτες συμπεριφορές. Για παράδειγμα, η σύγκριση *"" == 0* επιστρέφει *true*, κάτι που μπορεί να προκαλέσει σφάλματα. Η ανάπτυξη εργαλείων συντακτικού ελέγχου όπως το **ESLint**[2] επιχείρησαν να δημιουργήσουν ένα πλαίσιο ανίχνευσης τέτοιων σφαλμάτων πριν την εκτέλεση, προτείνοντας τη χρήση του αυστηρού ισοδύναμου (===) αντί του χαλαρού (==) στο συγκεκριμένο παράδειγμα.
Η φύση της JavaScript, μέσω των prototype-based objects, μπορεί να προκαλέσει σύγχυση, ειδικά στους νέους προγραμματιστές, καθώς η κληρονομικότητα είναι λιγότερο εμφανής από ό,τι στις γλώσσες που βασίζονται σε κλάσεις. Για την επίλυση αυτού του ζητήματος, η κοινότητα εισήγαγε τη συντακτική δομή των κλάσεων στο EcmaScript 2015, η οποία παρέχει έναν πιο σαφή και οργανωμένο τρόπο για τη διαχείριση της κληρονομικότητας.
Ακόμη ένα πρόβλημα που εμφανιζόταν, ειδικότερα σε εφαρμογές με μεγάλη πολυπλοκότητα και μεγάλους όγκους κώδικα, ήταν αυτό του *unreachable code block*, δηλαδή τη δημιουργία ακροατών συμβάντων[3] για ένα συμβάν το οποίο δεν ενεργοποιούταν ποτέ σε μία εφαρμογή, αυξάνοντας την πολυπλοκότητα και μειώνοντας την αναγνωσιμότητα μίας βάσης κώδικα. Για

[2] https://eslint.org
[3] Το κομμάτι κώδικα που εκτελείται μετά την ενεργοποίηση ενός συμβάντος

την αντιμετώπιση τέτοιων προβλημάτων η κοινότητα δημιούργησε εργαλεία όπως το **RxJS**[4] για τη διαχείριση ροών δεδομένων, παρέχοντας εργαλεία για την ακύρωση events και τον έλεγχο της κατάστασης.
Όπως αναφέρθηκε παραπάνω, η καθιέρωση της τεχνολογίας AJAX έδωσε στην γλώσσα την δυνατότητα ασύγχρονης επικοινωνίας με server για την ανταλλαγή δεδομένων. Με την πάροδο των χρόνων και την παρουσίαση νέων αλλαγών στους συντακτικούς κανόνες καθώς και στις βασικές λειτουργίες της γλώσσας από την κοινότητα και το πρότυπο EcmaScript εισήχθησαν στην γλώσσα λέξεις κλειδιά όπως τα *async/await* και η έννοια των αντικειμένων "υπόσχεσης" (Promise Object) για την αποδοτικότερη υλοποίηση εφαρμογών με ασύγχρονη επικοινωνία. Συγκεκριμένα, η λέξη *await* αποδίδεται σε συναρτήσεις για να δηλωθεί πως πραγματοποιείται ασύγχρονη επικοινωνία εσωτερικά τους και θα εμφανιστεί και η λέξη *await* για να δηλωθεί η αναμονή για την απάντηση της ασύγχρονης επικοινωνίας. Με βάση την ερμηνεία του Turcotte και συν. [7] η λογική των αντικειμένων "υπόσχεσης" είναι αναγκαία στις ασύγχρονες επικοινωνίες καθώς αντικατοπτρίζει την κατάσταση της επικοινωνίας που μπορεί να βρίσκεται μεταξύ των εξής τριών τιμών: σε εκκρεμότητα, εκπληρώθηκε και απέτυχε. Θεωρώντας ως παράδειγμα μιας ασύγχρονης επικοινωνίας την συνάρτηση *fetchUserData* στο σχήμα 1.7, ένα αντικείμενο υπόσχεσης δημιουργίται αυτόματα όταν κληθεί η συνάρτηση (όπως γίνεται στο σχήμα 1.8). Ο καθορισμός της συνάρτησης ως *async* υποδηλώνει πως η συνάρτηση σίγουρα θα επιστρέψει το αντικείμενο "υπόσχεσης" που δημιουργείται με κάποια κατάσταση. Στην πρώτη χρήση της λέξης await σταματά η εκτέλεση μέχρι η συνάρτηση *fetch* επιλυθεί με μία απόκριση (επιτυχία ή αποτυχία),η οποία επιχειρεί ανάκτηση δεδομένων από ένα API, η κατάσταση του αντικειμένου "υπόσχεσης" που επιστρέφει η *fetchUserData* είναι: σε εκκρεμότητα. Μόλις η *fetch* επιστρέψει μια επιτυχής HTTP απάντηση, το αντικείμενο "υπόσχεσης" περνά σε κατάσταση εκπλήρωσης και αναλόγως σε κατάσταση αποτυχίας στην περίπτωση που η απάντηση του HTTP αιτήματος που κάνει η *fetch* δεν είναι επιτυχής. Εφόσον η απάντηση είναι επιτυχής, το *await response.json()* περιμένει να μετατραπεί το περιεχόμενο της απάντησης σε αντικείμενο JavaScript, και εδώ, το αντικείμενο "υπόσχεσης" παραμένει σε κατάσταση "σε εκκρεμότητα" μέχρι να ολοκληρωθεί η μετατροπή. Όταν ολοκληρωθεί η μετατροπή του περιεχομένου σε αντικείμενο JavaScript, το αντικείμενο "υπόσχεσης" της *fetchUserData* εκπληρώνεται και επιστρέφει το τελικό αντικείμενο *userData*.
Η δυνατότητα υλοποίησης ασύγχρονων επικοινωνιών στις εφαρμογές JavaScript προσφέρει ευελιξία και περισσότερη εκφραστικότητα όσον αφορά τις αλληλεπιδράσεις με τον χρήστη, ωστόσο αυξάνει την πολυπλοκότητα ,την δυσκολία συντήρησης του ασύγχρονου κώδικα, καθώς και την απαιτούμενη εμπειρία για την υλοποίηση της, γεγονός που δημιουργεί χώρο εμφάνισής ανεξήγητων (ακόμη και μη ανιχνεύσιμων) σφαλμάτων και συμπεριφορών στις εφαρμογές. Όπως διατύπωσαν και οι συντάκτες του [7] υπάρχουν διάφορα μοτίβα σφαλμάτων που εμφανίζονται σε ασύγχρονες εφαρμογές, όπως για παράδειγμα η δημιουργία μιας *aynsc* συνάρτησης χωρίς την χρήση των αποκρίσεων await. Τέτοια μοτίβα σφαλμάτων και ανεξήγητων συμπεριφορών δημιουργούν προβλήματα σε εφαρμογές JavaScript που σηνήθως αφορούν την λειτουργικότητα της εφαρμογής, το χρόνο εκτέλεσης ή ακόμη και θέματα ασφάλειας και ακεραιότητας των δεδομένων που μεταφέρονται στις ασύγχρονες επικοινωνίες.

[4]https://rxjs.dev

```
async function fetchUserData(userId) {
  try {
    const response = await fetch(`https://example.com/users/${userId}`);
    if (!response.status === 200) {
      throw new Error(`Error: ${response.text}`);
    }
    const userData = await response.json();
    return userData;
  } catch (error) {
    console.error("Failed to fetch user data:", error.message);
  }
}
```

Σχήμα 1.7: Παράδειγμα χρήσης των λέξεων κλειδιά async, await

```
(async () => {
  const user = await fetchUserData(1);
  if (user) {
    console.log("User data:", user);
  }
})();
```

Σχήμα 1.8: Παράδειγμα χρήσης μιας συνάρτησης ασύγχρονης επικοινωνίας και δημιουργίας και εκπλήρωσης του αντικειμένου ”υπόσχεσης”

Συνοψίζοντας, ο τρόπος δομής και σύνταξης εφαρμογών JavaScript προσφέρει μεγάλη ευελιξία και προσαρμοστικότητα στην ανάπτυξη εφαρμογών, διευρύνοντας ωστόσο τον χώρο εμφάνισης σφαλμάτων και συμπεριφορών που περιπλέκουν τον χρήστη της εφαρμογής όταν δεν αντιμετωπίζονται ορθά. Για αυτό τον λόγο όμως η κοινότητα των προγραμματιστών JavaScript έχει αναπτύξει εργαλεία όπως το *npm*[5] το οποίο είναι ο διαχειριστής πακέτων λογισμικού για την JavaScript ώστε να είναι ευκολότερη η κοινοποίηση βιβλιοθηκών και γενικών λειτουργιών για την αντιμετώπιση των ευπαθειών που αναλύθηκαν παραπάνω (όπως προαναφέρθηκε το εργαλείο *ESlint*) κατά την εκτέλεση του κώδικα, καθώς και επαναχρησιμοποίηση λειτουργιών που συναντιούνται σε πολλές εφαρμογές (π.χ. η βιβλιοθήκη axios που προσφέρει λειτουργίες στην μορφή αντικειμένων και συναρτήσεων για την πραγματοποίηση HTTP ερωτημάτων). Ένα κομμάτι της κοινότητας όμως, και αρκετοί ερευνητές στον τομέα της μηχανικής λογισμικού, έχουν εργαστεί στην ανάπτυξη εργαλείων και μεθοδολογιών με σκοπό την ανάλυση και διερμήνευση κώδικα και των ευπαθειών που ενδέχεται να παρουσιάζει πριν την εκτέλεση του (καθώς η JavaScript δεν διαθέτει μεταγλωττιστή του κώδικα σε κώδικα μηχανής). Σύμφωνα με τους συντάκτες του [6] αυτές οι έρευνες διακρίνονται σε εξής ερευνητικά θέματα: στατική και δυναμική ανάλυση τα οποία κατέχουν και το μεγαλύτερο ποσοστό έρευνας, την

[5] https://www.npmjs.com

συντακτική προτυποποίηση, και την ασφάλεια των εφαρμογών, κάποιες από τις οποίες θα αναλυθούν στην ενότητα 2.1.

1.2 Επεξεργασία Φυσικής Γλώσσας

Η επιστήμη της μηχανικής μάθησης έχει παρουσιάσει αξιοσημείωτη πρόοδο τα τελευταία χρόνια, ξεκινώντας από απλά συστήματα βασισμένα σε κανόνες και καταλήγοντας σε πολύπλοκους αλγορίθμους, με βασικό στοιχείο τα νευρωνικά δίκτυα, πού έχουν τη δυνατότητα να "μάθουν" από δεδομένα. Ειδικότερα στον τομέα της επεξεργασίας φυσικής γλώσσας (NLP), αλγόριθμοι όπως το bag-of-words και το n-grams ήταν εξαρτώμενα από στατιστικές μεθόδους με μικρή εμβέλεια, και μπορούσαν μόνο να προσπελάσουν δεδομένα κειμένου χωρίς να έχουν τη δυνατότητα ανάκτησης νοήματος ή/και σημασιολογίας που βρίσκεται στο ύφος της γλώσσας.Όσο ο όγκος των διαθέσιμων δεδομένων μεγάλωνε σε αντιστοιχία με τις δυνατότητες υπολογιστικής ισχύος, η ανάπτυξη αλγορίθμων μηχανικής μάθησης σε συνδυασμό με την βαθιά μάθηση εφαρμόστηκε και σε εργασίες επεξεργασίας φυσικής γλώσσας. Η ανάπτυξη όσο στην έρευνα τόσο και στην βιομηχανία στον τομέα των νευρωνικών δικτύων, και συγκεκριμένα των αναδρομικών (RNNs) και συνελικτικών (CNNs) νευρωνικών δικτύων σε συνδυασμό με τη δημιουργία των μηχανισμών προσοχής (attention mechanisms) έδωσαν τη δυνατότητα σε αλγορίθμους NLP να μοντελοποιήσουν σχέσεις και εξαρτήσεις στοιχείων μέσα σε ελεύθερο κείμενο. Ώσπου η δημοσιοποίηση των αρχιτεκτονικών transformer [8] δημιούργησε ένα πλαίσιο στο οποίο αλγόριθμοι NLP χρησιμοποιούν τον μηχανισμό προσοχής για να ανακτήσουν εννοιολογική πληροφορία από το κείμενο καθιστώντας τους ιδανικά συστήματα για εργασίες όπως κατανόηση, κατηγοριοποίηση, περίληψη, μετάφραση και παραγωγή κειμένου. Επιπλέον, είναι ευνόητο πως το συντακτικό γλωσσών προγραμματισμού high-level[6] όπως η JavaScript, είναι βασισμένες και ακολουθούν την λογική της φυσικής γλώσσας, επομένως οι ερευνητές από πολύ πρώιμο στάδιο επιχείρησαν να εφαρμόσουν μοντέλα αρχιτεκτονικής transformer (όπως τα Bert, GPT) σε εργασίες διερμήνευσης και αποσφαλμάτωσης λογισμικού.Αυτή η ενότητα εξετάζει πώς η εξέλιξη της μηχανικής μάθησης, ιδιαίτερα της επεξεργασίας φυσικής γλώσσας, έχει εντάξει στον δρόμο της και στην επιστήμη της μηχανικής λογισμικού.

1.2.1 Επεξεργασία φυσικής γλώσσας - NLP

Ο τομέας της επεξεργασίας φυσικής γλώσσας αποτελούσε αντικείμενο έρευνας πριν την εδραίωση των νευρωνικών δικτύων από τα μέσα του προηγούμενου αιώνα με τους ερευνητές να επικεντρώνονται κυρίως σε εργασίες αυτόματης μετάφρασης. Στα πρώιμα στάδια του τομέα NLP οι ερευνητές συνεργάζονταν με γλωσσολόγους για την δημιουργία γλωσσικών κανόνων που βασιζόντουσαν στην σειρά των λέξεων σε μία πρόταση ,με τους οποίους οι υπολογιστές εκείνης της εποχής θα επιχειρούσαν να μεταφράσουν ελεύθερο κείμενο, όπως το πείραμα από το πανεπιστήμιο Georgetown και την εταιρεία IBM [9] όπου επιτεύχθηκε η μηχανική μετάφραση ρώ-

[6] Οι γλώσσες προγραμματισμού για πιο αφηρημένες λειτουργίες που δεν είναι πολύ κοντά στην γλώσσα μηχανής

σικων προτάσεων. Με την πάροδο των χρόνων και την ραγδαία ανάπτυξη του διαδικτύου και των δυνατοτήτων των υπολογιστών, δημιουργήθηκαν αλγόριθμοι και στατιστικές μέθοδοι που είχαν την δυνατότητα να μαθαίνουν από γλωσσικά δεδομένα, οδηγώντας στην ενσωμάτωση τους σε διάφορες εφαρμογές όπως μηχανές αναζήτησης και υπηρεσίες αυτόματης μετάφρασης όπως το Google Translate. Αυτές οι μέθοδοι βασιζόντουσαν πάνω σε αλγορίθμους όπως τα δίκτυα LSTM [10] ,τα κρυμμένα μοντέλα Markov [11] και τα δίκτυα SVM [12]. Μεταγενέστερα, η δημοσίευση μοντέλων όπως το *word2vec* [13] κατάφερε να επιλύσει το κυριότερο αδιέξοδο που εμφανιζόταν σε εφαρμογές NLP και αφορούσε την διανυσματική αναπαράσταση στοιχείων κειμένου επιτρέποντας την κατανόηση γλωσσικής πληροφορίας και σημασιολογίας από υπολογιστικά συστήματα. Οι δυνατότητες που πρόσφερε το μοντέλο word2vec σε συνδυασμό με την εδραίωση των αρχιτεκτονικών κωδικοποιητή - αποκωδικοποιητή, στην οποία ένα μοντέλο έχει την δυνατότητα να παράξει κείμενο από κείμενο (sequence to sequence modeling) και την τεχνική της βαθιάς μάθησης αποτέλεσαν το βασικότερο δομικό στοιχείο για την ευρύτερη χρήση εφαρμογών NLP. Τέλος, η δημοσίευση της αρχιτεκτονικής transformer [8], αποτέλεσε το κλειδί για την ραγδαία ανάπτυξη νέων αλγορίθμων και μεθοδολογιών που ειδικεύονται σε εργασίες NLP που υφίσταται τα τελευταία χρόνια, όπου μοντέλα όπως τα GPT και Bert χρησιμοποιούνται για διάφορες υπό εργασίες που αφορούν την κατανόηση και παραγωγή φυσικής γλώσσας, όπως η σημασιολογική αναζήτηση, η κατηγοριοποίηση κειμένου , η περίληψη και κυριότερα η παραγωγή κειμένου.

1.2.2 Παραγωγή Κειμένου

Η παραγωγή κειμένου φυσικής γλώσσας (Natural Language Generation - NLG) αποτελεί υποσύνολο του τομέα NLP και ορίζεται ως η διαδικασία παραγωγής φράσεων και προτάσεων με νόημα σε μορφή φυσικής γλώσσας. Ωστόσο, τα μοντέλα NLG για να αποδόσουν εξαρτόνται από την τροφοδότηση τους με δομημένα δεδομένα δηλαδή δεδομένα που έχουν διερμηνευτεί. Αυτή η διαδικασία της διερμήνευσης ελεύθερου αδόμητου κειμένου σε δομημένα δεδομένα πραγματοποιείται από άλλο υποσύνολο του τομέα NLP, την κατανόηση φυσικής γλώσσας (Natural Language Understanding - NLU). Επομένως η εργασία παραγωγής κειμένου εμπεριέχει τις εργασίες απλής παραγωγής κειμένου από δομημένα δεδομένα και τις εργασίες παραγωγής κειμένου από κείμενο.
Οι περιπτώσεις χρήσης των μοντέλων NLG ποικίλουν και χρησιμοποιούν διαδεδομένες ή καινούριες αρχιτεκτονικές transformer με διάφορες αλλαγές που σχετίζονται στο πώς επεξεργάζονται το κείμενο. Οι περιπτώσεις μπορούν να διακριθούν ως:

- Παραγωγή και συμπλήρωση ιστοριών: όπου ένα μοντέλο δεδομένου ενός κειμένου μικρού μήκους θα επιχειρήσει να τελειώσει την φράση με βάση το κείμενο που του τροφοδοτήθηκε. Τα μοντέλα αυτής της εργασίας εκπαιδεύονται να παράγουν την επόμενη λέξη δεδομένου μιας ακολουθίας από στοιχεία κειμένου τα οποία δεν έχουν ετικέτες.

- Παραγωγή κειμένου από κείμενο (sequence to sequence): Αυτά τα μοντέλα εκπαιδεύονται στο να αντιστοιχίζουν ζεύγη δειγμάτων κειμένου , το οποίο εμπεριέχεται σε πολλές εργασίες όπως περίληψη, μετάφραση κ.α.

- Μοντέλα που εκπαιδεύονται να ακολουθούν συγκεκριμένες οδηγίες που προσφέρονται σε φυσική γλώσσα (Instruction Models). Αυτά τα μοντέλα εκπαιδεύονται να διερμηνεύουν ερωτήματα και εντολές σε μορφή φυσικής γλώσσας χωρίς την απαίτηση τροφοδότησης τους με δομημένα μηνύματα.

Παρόλο που οι περιπτώσεις χρήσεις των μοντέλων NLG διαφέρουν υπάρχει ευελιξία στο εύρος χρήσης τους και καθορίζεται από την δομή που έχει το σύνολο δεδομένων (π.χ. αν περιέχει ετικέτες) και την μορφή που θέλει ο προγραμματιστής να έχουν οι έξοδοι που παράγει το μοντέλο. Η ευελιξία επεκτείνεται και στις περιπτώσεις διερμήνευσης και παραγωγής κώδικα, όπου υπάρχουν περιπτώσεις που ένα μοντέλο παραγωγής κώδικα μπορεί να θεωρηθεί και instruction model, παραδείγματος χάριν ένα μοντέλο που τροφοδοτείται με την περιγραφή μιας λειτουργίας και παράγει τον αντίστοιχο κώδικα. Στην περίπτωση της αποσφαλμάτωσης τα πιο διαδεδομένα μοντέλα ανήκουν στην ομάδα μοντέλων παραγωγής κειμένου από κείμενο με κάποιες παραμετροποιήσεις, όπου το μοντέλο δέχεται τον κώδικα με σφάλματα και ίσως κάποιες περαιτέρω πληροφορίες (π.χ. σχόλια, περιγραφή λειτουργίας, δενδρική αναπαράσταση κώδικα) και επιχειρεί να παράξει τον διορθωμένο κώδικα.

1.2.3 Δίκτυα Transformers

Τα πιο αποτελεσματικά μοντέλα μεταγωγής συμβολοσειρών είναι βασισμένα είτε σε αναδρομικά είτε σε συνελικτικά νευρωνικά δίκτυα τα οποία εμπεριέχουν αρχιτεκτονικές κωδικοποιητή/αποκωδικοποιητή και μερικές φορές με την προσθήκη ενός ”μηχανισμού προσοχής”. Στις αρχιτεκτονικές κωδικοποιητή/αποκωδικοποιητή σύμφωνα με τους συντάκτες του [8], ο κωδικοποιητής συσχετίζει μία ακολουθία συμβόλων με μία ακολουθία από συνεχόμενες αναπαραστάσεις, οι οποίες έπειτα τροφοδοτούνται στον αποκωδικοποιητή ώστε να παράξει μια ακολουθία συμβόλων ως έξοδο, ένα στοιχείο την φορά. Σε κάθε βήμα εκπαίδευσης, το μοντέλο είναι αυτό-παλινδρομικό, χρησιμοποιώντας την παραγόμενη ακολουθία ως επιπλέον είσοδο για την επόμενη ακολουθία. Σε ένα δίκτυο transformer με αρχιτεκτονική κωδικοποιητή/αποκωδικοποιητή ο κωδικοποιητής αποτελείται από έξι όμοια στρώματα. Κάθε τέτοιο στρώμα περιέχει δύο υπό στρώματα,

- ένα δίκτυο μηχανισμού ”προσοχής” πολλαπλών κεφαλών
- ένα πλήρως συνδεδεμένο δίκτυο τροφοδότησης προς τα εμπρός με βάση την θέση

τα οποία θα αναλυθούν παρακάτω. Μετά από κάθε υπόστρωμα εφαρμόζεται υπολειμματική σύνδεση(residual connection) η οποία ακολουθείται από ένα στρώμα κανονικοποίησης. Οι υπολειμματικές συνδέσεις κατέχουν σημαντικό ρόλο στα δίκτυα transformers καθώς λειτουργούν ως ασπίδες ασφαλείας σε ένα δίκτυο, εξασφαλίζοντας πως ακόμη και στην περίπτωση που ένα υπόστρωμα αποτύχει να ”μάθει” τη σημασιολογία μιας ακολουθίας, η αρχική πληροφορία θα διατηρηθεί και για τα επόμενα υποστρώματα. Συγκεκριμένα, στα υποστρώματα που εφαρμόζεται ο μηχανισμός προσοχής, η υπολειμματική σύνδεση εξασφαλίζει πως η αρχική αναπαράσταση της ακολουθίας εισόδου (embedding), ή η έξοδος από κάποιο υπόστρωμα του

κωδικοποιητή, δεν θα αντικατασταθεί πλήρως από τον μηχανισμό προσοχής, διατηρώντας μία ισορροπία μεταξύ νέων χαρακτηριστικών που παράχθηκαν από τις μεταγωγές της αρχικής αναπαράστασης και της ίδιας της αρχικής αναπαράστασης.
Ο αποκωδικοποιητής αποτελείται κι αυτός από έξι όμοια στρώματα, όπου εφαρμόζονται τα δύο ίδια υποστρώματα που έχει και ο κωδικοποιητής με την προσθήκη ενός ακόμη υποστρώματος μηχανισμού «προσοχής» πολλαπλών κεφαλών στην αρχή της στοίβας το οποίο εφαρμόζει τον μηχανισμό χρησιμοποιώντας την έξοδο του κωδικοποιητή. Ομοίως, εφαρμόζεται υπολειμματική σύνδεση που ακολουθείται από στρώμα κανονικοποίησης.

Μηχανισμός Απόδοσης Προσοχής

Σύμφωνα με την ερμηνεία των συντακτών του [8], η διαδικασία απόδοσης "προσοχής" μπορεί να περιγραφεί ως η αντιστοίχιση ενός ερωτήματος και ενός συνόλου ζευγαριών "κλειδί", "τιμή" με μια έξοδο όπου όλοι οι παραπάνω όροι αποτελούν διανύσματα. Η έξοδος της συνάρτησης είναι το σταθμισμένο άθροισμα των "τιμών", όπου το "βάρος" που ανατίθεται σε κάθε "τιμή" υπολογίζεται από μια συνάρτηση συμβατότητας του ερωτήματος με το αντίστοιχο "κλειδί" (η έξοδος της συνάρτησης είναι ένας αριθμός που δηλώνει πόσο συμβατό είναι να βρίσκεται η λέξη πάνω στην οποία εφαρμόζεται ο μηχανισμός προσοχής σε ένα συγκεκριμένο σημείο στην ακολουθία εισόδου).Οι συντάκτες ονόμασαν τον τύπο της τιμής προσοχής *Scaled Dot-Product Attention* και υπολογίζεται ως εξής:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Όπου Q, K, V τα διανύσματα των ερωτημάτων, κλειδιών και τιμών και d_k αποτελεί την διάσταση των διανυσμάτων των ερωτημάτων και των κλειδιών.Ωστόσο, παρατηρήθηκε πως είναι αποδοτικότερη η ταυτόχρονη αναπαράσταση των διανυσμάτων Q, K, V σε διαφορετικές διαστάσεις για κάθε διάνυσμα d_Q, d_K, d_V πολλαπλές φορές και κάθε φορά να εφαρμόζεται το *Scaled Dot-Product Attention* παράλληλα σε κάθε αναπαράσταση των ερωτημάτων, κλειδιών και τιμών αναλόγως και τέλος να τα συνενώσουν με διάσταση d_V. Για την ευκολότερη κατανόηση της λογικής του υπολογισμού των τιμών προσοχής κάθε παράλληλη εφαρμογή του *Scaled Dot-Product Attention* στα αντίστοιχα Q, K, V ονομάστηκε "κεφαλή" και όλη αυτή η διαδικασία *Multi-head-Attention*. Τέλος, κάθε "κεφαλή" συνενώνεται μέσω ενός στρώματος γραμμικού μετασχηματισμού, καταλήγοντας σε μία γραμμική αναπαράσταση των τιμών προσοχής. Δηλαδή η έξοδος του *Multi-head-Attention* αντιπροσωπεύει μία "στιλβωμένη" αναπαράσταση της ακολουθίας εισόδου αντιστοιχίζοντας για κάθε στοιχείο της ένα διάνυσμα που εκφράζει τις συσχετίσεις του στοιχείου με τα υπόλοιπα στοιχεία στην ακολουθία εισόδου. Με αυτό τον τρόπο κάθε κεφαλή μπορεί να "εστιάσει" σε διαφορετικές συσχετίσεις που υπάρχουν στην ακολουθία εισόδου και μπορεί να αφορούν το συντακτικό την σημασιολογία κ.α.
Η έξοδος που παράγεται σε κάθε στρώμα *Multi-head-Attention* αποτελεί μία μαθηματική μήτρα και τροφοδοτείται στο δίκτυο τροφοδότησης προς τα εμπρός. Δίκτυα τέτοιου τύπου υπάρχουν σε όλα τα υποστρώματα και του κωδικοποιητή και αποκωδικοποιητή, το οποίο εφαρμόζεται σε κάθε θέση της μήτρας και πραγματοποιεί δύο γραμμικούς μετασχηματισμούς φιλτράροντας

την έξοδο του πρώτου από ένα στρώμα ενεργοποίησης (activation layer) με βάση την συνάρτηση ReLU. Επίσης οι συντάκτες επισημαίνουν πως τα δίκτυα *Multi-head-Attention* είναι δυνατό να εφαρμοστούν και σε αρχιτεκτονικές μόνο με κωδικοποιητή ή μόνο αποκωδικοποιητή.

Μετασχηματισμός του κειμένου σε διανυσματική μορφή

Είναι προφανές πως όλες οι αρχιτεκτονικές νευρωνικών δικτύων δουλεύουν χρησιμοποιώντας διανύσματα, γεγονός που δημιουργεί την ανάγκη μετασχηματισμού του κειμένου που θα τροφοδοτηθεί στο μοντέλο σε διάνυσμα ή μαθηματική μήτρα. Αυτή η διαδικασία επιτυγχάνεται αντιστοιχίζοντας διανυσματικές αναπαραστάσεις (embeddings) με ένα σύνολο λέξεων ή φράσεων με βάση ένα λεξιλόγιο, και θα αναλυθεί εκτενέστερα στην ενότητα 4.1.

Κεφάλαιο 2

Βιβλιογραφική Ανασκόπηση

Όπως αναφέρθηκε και στην ενότητα 1.1.4 ένα μεγάλο μέρος της ερευνητικής κοινότητας εργάστηκε τόσο στην γενικότερη ανάλυση και αιτιολόγηση των ευπαθειών που εμφανίζονται στις εφαρμογές JavaScript όσο και στην δημιουργία και εδραίωση τεχνικών εργαλείων καθώς και μεθοδολογιών για την αντιμετώπιση τους. Ένα τέτοιο εργαλείο είναι το ESlint το οποίο ενσωματώνεται εύκολα στον κώδικα μιας εφαρμογής. Λειτουργεί εσωτερικά στο πρόγραμμα ανάπτυξης κώδικα του προγραμματιστή και σε κάθε αλλαγή τους αναλύει ολόκληρη την εφαρμογή ελέγχοντας αν υπάρχει παραβίαση των κανόνων του, όπου στις περιπτώσεις που υπάρχει ειδοποιεί τον προγραμματιστή μέσω του προγράμματος ανάπτυξης κώδικα (όπως στο σχήμα 2.1), καθώς και σε συγκεκριμένες περιπτώσεις προτείνεται και κάποια διόρθωση στον κώδικα. Το εργαλείο παρέχει έναν μεγάλο όγκο κανόνων που σχετίζονται με συντακτικά, λογικά σφάλματα και πιθανά σφάλματα εκτέλεσης, σε συνδυασμό με την δυνατότητα καθορισμού προσωποποιημένων κανόνων.
Ένα ακόμη εργαλείο που βοήθησε στην αντιμετώπιση ευπαθειών της γλώσσας που σχετίζονται με τα συντακτικά χαρακτηριστικά της είναι το TypeScript[1]. Το TypeScript είναι υπερσύνολο της JavaScript, με επιπλέον χαρακτηριστικά που το καθιστούν πιο ισχυρό και στατικό. Δημιουργήθηκε από τη Microsoft και προορίζεται να λύσει ορισμένες από τις αδυναμίες της JavaScript προσθέτοντας περισσότερα στοιχεία αντικειμενοστραφή προγραμματισμού (παραδείγματος χάριν στο σχήμα 2.2). Το TypeScript είναι ένα εργαλείο που επιτρέπει τη δημιουργία πιο ασφαλούς, οργανωμένου και διατηρήσιμου κώδικα. Παρότι διαθέτει μια μικρή καμπύλη εκμάθησης, προσφέρει σημαντικά οφέλη, ειδικότερα σε έργα μεγάλης πολυπλοκότητας που συμμετέχουν πολλοί προγραμματιστές.
Η ανάπτυξη εργαλείων όπως το ESlint και το TypeScript, καθώς και οι ολοένα και πιο σύνθετες επεκτάσεις που βασίζονται σε αυτά τα δύο εργαλεία, προώθησε τη JavaScript να γίνει μία από τις πιο ευρέως χρησιμοποιούμενες γλώσσες γενικής χρήσης στον κόσμο.

[1]https://www.typescriptlang.org

Σχήμα 2.1: Παράδειγμα των προειδοποιητικών ειδοποιήσεων που προσφέρει το ESlint, στην συγκεκριμένη περίπτωση δηλώνεται η σταθερά a και μετά επιχειρείται αλλαγή της τιμής της.

```
interface User {
  id: number;
  name: string;
  email: string;
}

class UserService {
  private users: User[] = [];

  addUser(user: User): void {
    this.users.push(user);
  }
}

const userService = new UserService();
const user = userService.getUserById(1);
```

Σχήμα 2.2: Παράδειγμα χρήσης των λειτουργιών του TypeScript

Στις παρακάτω ενότητες του κεφαλαίου θα αναλυθούν μέθοδοι και τεχνικές αποσφαλμάτωσης κώδικα JavaScript χωρίς την χρήση μεθόδων μηχανικής μάθησης καθώς και τεχνικές και συστήματα που χρησιμοποιούν μεθόδους μηχανικής μάθησης για αποσφαλμάτωση κώδικα.

2.1 Τεχνικές αποσφαλμάτωσης σε εφαρμογές Javascript

Σε αυτή την ενότητα παρατίθεται επεξήγηση κάποιων μεθόδων και τεχνικών αποσφαλμάτωσης και διερμήνευσης των ευπαθειών που εμφανίζονται σε εφαρμογές JavaScript, και σχετίζονται με τις ασύγχρονες επικοινωνίες και την ανάλυση εφαρμογών με εκτέλεση με βάση τα

συμβάντα, την ποσοτική και ποιοτική ανάλυση εφαρμογών με βάση το γράφημα κλήσεων[2].
Με την πάροδο τον χρόνων η κοινότητα των προγραμματιστών δημιούργησε εργαλεία όπως το jQuery [14] και το πακέτο βιβλιοθηκών React [15] τα οποία προσέφεραν περαιτέρω στρώματα αφαίρεσης με ευνόητο συντακτικό πάνω στην ήδη υπάρχουσα λειτουργία με βάση τα συμβάντα, της JavaScript. Ωστόσο, η κυκλοφορία του Node.js [5] παρουσίασε την δυνατότητα ανάπτυξης εφαρμογών JavaScript και σε server, μεταφέροντας την εκτέλεση με βάση τα συμβάντα και τις ασύγχρονες λειτουργίες και σε εφαρμογές server, όπως μία λειτουργία που χρειάζεται πρόσβαση σε μία βάση δεδομένων. Στις εφαρμογές Node.js για server ενδείκνυται ένα συμβάν να αναπαριστά τον HTTP σύνδεσμο (πχ. GET api/users) και τον κώδικα που τρέχει όταν αποζητάται ο σύνδεσμος να αναπαριστά τον ακροατή συμβάντος.
Η μεταφορά της εκτέλεσης με βάση τα συμβάντα δημιούργησε κάποιες ευπάθειες που δεν υπήρχαν σε εφαρμογές πελάτη. Κάποιες από αυτές σύμφωνα με τους συντάκτες του [16] αφορούν την ονοματοδοσία των συμβάντων και τον ακροατών τους τα οποία είναι απλές συμβολοσειρές οδηγώντας έτσι σε πιθανά σφάλματα που αφορούν την σύνδεση συμβάντος και ακροατή, όπου κάποια παράμετρος δεν έχει οριστεί ορθά οδηγώντας στον ακροατή του συμβάντος να μην ενεργοποιείτε ποτέ (dead listener), είτε ένα συμβάν να μην χρησιμοποιείτε ποτέ (lost event). Αυτό το πρόβλημα έσπευσαν να λύσουν οι ερευνητές, εφαρμόζοντας έναν απλό ταξινομιτή με σκοπό να εξακριβωθεί αν για ένα δεδομένο συμβάν α θεωρείται σπάνιος ένας ακροατής e. Η κατηγοριοποίηση αυτή μετράται συγκρίνοντας την πιθανότητα εμφάνισης ενός τυχαίου ζεύγους $p_{(a,e)}$ στο σύνολο δεδομένων που ανέκτησαν από εφαρμογές ανοιχτού κώδικα στην πλατφόρμα GitHub, με μία δεδομένη τιμή "κατωφλιού" p_a (π.χ. έστω ότι $p_a = 0.5$ και $p_{(a,e)} = 0.48$ τότε ο ακροατής e θεωρείται σπάνιος για συμβάντα με όνομα a). Με αυτή την μεθοδολογία οι συντάκτες κατάφεραν να εξάγουν μοτίβα ζευγών συμβάντος, ακροατή που δύναται να οδηγήσουν σε dead listeners.
Ομοίως, παρόμοια προβλήματα που προκύπτουν σε κομμάτια κώδικα που διαχειρίζονται τα συμβάντα και τους ακροατές τους επιχειρούν να αναλύσουν οι συντάκτες του [17] ακολουθώντας μία διαφορετική στρατηγική. Εκτός από τις περιπτώσεις των dead listeners, σημαντική ευπάθεια είναι και οι καταστάσεις που ένα συμβάν εκδηλώνεται για ένα αντικείμενο αλλά αυτό το αντικείμενο δεν έχει δηλωμένο κάποιον ακροατή για αυτό το συμβάν, η ακόμη και ασύγχρονες συναρτήσεις που δεν καλούνται ποτέ είτε άμεσα είτε έμμεσα από το σύστημα συμβάντων καθώς και διεργασίες που δύναται να εκτελεστούν ασύγχρονα είτε σύγχρονα (όπως η προσπέλαση αρχείου κειμένου) και υλοποιούνται λανθασμένα και με τους δύο τρόπους δημιουργώντας έτσι μια κατάσταση "αγώνα". Εκφάνσεις τέτοιων τύπων σφαλμάτων μπορεί να ανιχνεύσει το εργαλείο που προτείνουν οι συντάκτες, το οποίο είναι βασισμένο στο γράφημα κλήσεων. Το γράφημα κλήσεων είναι μία πολύ σημαντική δομή αναπαράστασης ενός προγράμματος που μπορεί να βοηθήσει σε πολλούς διαφορετικούς τομείς που αφορούν την ανάπτυξη, συντήρηση, επιδιώρθωση και τον ποιοτικό έλεγχο ενός προγράμματος, παραδείγματος χάριν το γράφημα κλήσεων μπορεί να χρησιμοποιηθεί από ένα εργαλείο για την ανίχνευση διαρροής δεδομένων από "μολυσμένες" εισόδους σε μία συνάρτηση. Συγκεκριμένα κατασκευάζεται ένα κατευθυνόμενο γράφημα όπου κόμβοι είναι εκφράσεις από τον πηγαίο κώδικα. Ένας κόμβος μπορεί να περιέχει διαφορετικούς τύπους ακμών, οι οποίες μπορεί να συνδέουν:

[2]Ένα σύνολο από κόμβους οι οποίοι αναπαριστούν συναρτήσεις και κατευθυνόμενες ακμές μεταξύ κόμβων οι οποίες αναπαριστούν τις κλήσεις μεταξύ συναρτήσεων

1. μια έκφραση με την δήλωση μίας συνάρτησης,
2. μια έκφραση δήλωσης ακροατή με το σώμα μιας συνάρτησης
3. μια έκφραση πυροδότησης ενός συμβάντος με μία συνάρτηση
4. μια έκφραση δήλωσης ακροατή με μία έκφραση πυροδότησης ενός συμβάντος

Χρησιμοποιώντας τους παραπάνω τέσσερις τύπους ακμών από το γράφημα κλήσεων με βάση τα συμβάντα, είναι δυνατό να αναγνωριστεί αν ο ακροατής για ένα συμβάν μπορεί να εκτελεστεί. Συγκεκριμένα, για να είναι δυνατό να εκτελεστεί ένας ακροατής l ενός συμβάντος e που βρίσκεται στο γράφημα θα πρέπει να υπάρχουν ακμές τύπου 2,3 και 4 που τα εμπεριέχουν. Οι εφαρμογές που περιέχουν λειτουργίες με βάσει τα συμβάντα έχουν χαρακτηριστεί ως επιρρεπείς σε σφάλματα ειδικότερα αυτές με μεγάλο βαθμό πολυπλοκότητας και αριθμό διεργασιών. Αυτό ισχύει καθώς πολλές φορές η υλοποίηση μιας ασύγχρονης λειτουργίας εμπεριέχει εμφωλευμένες δηλώσεις συμβάντων ή/και των ακροατών τους, γεγονός που δυσκολεύει την κατανόηση της ροής εκτέλεσης του κώδικα το οποίο συνεπάγεται και μία πολύπλοκη διαδικασία αποσφαλμάτωσης. Για την αντιμετώπιση τέτοιων προβλημάτων εισήχθηκε στην γλώσσα η έννοια της υπόσχεσης, η οποία αναπαριστά την τιμή που επιστράφηκε από μία ασύγχρονη διεργασία από το EcmaScript του 2015, καθώς επιχειρήθηκε να επιλυθεί και από κάποιες βιβλιοθήκες όπως το jQuery [14]. Ωστόσο, η σημασιολογία γύρω από τα αντικείμενα υπόσχεσης είναι αρκετά περίπλοκη χωρίς να υπάρχει και κάποιος μηχανισμός ελέγχου του συντακτικού, δημιουργώντας έτσι χώρο για εμφάνιση σφαλμάτων. Το εξής πρόβλημα επιχειρούν να επιλύσουν οι συντάκτες του [18] δημιουργώντας αρχικά ένα μαθηματικό σύστημα λ_p το οποίο αντικατοπτρίζει την σημασιολογία των αντικειμένων υπόσχεσης και έπειτα βασισμένοι στο λ_p δημιουργούν το "γράφημα υποσχέσεων" για την ανίχνευση σφαλμάτων που αφορούν αντικείμενα υπόσχεσης, με παρόμοια μεθοδολογία όπως οι συντάκτες του [17]. Συγκεκριμένα το γράφημα υποσχέσεων είναι κατευθυνόμενο και αποτυπώνει την ροή εκτέλεσης και δεδομένων σε προγράμματα με αντικείμενα υπόσχεσης. Οι κόμβοι του αφορούν απόδοση τιμών, απόδοση αντικειμένου υπόσχεσης και δηλωμένη ή ανώνυμη συνάρτηση. Οι ακμές αποτυπώνουν τις σχέσεις μεταξύ των κόμβων του γραφήματος οι οποίες μπορεί να είναι:

1. η εκπλήρωση/αποτυχία μίας υπόσχεσης, όταν η ακμή συνδέει κόμβο τιμής με κόμβο υπόσχεσης
2. την αντίδραση μίας υπόσχεσης (δηλαδή το μπλοκ κώδικα που εκτελείται όταν εκπληρωθεί ή αποτύχει), όταν η ακμή συνδέει κόμβο υπόσχεσης με κόμβο συνάρτησης
3. συσχέτιση δύο υποσχέσεων όταν συνδέει δύο κόμβους υποσχέσεων. Η μεταγενέστερη θα έχει πάντα την κατάσταση της προγενέστερης υπόσχεσης.
4. την επιστροφή τιμής μίας συνάρτησης, όταν συνδέεται κόμβος συνάρτησης με κόμβο τιμής.

Οι συντάκτες του [18] ερμηνεύουν πως μέσω του γραφήματος σε συνδυασμό με το γράφημα κλήσεων ή/και απλούς συντακτικούς ελέγχους είναι δυνατό να ανιχνευθούν σφάλματα που οι

ίδιοι τα διακρίνουν σε σφάλματα που αφορούν αντικείμενα που δεν εκπληρώνονται ή απορρίπτονται ποτέ, όταν στο γράφημα υποσχέσεων υπάρχει μία υπόσχεση p που δεν συνδέεται πουθενά με κάποια ακμή τύπου 1 ή τύπου 3. Επιπλέον, μόνο μέσω του γραφήματος μπορούν να ανιχνεύσουν περιπτώσει όπου λείπει από ένα αντικείμενο υπόσχεσης η αντίδραση (συνάρτηση εκπλήρωσης ή απόρριψης), βρίσκοντας αντικείμενα υπόσχεσης στο γράφημα που δεν εμπεριέχουν εξερχόμενες ακμές τύπου 2. Κάποια ακόμη μοτίβο σφαλμάτων που υποστηρίζουν ότι γίνεται να ανιχνευτούν μέσω του γραφήματος, αφορούν σφάλματα όπου μία συνάρτηση αντίδρασης δεν επιστρέφει κάποια τιμή, ακόμη και σφάλματα που ακολουθούν το μοτίβο όπου ένα αντικείμενο υπόσχεσης εκπληρώνεται ή απορρίπτεται πάνω από μία φορές, ελέγχοντας αν στο γράφημα για ένα αντικείμενο υπόσχεσης υπάρχουν πάνω από μία ακμές τύπου 2. Επιθεωρώντας το γράφημα και το μπλοκ κώδικα μίας υπόσχεσης μπορούν να ανιχνεύσουν την ύπαρξη μη αναγκαίων υποσχέσεων. Τέλος, υποστηρίζουν πως οι προγραμματιστές μπορούν να χρησιμοποιήσουν το γράφημα υποσχέσεων σαν εργαλείο παρατήρησης της συμπεριφοράς των λειτουργιών που χρησιμοποιούν αντικείμενα υπόσχεσης, για να επιβεβαιώσουν την ορθή εκτέλεση τους.

2.2 Εφαρμογές παραγωγής και επιδιόρθωσης κώδικα με χρήση μεθόδων μηχανικής μάθησης

Όπως αναλύθηκε και στην προηγούμενη ενότητα, δομές δεδομένων όπως τα γραφήματα κλήσεων και τα δέντρα βοηθούν στην διερμήνευση και επιδιόρθωση λογισμικού σε γενικό βαθμό ή ακόμη και σε ειδικευμένο πλαίσιο όπως το γράφημα υποσχέσεων, χωρίς να χρειάζεται η πλήρης ανάγνωση του γραμμή πως γραμμή. Κάτι παρόμοιο επιχείρησαν και οι συντάκτες του [19] παραθέτοντας τα δίκτυα αφηρημένης σύνταξης (Abstract Syntax Networks), όχι με σκοπό διόρθωσης κώδικα αλλά για την εργασία παραγωγής κώδικα. Τα δίκτυα αφηρημένης σύνταξης αποτελούν ένα πλαίσιο μοντελοποίησης εργασιών παραγωγής κώδικα και επιχειρούν την αναπαράσταση προγραμμάτων και εφαρμογών με βάση αφηρημένα δέντρα συντακτικού (Abstract Syntax Trees). Ο κάθε κόμβος του δέντρου αφορά ένα τμήμα του προγράμματος και μπορεί να έχει διαφορετικούς τύπους, που μπορεί να είναι:

- Πρωτογενής τύπος, που αφορά απόδοση τιμών και την δημιουργία αντικειμένων.
- Σύνθετος τύπος, που αφορά σχήματα γλώσσας όπως εκφράσεις για παράδειγμα μία συνθήκη boolean, μία κλήση συνάρτησης και δηλώσεις οντοτήτων (π.χ. δήλωση συνάρτησης ή κλάσης κ.α.), τέτοιες μπορεί να είναι και δηλώσεις ή αναφορές σε συναρτήσεις

Ένας κόμβος μπορεί να έχει:

- Έναν τελικό γόνο - singular cardinality (π.χ. ένας κόμβος δήλωσης κλάσης έχει πάντα τελικό γόνο τον δομητή του)

- Προαιρετικούς γόνους - optional cardinality (π.χ. μια έκφραση ”return”)
- Διαδοχικούς γόνους - sequential cardinality (π.χ. το σώμα μίας συνάρτησης)

Οι συντάκτες θεώρησαν πως τα μοντέλα μεταγωγής συμβολοσειρών με τη χρήση κωδικοποιητή/αποκωδικοποιητή, όπως και τα δίκτυα transformers, αποδείχθηκαν τα αποτελεσματικότερα για εργασίες φυσικής γλώσσας και κατ' επέκταση εργασίες που αφορούν γλώσσες προγραμματισμού (ένα πρόγραμμα λογισμικού είναι δυνατόν να θεωρηθεί κείμενο φυσικής γλώσσας). Ωστόσο, είναι δεδομένο πως ένα κείμενο γλώσσας προγραμματισμού (ειδικότερα υψηλού επιπέδου) χρησιμοποιεί τη φυσική γλώσσα παρουσιάζοντας έμμεσα επιπρόσθετη πληροφορία για τη λειτουργία που υλοποιεί (μια έκφραση if πρέπει να έχει συγκεκριμένη σύνταξη και να ακολουθεί κάποιους κανόνες). Επομένως, συστήσουν μία αρχιτεκτονική κωδικοποιητή/αποκωδικοποιητή, η οποία επεξεργάζεται embeddings που παράχθηκαν από Abstract Syntax Trees, τα οποία αναπαριστούν ένα πρόγραμμα. Επιπλέον, ο αποκωδικοποιητής χρησιμοποιεί και τον μηχανισμό απόδοσης προσοχής στην ακολουθία εισόδου. Το μοντέλο που δημιούργησαν εκπαιδεύτηκε στην συγγραφή κώδικα για την δημιουργία χαρακτήρων σε ένα επιτραπέζιο παιχνίδι καρτών. Η είσοδος του μοντέλου αναπαριστάται ως μια συλλογή από στοιχεία (tokens) που αντιστοιχίζονται συνδυαστικά με την περιγραφή του χαρακτήρα (ακολουθία από στοιχεία) μαζί και τις ιδιότητές της (ένα στοιχείο). Το μοντέλο ακολουθεί μία απλή αρχιτεκτονική κωδικοποιητή/αποκωδικοποιητή με ιεραρχικό μηχανισμό προσοχής.Ο κωδικοποιητής χρησιμοποιεί αμφίδρομα LSTM δίκτυα για να δημιουργήσει το embedding κάθε στοιχείου το οποίο έπειτα τροφοδοτείται σε ένα δίκτυο τροφοδότησης προς τα εμπρός. Ο αποκωδικοποιητής δομείται ως μια κατακόρυφα συνδεδεμένη συλλογή αμοιβαία αναδρομικών ενοτήτων (δίκτυο LSTM), όπου κάθε ενότητα αναπαριστά κόμβους του Abstract Syntax Tree, και οι έξοδοι τους συνδυάζονται αντικατοπτρίζοντας τη δημιουργία του δέντρου. Ο μηχανισμός προσοχής εφαρμόζεται και σε επίπεδο ενότητας και σε επίπεδο στοιχείου (token).
Παρόλο που ο βασικός στόχος των ερευνητών στο συγκεκριμένο παράδειγμα δεν ήταν η επιδιόρθωση αλλά η παραγωγή κώδικα (η επιδιόρθωση κώδικα εμπεριέχει και την παραγωγή), κατάφεραν να συνδέσουν μία δομή δεδομένων που συνδέει μια εφαρμογή λογισμικού, τα Abstract Syntax Trees με μία πρώιμη αρχιτεκτονική που χρησιμοποιεί μηχανισμό προσοχής, γεγονός που συνέβαλε στην δημιουργία ενός νέου αντικειμένου έρευνας στον τομέα της παραγωγής κώδικα το οποίο απασχολεί μεγάλο μέρος της ερευνητικής κοινότητας.
Πάνω στην έρευνα του [19] βασίστηκαν και οι συντάκτες του [20] όπου χρησιμοποιώντας την αναπαράσταση ενός μπλοκ κώδικα (όχι ολόκληρο το πρόγραμμα) ως Abstract Syntax Tree παρουσιάζουν ένα μοντέλο το οποίο εκπαιδεύεται στην εργασία της αναπαράστασης μπλοκ κώδικα (code snippets) ως συνεχή κατανεμημένα διανύσματα όπως στο μοντέλο word2vec. Συγκεκριμένα, το μοντέλο μαθαίνει την ατομική αναπαράσταση (embedding) κάθε μονοπατιού στο δέντρο ενώ ταυτόχρονα μαθαίνει πώς να συγκεντρώνει ένα σύνολο από αυτά, υλοποιώντας με αυτόν τον τρόπο την ανάκτηση ιδιοτήτων που υπάρχουν στο συντακτικό ενός μπλοκ κώδικα. Το μοντέλο μαθαίνει σημασιολογικές συσχετίσεις κατασκευάζοντας μονοπάτια, καθιστώντας το εφαρμόσιμο σε εργασίες όπως περίληψη, κατηγοριοποίηση ή απλή βαθμολόγηση κώδικα. Ισχυρίζονται πως η χρήση των Abstract Syntax Trees μόνο για ένα κομμάτι κώδικα, όχι όπως είναι η κανονική τεχνική που εφαρμόζεται πάνω σε ένα ολοκληρωμένο πρόγραμμα, έπειτα από ανάλυση του ανεπεξέργαστου κειμένου (code snippet) ελαφρύνει την ανάγκη για μεγάλο όγκο δεδομένων για την εκπαίδευση, διότι το μοντέλο δεν χρειάζεται να επανεκπαι-

δεύεται στο συντακτικό και τη σημασιολογία, ανιχνεύοντας έτσι μοτίβα. Ένα κομμάτι κώδικα αναπαριστάται ως ένα πολυσύνολο από πιθανά συντακτικά μονοπάτια και έπειτα μετασχηματίζεται σε διάνυσμα, καταλήγοντας σε ένα υποσύνολο από διανύσματα που αναπαριστούν τα πιθανά συντακτικά μονοπάτια. Αναλόγως το μοντέλο εκπαιδεύεται στην εύρεση των στοιχείων του κάθε μονοπατιού του πολυσυνόλου που πρέπει να αποδώσει μεγαλύτερη προσοχή, καθιστώντας το δυνατό έπειτα στην συλλογή των "σημαντικότερων" στοιχείων από κάθε πιθανό μονοπάτι και την συνένωση τους σε ένα διάνυσμα. Δηλαδή ο μηχανισμός προσοχής υπολογίζει τον μαθημένο σταθμισμένο μέσο όρο για να παράξει ένα τελικό διάνυσμα εξόδου το οποίο αναπαριστά το αρχικό μπλοκ κώδικα και τις συσχετίσεις που υπάρχουν μεταξύ των εκφράσεων εσωτερικά του. Εν κατακλείδι, οι συντάκτες παρουσιάζουν ένα νευρωνικό μοντέλο με μηχανισμό προσοχής, συγκεκριμένα soft attention[3] το οποίο εκπαιδεύτηκε στην παραγωγή ενός διανύσματος αναπαράστασης του αρχικού μπλοκ συλλέγοντας όλες τις αναπαραστάσεις των πιθανών συντακτικών μονοπατιών που προέκυψαν από την αναπαράσταση του μπλοκ ως Abstract Syntax Tree.

Το σημαντικότερο εύρημα των συντακτών αποτελεί η προσπάθεια τους στο να εντάξουν ένα μηχανισμό προσοχής, ο οποίος εφαρμόζεται με σκοπό την διανυσματική αναπαράσταση της σημασιολογίας που εμφανίζεται σε μικρά μπλοκ κώδικα, γεγονός που κατέστησε το μοντέλο ικανό να αντιστοιχήσει το νόημα που "κρύβεται" σε ένα μπλοκ κώδικα με όλες τις δυνατές υλοποιήσεις του (π.χ. η υλοποίηση μιας συνάρτησης που γράφει σε ένα αρχείο μπορεί να εφαρμοστεί με πολλούς τρόπους και το code2vec κατάφερε να αντιστοιχήσει αυτούς τους τρόπους), καθιστώντας το αρκετά ικανό για εφαρμογή σε διάφορες εργασίες που αφορούν την ανάπτυξη λογισμικού, όπως διερμήνευση, βαθμολόγηση και υπολογισμός βαθμού συσχέτισης μεταξύ δύο μπλοκ κώδικα.

Ομοίως, οι συντάκτες του [1] επιχειρούν να εφαρμόσουν αρχιτεκτονικές transformer, συγκεκριμένα τις αρχιτεκτονικές κωδικοποιητή BERT και αποκωδικοποιητή GPT για την αποσφαλμάτωση προγραμμάτων JavaScript, επωφελούμενοι την δενδρική αναπαράσταση μπλοκ κώδικα (Abstract Syntax Trees). Διακρίνουν την διαδικασία εκπαίδευσης των μοντέλων τους σε στάδια, αρχικά μοντελοποιούν τις εισόδους των μοντέλων ως ζεύγη (x, y) όπου x είναι το μπλοκ κώδικα που περιέχει σφάλματα και y το μπλοκ του διορθωμένου κώδικα. Έπειτα εφαρμόζουν τον Roberta tokenizer, τον οποίο εφάρμοσα με παρόμοιο τρόπο στην υλοποίηση μου (θα περιγραφεί αναλυτικά στην ενότητα 4.1.2), για την στοιχειοποίηση των δειγμάτων. Έπειτα, εκπαιδεύουν το μοντέλο τους NSEdit να παράγει ακολουθίες που εκφράζουν μόνο τις αλλαγές που χρειάζεται το μπλοκ κώδικα με σφάλματα. Αναλυτικότερα, τροφοδοτείται στον κωδικοποιητή BERT η ακολουθία από στοιχεία του μπλοκ κώδικα με σφάλματα και από την έξοδο του χρησιμοποιείται η μνήμη του, δηλαδή οι διανυσματικές αναπαραστάσεις που εκφράζουν την σημασιολογία μεταξύ των λέξεων ενός μπλοκ κώδικα καθώς και τις σχέσεις του κάθε στοιχείου με τα υπόλοιπα, αυτός ο όρος συνήθως αναφέρεται ως "κρυφή κατάσταση". Ως είσοδο στον αποκωδικοποιητή GPT συνδυάζεται η μνήμη του κωδικοποιητή με το τρέχον στοιχείο αλλαγής που προέκυψε από την διόρθωση του κώδικα (ground truth edit token) όπου και εκπαιδεύεται με σκοπό να παράγει το επόμενο στοιχείο αλλαγής αυτοπαλινδρομικά. Επιπλέον, παραμετροποιούν τα τελευταία στρώματα του αποκωδικοποιητή ώστε να λειτουργεί με δύο τρόπους, να παράγει τις αλλαγές με την χρήση της συνάρτησης softmax καθώς και τις τοποθεσίες που πρέπει να εφαρμοστεί κάθε αλλαγή μέσω ενός δικτύου τροφοδότησης προς τα εμπρός, το οποίο

[3]τα συναπτικά βάρη κατανέμονται σε όλα τα μονοπάτια που προέκυψαν από το δέντρο

μετασχηματίζει την έξοδο του προ-τελευταίου στρώματος και υπολογίζεται το εσωτερικό γινόμενο της με την μνήμη του κωδικοποιητή, όπου έπειτα τροφοδοτούνται σε ένα δίκτυο softmax το οποίο παράγει ένα διάνυσμα πιθανοτήτων το οποίο εκφράζει την πιθανότητα αλλαγής για το κάθε στοιχείο από το ground truth edit token. Είναι αξιοσημείωτο το γεγονός πως οι Hu, Shi και συν. [1] κατάφεραν να δημιουργήσουν ένα μοντέλο το οποίο παράγει κατευθείαν τις ακολουθίες αλλαγής πάνω σε εσφαλμένο κώδικα δημιουργώντας μία δική τους γραμματική για τις πιθανές δράσεις (δηλ. διαγραφή ή προσθήκη κώδικα) την οποία στοιχειοποίησαν και πρόσθεσαν στο υπάρχον λεξιλόγιο του tokenizer του κωδικοποιητή BERT.
Οι Wang και συν. [2] επιχείρησαν να εφαρμόσουν μια αρχιτεκτονική transformer που περιλαμβάνει κωδικοποιητή αποκωδικοποιητή με βάση το μοντέλο T5 χρησιμοποιώντας και αυτοί την προστιθέμενη πληροφορία της σημασιολογίας ενός μπλοκ κώδικα που παρέχει το Abstract Syntax Tree. Κατάφεραν να υλοποιήσουν ένα μοντέλο γενικού σκοπού με την δυνατότητα να επεξεργάζεται μπλοκ κώδικα για πολλές εργασίες όπως περίληψη και σύνθεση κώδικα, με αρκετή ευκολία καθώς εφάρμοσαν ένα μηχανισμό απόκρυψης των σημαντικών στοιχείων στην ακολουθία εισόδου ώστε το μοντέλο να εκπαιδευτεί στην πρόβλεψη των ”κρυμμένων” σημαντικών στοιχείων στην ακολουθία εισόδου. Περισσότερες λεπτομέρειες του θα αναλυθούν στην ενότητα 4.2 καθώς είναι ένα από τα προ εκπαιδευμένα μοντέλα που χρησιμοποίησα στην υλοποίησή μου.
Σε παρόμοια σκοπιά με τους συντάκτες του [1] εργάστηκαν, εφαρμόζοντας προ εκπαιδευμένα μοντέλα στην διανυσματική αναπαράσταση κώδικα JavaScript πάνω στο σύνολο δεδομένων που κατασκεύασα με ζεύγη κώδικα με σφάλματα και διορθωμένου κώδικα, με την διαφορά πως στην προσωπική μου υλοποίηση τα μοντέλα εξάγουν όλο το αρχικό μπλοκ κώδικα διορθωμένο και έπειτα την προσθήκη ενός ταξινομητή που αποδίδει ένα ή παραπάνω τύπο σφάλματος σε κάθε δείγμα.

Κεφάλαιο 3

Εύρεση πηγών δεδομένων

Η χρήση αρχιτεκτονικών transformer σε εργασίες επεξεργασίας γλωσσών προγραμματισμού αποτελεί σχετικά καινούριο θέμα έρευνας επομένως η ποσότητα των δημόσιων συνόλων δεδομένων είναι μικρή, και στις περισσότερες περιπτώσεις δεν έχει εφαρμοστεί καθόλου προεπεξεργασία ή είναι ελάχιστη στα δείγματα των συνόλων, ειδικότερα στην περίπτωση της διόρθωσης κώδικα που είναι αναγκαία η επισήμανση των δειγμάτων. Επίσης, η δημιουργία προσαρμοσμένου συνόλου δεδομένων μέσω ανοιχτών πλατφορμών όπως το GitHub σε συνδυασμό με την επισήμανση του καθίσταται χρονοβόρα και απαιτεί προηγμένες γνώσεις στην επιστήμη των δεδομένων, επομένως επιλέχθηκε η χρήση του ήμι-επεξεργασμένου συνόλου δεδομένων COMMITPACK μέσω της πλατφόρμας huggingface [21] η οποία θα αναλυθεί στο κεφάλαιο 4.

3.1 OctoPack & COMMITPACK Σύνολο Δεδομένων

Οι συγγραφείς του [22] παρουσιάζουν ένα μεγάλο corpus από δείγματα μπλοκ κώδικα σε 350 διαφορετικές γλώσσες προγραμματισμού, στην μορφή πολλαπλών συνόλων δεδομένων με σκοπό την χρήση τους στην υλοποίηση μεγάλων γλωσσικών μοντέλων σε εργασίες που αφορούν την ανάπτυξη λογισμικού. Την αρχική πηγή ανάκτησης των δειγμάτων αποτελεί η πλατφόρμα GitHub μέσω του ανοιχτού API που παρέχει για την ανάκτηση git commit[1]. Δημιούργησαν δύο σύνολα δεδομένων τα οποία ονόμασαν COMMITPACK και HumanEvalPack.

3.1.1 COMMITPACK

Για την δημιουργία αυτού του συνόλου τα δείγματα κώδικα ανακτήθηκαν από ένα αρχείο του GitHub[2] το οποίο φτιάχτηκε για την καταγραφή ιστορικού των δημόσιων αποθετηρίων της

[1] μια καταγραφή ή ”στιγμιότυπο” της κατάστασης των αρχείων ενός αποθετηρίου (repository) σε μια συγκεκριμένη χρονική στιγμή

[2] https://www.gharchive.org

πλατφόρμας. Με σκοπό την ορθή δόμηση του συνόλου εφαρμόστηκαν κάποια φίλτρα όπως, η ανάκτηση στιγμιοτύπων που αφορούν αποκλειστικά ένα αρχείο στο αποθετήριο για την αποφυγή περαιτέρω πολυπλοκότητας. Μέσο του φιλτραρισμένου συνόλου επιχείρησαν για κάθε δείγμα να αποθηκεύσουν το στιγμιότυπο πριν και μετά την αλλαγή του προγραμματιστή. Συνδυάζοντας τα μεταδεδομένα που πρόσφερε το GitHub για κάθε στιγμιότυπο με το στιγμιότυπο του κώδικα πριν και μετά την αλλαγή κατάφεραν να κατασκευάσουν ένα σύνολο δεδομένων μεγέθους 4 TB το οποίο κάλυψε περίπου 350 γλώσσες προγραμματισμού, και εμπεριείχε τις εξής ιδιότητες για κάθε στιγμιότυπο:

- commit hash: Μία αλφαριθμητική τιμή που είναι μοναδική για κάθε στιγμιότυπο και δημιουργείται από το εργαλείο git
- Το όνομα του αρχείου πριν και μετά την αλλαγή (τις περισσότερες φορές είναι το ίδιο)
- Τα περιεχόμενα του αρχείου πριν και μετά την αλλαγή
- Το μήνυμα που έγραψε ο προγραμματιστής όταν αποθήκευσε την αλλαγή στο αποθετήριο
- Το όνομα του αποθετηρίου
- την γλώσσα προγραμματισμού

Από το σύνολο τον δειγμάτων αυτά που αφορούσαν την JavaScript ήταν λίγο λιγότερα από 100.000 δείγματα πάνω στα οποία εφαρμόστηκαν κάποια φίλτρα ακόμη ώστε να αγνοηθούν στιγμιότυπα που δεν θα είναι χρήσιμα για τροφοδότηση σε μεγάλα γλωσσικά μοντέλα. Όπως τα αρχικά στιγμιότυπα ενός αποθετηρίου, είτε στιγμιότυπα που το μήνυμα δεν περιέγραφε κάποια αλλαγή ή προσθήκη λειτουργίας, ή ακόμη στιγμιότυπα που το μήνυμα του προγραμματιστή ήταν πολύ μικρό ή εκτενές. Σε αυτό το σημείο είναι σημαντικό να παρατεθεί πως το μήνυμα του προγραμματιστή, όταν είναι αρκετά περιγραφικό, προσθέτει σημασιολογία στο μπλοκ κώδικα που αφορά, καθιστώντας το έτσι ιδανικό για την χρήση του σε γλωσσικά μοντέλα που πραγματοποιούν εργασίες που αφορούν την διερμήνευση ή παραγωγή κώδικα.
Για αυτούς του λόγους επιλέχθηκε το προ επεξεργασμένο σύνολο COMMITPACK για την εκπαίδευση των μοντέλων που χρησιμοποίησα. Ωστόσο, δεν ήταν δυνατή η χρήση όλων τον δειγμάτων JavaScript που περιείχε διότι δεν εφαρμόστηκε κάποιο φίλτρο που να εξακρίβωνε πως όλα τα δείγματα αφορούσαν αποσφαλμάτωση ή/και βελτίωση κώδικα. Κάτι που αναφέρεται και από τους συντάκτες, οι οποίοι επισημαίνουν πως το σύνολο περιείχε στιγμιότυπα όπου οι αλλαγές αφορούσαν διάφορα θέματα που ενδέχεται να προκύψουν στην ανάπτυξη λογισμικού, όπως η προσθήκη νέων λειτουργιών (το οποίο αποτελεί και το μεγαλύτερο μέρος των δειγμάτων), αλλαγές σε αρχεία ρυθμίσεων της εφαρμογής, προσθήκη/αφαίρεση βιβλιοθηκών και πολλά άλλα. Επομένως, κρίθηκε αναγκαίο το περαιτέρω φιλτράρισμα του συνόλου δειγμάτων JavaScript που ανέκτησα από το αρχικό σύνολο, με σκοπό την εξαγωγή δειγμάτων που αφορούν αποσφαλμάτωση ή/και βελτίωση κώδικα, για την πιο στοχευμένη εκπαίδευση των μοντέλων.

3.1.2 Εξαγωγή δειγμάτων που αφορούν αποσφαλμάτωση & Δημιουργία του συνόλου εκπαίδευσης και αξιολόγησης

Ο στόχος που τέθηκε κατά την δημιουργία της εφαρμογής ήταν αρχικά η ανίχνευση των υποψήφιων σφαλμάτων σε ένα δείγμα κώδικα και έπειτα η κατηγοριοποίηση του καθώς και η διόρθωση του. Από το αρχικό σύνολο των φιλτραρισμένων δειγμάτων του COMMITPACK, τα 52989 ήταν σε JavaScript. Όπως αναλύθηκε παραπάνω η δομή που κατασκευάστηκε η πηγή COMMITPACK δεν εμπεριείχε κάποια έτοιμη πληροφορία που να υποδηλώνει διόρθωση σφάλματος, ή τον τύπο του σφάλματος. Επομένως ήταν αναγκαία η περαιτέρω επεξεργασία του, ώστε να απομείνουν τα δείγματα που αποτελούν διόρθωση σφάλματος καθώς και να καταγραφεί ο τύπος σφάλματος. Ο πιο έγκυρος τρόπος για να πραγματοποιηθεί αυτό θα ήταν η σειρά προς σειρά ανάλυση του κάθε δείγματος και έπειτα απόδοση τύπου σφάλματος από ειδικούς, αλλά αυτό καθίσταται χρονικά αδύνατο σε ένα σύνολο 52989 δειγμάτων. Επομένως υλοποίησα μία διαδικασία που διακρίνεται σε δύο στάδια και χρησιμοποιεί για βάση της το μύνημα του προγραμματιστή. Αρχικά, δεδομένης μίας λίστας από λέξεις που υποδηλώνουν διόρθωση χωρίς να ελέγχεται ο τύπος σφάλματος (π.χ. *fix*, *patch*, *debug*, *repair* κ.α.) προσπελάσθηκε το σύνολο των 52989 δειγμάτων ώστε να αγνοηθούν τα δείγματα που δεν αφορούν διόρθωση, χωρίς όμως να κατηγοριοποιείται το δείγμα με κάποιο τύπο σφάλματος. Δεύτερον, δημιουργήθηκε ένα λεξιλόγιο με λέξεις και φράσεις κλειδιά, βάση του οποίου θα γίνει η κατηγοριοποίηση του τύπου σφάλματος των δειγμάτων προγραμματιστικά, ελέγχοντας δηλαδή την ύπαρξη ή όχι κάποιον λέξεων η φράσεων κλειδιών για κάθε τύπο σφάλματος στο μήνυμα του κάθε δείγματος (μπορεί να εμπεριέχονται φράσεις για παραπάνω από έναν τύπο σφάλματος). Για την ομοιογενή κατανομή των δειγμάτων σε κατηγορίες σφαλμάτων , η κατηγοριοποίηση των μηνυμάτων σε τύπους σφαλμάτων εφαρμόζεται επιλέγοντας ανάμεσα από πέντε κλάσεις:

- general: για γενικές διορθώσεις που αφορούν την λογική,
- functionality: για διορθώσεις που αφορούν την ορθή λειτουργία μιας εφαρμογής,
- compatibility/performance: για διορθώσεις που αφορούν την απόδοση και την συμβατότητα του κώδικα,
- network/security: για διορθώσεις που αφορούν την επικοινωνία με άλλες εφαρμογές ή/και την ασφάλεια των δεδομένων του χρήστη
- ui-ux: για διορθώσεις που αφορούν σφάλματα ή λάθος συμπεριφορές της εφαρμογής σχετικά με την εμπειρία ή/και την διεπαφή χρήστη.

Ο κάθε τύπος αντιστοιχίζεται με λέξεις κλειδιά (π.χ. λέξη κλειδί της ομάδας ”ui-ux” είναι η λέξη ”body”) , και κάθε δείγμα ενδέχεται να ανήκει σε παραπάνω από μία κλάσεις όταν εμπεριέχονται στο μήνυμα λέξεις κλειδιά που ανήκουν σε διαφορετικές κλάσεις. Παραδείγματος χάριν, έστω ότι ελέγχουμε το μύνημα ”*Fix config so that color legend shows up correctly*” για απόδοση τύπου σφάλματος.

1. Το μύνημα υποδεικνύει την ύπαρξη σφάλματος μέσω της λέξης ”fix”

2. Η ομάδα ”ui-ux” περιέχει στις λέξεις κλειδιά της την λέξη ”color” επομένως το δείγμα θα κατηγοριοποιηθεί ως σφάλμα διεπαφής χρήστη.

3. Η ομάδα ”compatibility/performance” περιέχει στις λέξεις κλειδιά της την λέξη ”config” επομένως το δείγμα θα κατηγοριοποιηθεί και ως σφάλμα απόδοσης και συμβατότητας.

Η κατηγορία ”general” δημιουργήθηκε για να καλύψει δείγματα που το μήνυμα τους υποδήλωνε την διόρθωση σφάλματος αλλά δεν ανατέθηκε σε κάποια άλλη κατηγορία. Η προσέγγιση αυτή εκτός από το γεγονός πως επιτάχυνε την κατηγοριοποίηση (ανάθεση τύπου σφάλματος) των δειγμάτων, αξιοποιεί και την σημασιολογία που υποδηλώνεται από το μήνυμα του προγραμματιστή για τις αλλαγές στον κώδικα ,όπου συνήθως είναι ο πιο αποδοτικός τρόπος εύρεσης ενός τύπου σφάλματος ειδικότερα σε μεγάλα αποθετήρια κώδικα. Ωστόσο, αυτή η προσέγγιση έχει και ορισμένα μειονεκτήματα. Η ακρίβεια της κατηγοριοποίησης μπορεί να είναι περιορισμένη λόγω της εξάρτησής της από την παρουσία συγκεκριμένων λέξεων ή φράσεων. Αυτό μπορεί να οδηγήσει σε μη ακριβή αποτελέσματα όταν τα μηνύματα είναι πιο σύνθετα ή χρησιμοποιούν τις λέξεις-κλειδιά με διαφορετικά συμφραζόμενα. Επιπλέον, οι προκαθορισμένες κατηγορίες και λέξεις-κλειδιά μπορεί να μην καλύπτουν όλες τις πιθανές περιπτώσεις σφαλμάτων ή τις διαφοροποιήσεις στα μηνύματα, με αποτέλεσμα να υπάρχει κίνδυνος εσφαλμένης κατηγοριοποίησης ή απώλειας λεπτομέρειας. Τέλος, η προσέγγιση αυτή μπορεί να μην είναι ευέλικτη απέναντι σε μηνύματα που χρησιμοποιούν ασυνήθιστες φράσεις ή διαφορετικές γλώσσες, περιορίζοντας έτσι την ικανότητά της να διαχειρίζεται ένα ευρύτερο φάσμα σεναρίων διόρθωσης σφαλμάτων.

Η διαδικασία αυτή κατάφερε να αποδώσει τύπο σφάλματος στα 29034 από τα 52989 που πρόσφερε η πηγή και έπειτα αυτά διαχωρίστηκαν σε σύνολα για εκπαίδευση και ανάκληση. Μετά την καταμέτρηση των δειγμάτων και τον διαχωρισμό τους σε σύνολα εκπαίδευσης και ανάκλησης με ποσοστά 80% και 20% αντίστοιχα, για κάθε τύπο σφάλματος στο σύνολο εκπαίδευσης, βρέθηκε ότι η κατηγορία ”general” περιλαμβάνει 2.862 δείγματα. Η κατηγορία ”functionality” είχε συνολικά 4.532 δείγματα, που είναι και η πιο πολυάριθμη κατηγορία, ενώ η κατηγορία ”compatibility/performance” περιλάμβανε 4.396 δείγματα. Στην κατηγορία ”network-security” βρέθηκαν 3.159 δείγματα, και τέλος, η κατηγορία ”ui-ux” περιλαμβάνει 3.147 δείγματα. Αυτά τα αποτελέσματα δείχνουν μια ποικιλία στον αριθμό των δειγμάτων ανά κατηγορία, με την ομάδα ”functionality” να έχει το μεγαλύτερο αριθμό και τη ”general” τον μικρότερο. Ανάλογα ήταν και τα αθροίσματα στο σύνολο αξιολόγησης όπου η κατηγορία ”general” είχε 723 δείγματα, ενώ η κατηγορία ”functionality” περιλάμβανε 1.118 δείγματα, με τον μεγαλύτερο αριθμό δειγμάτων, επιβεβαιώνοντας την τάση που παρατηρήθηκε και στο σύνολο εκπαίδευσης. Η κατηγορία ”compatibility-performance” με 1.063 δείγματα. Η κατηγορία ”network-security” είχε 761 δείγματα, ενώ η κατηγορία ”ui-ux” είχε 772 δείγματα. Η εμφάνιση περισσότερων δειγμάτων στην κατηγορία ”functionality” και στα δύο σύνολα, δύναται να αποδοθεί στο γεγονός πως τα λειτουργικά ζητήματα είναι συνήθη προβλήματα στην ανάπτυξη λογισμικού, ειδικότερα σε εφαρμογές που συνεχώς αλλάζουν ή προστίθενται λειτουργικές απαιτήσεις. Αντίστοιχα, ο τύπος σφάλματος με τα λιγότερα δείγματα είναι η ”general” και στα δύο σύνολο γεγονός που δικαιολογείται από την λογική της διαδικασίας της κατηγοριοποίησης. Ωστόσο, ομοίως ο τύπος ”network-security” έχει σχετικά μικρό άθροισμα δειγμάτων κάτι που αιτιολογείται από την φύση της γλώσσας, η οποία κατά κύριο λόγο χρησιμοποιείται για εφαρμογές πελάτη στις οποίες οι λειτουργίες που σχετίζονται με δίκτυα και ασφάλεια δεδομένων είναι μειωμένες, και

έχουν επιλυθεί καθολικά από βιβλιοθήκες ή πλαίσια εφαρμογής.

3.1.3 HumanEvalPack

Αφού οι συντάκτες επισημάνουν πως η χρήση κώδικα σε μεγάλα γλωσσικά μοντέλα συνήθως χρησιμοποιείτε σε εργασίες σύνθεσης ή περιγραφής κώδικα, πάνω στις οποίες έχουν αναπτυχθεί δείκτες αναφοράς και αξιολόγησης των γλωσσικών μοντέλων όπως είναι τα CodeXGlue [23] και HumanEval [24] τα οποία περιέχουν εξατομικευμένα σύνολα δεδομένων κώδικα και χρησιμοποιούνται για την αξιολόγηση της απόδοσης γλωσσικών μοντέλων στις παραπάνω εργασίες. Οι συντάκτες του HumanEvalPack [22] εργάστηκαν στην επέκταση του HumanEval με μια νέα δομή έτσι ώστε να μπορεί να χρησιμοποιηθεί και για την εργασία αποσφαλμάτωσης κώδικα. Το συγκεκριμένο σύνολο εφαρμόστηκε τόσο στην διαδικασία αξιολόγησης των αρχιτεκτονικών που υλοποίησα, καθώς και κάποια δείγματα του χρησιμοποιήθηκαν για την την ανάκληση των μοντέλων.

Κεφάλαιο 4

Επιλογή & Ανάκτηση Μοντέλων

Η ανάκτηση των προ εκπαιδευμένων μοντέλων έγινε ομοίως με το σύνολο δεδομένων από την πλατφόρμα huggingface [21]. Το huggingface είναι μία διακεκριμένη ανοιχτή πλατφόρμα που μεγιστοποίησε την διαδικασία ανάπτυξης μοντέλων μηχανικής μάθησης και εφαρμογών, παρέχοντας εργαλεία και πόρους σε ερευνητές και προγραμματιστές. Η κύρια προσφορά της είναι μία πληθώρα από προ-εκπαιδευμένα μοντέλα της αρχιτεκτονικής transformer (όπως τα T5, Bert) και όλες τις παραλλαγές τους στις οποίες εφαρμόστηκε *finetuning* για διάφορες λειτουργίες επεξεργασίας φυσικής γλώσσας. Επίσης, παρέχει και ένα μεγάλο αποθετήριο από σύνολα δεδομένων για όλες τις εργασίες φυσικής γλώσσας και άλλων τεχνικών μηχανικής μάθησης, συνδυάζοντας τα και με τη δυνατότητα της δημοσίευσης νέων συνόλων δεδομένων και μοντέλων από τους χρήστες της.
Εξαιρετικά σημαντική είναι και η βιβλιοθήκη σε Python που παρέχει για την ανάκτηση των μοντέλων και συνόλων δεδομένων που διαθέτει, η οποία ενσωματώνεται άριστα με τα γνωστά πλαίσια εφαρμογής για βαθιά μάθηση (όπως PyTorch, TensorFlow). Διευκολύνοντας την πρόσβαση σε εργαλεία και τεχνολογίες αιχμής το huggingface κατάφερε να διευρύνει το εύρος χρήσης των μεγάλων γλωσσικών μοντέλων μειώνοντας ταυτόχρονα τις απαραίτητες γνώσεις που χρειάζονται, ωθώντας την κοινότητα της μηχανικής μάθησης σε ραγδαία ανάπτυξη.

4.1 Επιλογή, ανάκτηση και παραμετροποίηση του Tokenizer

4.1.1 Ανασκόπηση των διεργασιών που εφαρμόζει ένας tokenizer

Στις εργασίες επεξεργασίας φυσικού λόγου είναι αναγκαία η διαδικασία του μετασχηματισμού του ακατέργαστου κειμένου σε μια αναπαράσταση που μπορεί να κατανοήσει ένα μοντέλο (όπως το Bert και το T5), δηλαδή διανυσματικά, και πραγματοποιείται με την χρήση ενός ”tokenizer”. Ο tokenizer είναι ένα εργαλείο που διασπά το κείμενο σε μικρότερα κομμάτια, που ονομάζονται tokens. Στην περίπτωση του Roberta Tokenizer, χρησιμοποιείται η τεχνική της κωδικοποίησης ζευγών bytes (Byte Pair Encoding - BPE), η οποία επιτρέπει την αποτελεσμα-

τική αναπαράσταση του κειμένου σε υποσύνολα χαρακτήρων και λέξεων. Η πρώτη διεργασία που ανατίθεται σε έναν tokenizer είναι ο διαχωρισμός του κειμένου σε ”tokens”. Όταν δίνεται ένα ακατέργαστο κείμενο, όπως μια πρόταση ή μια παράγραφος, ο tokenizer πρώτα αναλύει το κείμενο σε υπομονάδες, όπως λέξεις, σημεία στίξης και ειδικούς χαρακτήρες. Στην περίπτωση του BPE, ο tokenizer αρχικά διασπά το κείμενο σε μεμονωμένους χαρακτήρες ή μικρές ομάδες χαρακτήρων.
Στην συνέχεια ένας ”tokenizer” BPE χρησιμοποιεί τον αλγόριθμο του BPE για να συνδυάσει τις υπομονάδες σε μεγαλύτερα σύνολα, βασιζόμενος στην συχνότητα εμφάνισης των ζευγών αυτών στο κείμενο εκπαίδευσης. Για παράδειγμα, αν το ζεύγος των χαρακτήρων ”e” και ”r” εμφανίζεται συχνά μαζί, ο tokenizer θα τα συνδυάσει σε ένα νέο token ”er”. Αυτή η διαδικασία συνεχίζεται επαναληπτικά μέχρι να φτάσουμε σε ένα σύνολο tokens που αντιπροσωπεύει αποδοτικά το κείμενο. Αφού το κείμενο διασπαστεί σε tokens, κάθε token αντιστοιχίζεται σε ένα μοναδικό αριθμητικό αναγνωριστικό (ID). Αυτά τα token IDs δημιουργούνται κατά τη διάρκεια της εκπαίδευσης των βασικών μοντέλων (όπως το BERT ή το Roberta) σε μεγάλα σύνολα δεδομένων φυσικής γλώσσας. Αυτή η διαδικασία αναφέρεται συνήθως απο την κοινότητα ως finetuning. Το αποτέλεσμα είναι ένα λεξικό ή πίνακας αναζήτησης που αντιστοιχεί κάθε token σε ένα ID, το οποίο μπορεί να χρησιμοποιηθεί από το μοντέλο, για υλοποίηση συγκεκριμένων εργασιών.
Μετά την αντιστοίχιση των υπομονάδων του αρχικού ακατέργαστου κειμένου με token IDs, το επόμενο βήμα είναι η μετατροπή αυτών των IDs σε ενσωματώσεις (embeddings). Οι ενσωματώσεις είναι μεγάλα πολυδιάστατα διανύσματα που αντιπροσωπεύουν τα tokens σε έναν χώρο υψηλών διαστάσεων. Κατά την εκπαίδευση του μοντέλου, αυτά τα διανύσματα μαθαίνονται έτσι ώστε να αποτυπώνουν τις σημασιολογικές και συντακτικές σχέσεις μεταξύ των λέξεων. Κάθε διάσταση του διανύσματος ενσωμάτωσης συμβάλλει σε διάφορες πτυχές της σημασιολογίας ενός token, όπως το πλαίσιο χρήσης του σε μια πρόταση ή τις συσχετίσεις του με άλλα tokens σε μία φράση. Το μοντέλο χρησιμοποιεί τα embeddings για να επεξεργαστεί το κείμενο και να εκτελέσει διάφορες εργασίες, όπως κατανόηση φυσικής γλώσσας, παραγωγή κειμένου, μετάφραση κ.α.
Για καλύτερη κατανόηση θα παρατεθεί ένα παράδειγμα των διεργασιών που υλοποιεί ένας tokenizer, χρησιομοποιώντας ως δείγμα την πρόταση ”**Η γάτα τρέχει γρήγορα.**”. Στο πρώτο βήμα ο tokenizer θα επιχειρήσει να στοιχειοποιήσει το παραπάνω κείμενο με αποτέλεσμα μια λίστα με τα εξής στοιχεία, **(”Η”, ”γάτα”, ”τρέχει”, ”γρήγορα”, ”.”)**. Έπειτα θα προσπαθήσει να αντιστοιχήσει το καθένα από τα στοιχεία της λίστας με ένα αριθμητικό αναγνωριστικό από το λεξιλόγιο του (έστω ότι η αντιστοίχηση του απεικονίζεται στο σχήμα 4.1). Επομένως καταλήγουμε στο συμπέρασμα πως το διάνυσμα ενσωμάτωσης της φράσης ”Η γάτα τρέχει γρήγορα.” είναι **[321, 1245, 678, 543, 19]** το οποίο θα αποτελεί και την ακολουθία εισόδου για το μοντέλο ώστε να μπορέσει το μοντέλο να το επεξεργαστεί.

Τα embeddings επιτρέπουν στο μοντέλο να εργάζεται με μια συνεπή αναπαράσταση της σημασίας και της σχέσης μεταξύ των λέξεων, ακόμη και όταν αυτές εμφανίζονται σε διαφορετικά πλαίσια ή μορφές, στην περίπτωση μας να παράξει έγκυρο κώδικα JavaScript. Συνοπτικά, η ορθή διαδικασία της κωδικοποίησης ζευγών bytes και της δημιουργίας embeddings είναι ζωτικής σημασίας για την κατανόηση του κειμένου από τα μοντέλα μηχανικής μάθησης, καθώς επιτρέπει τη μετατροπή του ακατέργαστου κειμένου σε μορφή που το μοντέλο μπορεί να επε-

```
"Η" -> ID: 321,
"γάτα" -> ID: 1245,
"τρέχει" -> ID: 678,
"γρήγορα" -> ID: 543
"." -> ID: 19
```

Σχήμα 4.1: Αντιστοίχηση στοιχείων πρότασης με αριθμητικά αναγνωριστικά από το λεξιλόγιο του tokenizer

ξεργαστεί και να κατανοήσει.

4.1.2 Χρησιμοποίηση του Roberta Tokenizer

Όσον αφορά την υλοποίηση, ο tokenizer που χρησιμοποιήθηκε έγινε προσιτός από την πλατφόρμα huggingface [21] όπως ακριβώς και με την πηγή του dataset καθώς και τα μοντέλα που χρησιμοποιήθηκαν.Συγκεκριμένα, οι συντάκτες και από τα δύο μοντέλα [2], [3] πρότειναν την χρήση του Roberta Tokenizer ο οποίος χρησιμοποιεί την κωδικοποίηση ζευγών Byte. Ο συγκεκριμένος tokenizer είναι παράγωγος του tokenizer του πρώιμου μοντέλου GPT-2 χρησιμοποιώντας μεγάλους όγκους δεδομένων κειμένου της αγγλικής γλώσσας, καθώς και εκπαιδεύτηκε με τέτοιο τρόπο ώστε να χειρίζεται τα κενά σαν μέρος του token έτσι ώστε μία λέξη να κωδικοποιηθεί με διαφορετικό τρόπο σε σχέση με την θέση της σε μια πρόταση (αν βρίσκεται στην αρχή ή όχι). Ωστόσο, όταν θέλουμε να εφαρμόσουμε ”finetuning”[1] σε ένα προ-εκπαιδευμένο μεγάλο γλωσσικό μοντέλο για την υλοποίηση μιας εργασίας σε έναν εξειδικευμένο τομέα (downward domain specific task) είναι απολύτως σημαντική η σωστή διαμόρφωση των παραμέτρων που δέχεται ο tokenizer διότι στην ουσία αυτές οι παράμετροι θα καθορίσουν το πως θα διασπαστεί το κείμενο σε μικρότερα κομμάτια (tokens), τα οποία στη συνέχεια τροφοδοτούνται στο μοντέλο. Κάποιοι από αυτές τις παραμέτρους είναι :

- Το ”max length”,το οποίο καθορίζει το μέγιστο μήκος των ακολουθιών των embeddings που θα δημιουργηθούν από τον tokenizer. Για πολλές εργασίες NLP[2], το μήκος των προτάσεων ή των κειμένων ποικίλλει, και η παράμετρος αυτή βοηθά να εξασφαλίσουμε ότι όλες οι ακολουθίες έχουν το ίδιο μήκος, ώστε να μπορεί το μοντέλο να τις επεξεργαστεί. Στην δικιά μας περίπτωση που τα δείγματα του συνόλου εκπαίδευσης ήταν συνήθως μία συνάρτηση είτε συνδυασμός συναρτήσεων που εξάγει λειτουργικότητα το μέσο μέγεθος των ακολουθιών tokens, κυμαινόταν στα 250-500 στοιχεία αναλόγως τον τύπο σφάλματος, δεν είχε μεγάλη διασπορά και έπρεπε να διαμορφωθεί ανάλογα για κάθε μοντέλο, καθώς και έχοντας υπό όψιν την μέγιστη ακολουθία tokens που δέχονται τα μοντέλα που χρησιμοποιήθηκαν.
- Η παράμετρος ”padding” καθορίζει εάν οι ακολουθίες των tokens θα πρέπει να συμπλη-

[1] Η διαδικασία επανεκπαίδευσης ενός μοντέλου για την βελτιστοποίηση των υπερ-παραμέτρων του
[2] Επεξεργασία φυσικής γλώσσας

ρώνονται στο ίδιο μήκος. Αυτό είναι ιδιαίτερα χρήσιμο για στην εκπαίδευση με χρήση παρτίδων δειγμάτων (batch), όπου όλα τα δείγματα σε ένα batch πρέπει να έχουν το ίδιο μήκος. Αυτή η παράμετρος συνδυάζεται συνήθως με την παράμετρο ”max length” και ορίζεται ώστε όλες οι ακολουθίες να συμπληρώνονται με βάση την τιμή που max length που ορίστηκε.

- Η παράμετρος ”truncation” καθορίζει εάν οι ακολουθίες των tokens που υπερβαίνουν το ”max length” θα πρέπει να αποκοπούν.

Συνεπώς, η σωστή διαμόρφωση των παραμέτρων του Roberta Tokenizer είναι κρίσιμη για την επίτευξη βέλτιστων αποτελεσμάτων στην διαδικασία του finetuning. Η επιλογή των παραμέτρων ”max length”, ”padding”, και ”truncation” πρέπει να γίνεται με βάση τα χαρακτηριστικά των δεδομένων εκπαίδευσης και τις απαιτήσεις της εργασίας που θέλουμε να εκτελέσουμε. Μέσω αυτών των ρυθμίσεων, μπορούμε να εξασφαλίσουμε ότι το μοντέλο μας θα λάβει την καλύτερη δυνατή αναπαράσταση των εισόδων και θα μπορεί να αποδώσει με μέγιστη ακρίβεια και αποδοτικότητα.

4.2 CodeT5

Τα προ εκπαιδευμένα μοντέλα Transformer όπως το T5, GPT, BERT μπορουν να γενικεύσουν σε ικανοποιητικό βαθμό, καθιστώντας τα ιδανικά για εκτενέστερη υλοποίηση εξατομικευμένων εργασιών φυσικής γλώσσας. Συγκεκριμένα, η λογική του ”pre-train then fine-tune”, δηλαδή η επανεκπαίδευση μεγάλων γλωσσικών μοντέλων με τροποποιημένα dataset και μικρή τροποποίηση των βασικών αρχιτεκτονικών τους έδωσε την ευκαιρία να εφαρμοστούν αυτά τα γλωσσικά μοντέλα για εργασίες ειδικού τομέα (π.χ. παραγωγή/σύγκριση/αντικατάσταση κώδικα) ειδικότερα σε περιπτώσεις που η σήμανση των δεδομένων (data annotation) δεν είναι ιδανική. Οι συγγραφείς προτείνουν ένα μοντέλο Transformer με αρχιτεκτονική κωδικοποιητή - αποκωδικοποιητή το οποίο δύναται να παράγει κομμάτια κώδικα. Ισχυρίζτονται πως το μοντέλο έχει τη ικανότητα να εκμεταλλεύεται την σημασιολογία που βρίσκεται στην δομή σύνταξης ενός μπλοκ κώδικα, καθορίζοντας κάποια στοιχεία σε αυτό ως αναγνωριστικά. Επιπλέον, το μοντέλο είναι διμορφικό, δηλαδή λαμβάνει υπόψη και κέιμενο φυσικής γλώσσας, συνήθως υπό την μορφή σχολίων. Η βασική διαφορά του CodeT5 με άλλες παραλλαγές των πιο γνωστών μεγάλων γλωσσικών μοντέλων που εφαρμόζονται σε εργασίες γλωσσών προγραμματισμού όπως το GPT και το CodeBERT είναι δομική και αφορά την αρχιτεκτονική. Μοντέλα βασισμένα μόνο σε κωδικοποιητή (BERT) ή μόνο σε αποκωδικοποιητή (GPT) τροφοδωτούνται με διανύσματα που είναι απλοί μετασχηματιστμοί του αρχικού κειμένου , χωρίς να χρησιμοποιούν την επιπλέον πληροφορία που προσφέρει η δομή και ο τρόπος σύνταξης ενός μπλοκ κώδικα, ειδικότερα μετά τον μετασχηματισμό του σε Abstract Syntax Tree, ενώ το CodeT5 είναι από τις πρώτες υλοποιήσεις μεγάλων γλωσσικών μοντέλων με αρχιτεκτονική κωδικοποιητή - απόκωδικοποιητη.
Για το CodeT5 χρησιμοποιείται η βασική αρχιτεκτονική του μοντέλου T5 το οποίο εφαρμόζει αποθορυβοποίηση ακολουθίας σε ακολουθία (seq2seq) και έχει αποδειχθεί πως αποδίδει σε

καλό βαθμό σε εργασίες κατανόησης και παραγωγής φυσικής γλώσσας. Το επιπλέον στοιχείο που αναφέρουν οι συντάκτες αφορά τη χρήση "αναγνωριστικών" στοιχείων (identifiers) τα οποία αντιστιχίζονται με ένα μοναδικό στοιχείο του λεξιλογίου του tokenizer για τον διαχωρισμό τους από τα υπόλοιπα στοιχεία μιας ακολουθίας. Συγκεκριμένα, το μοντέλο εκπαιδεύεται με τέτοιον τρόπο ώστε τα στοιχεία που είναι "αναγνωριστικά" να αποκρύπτονται (masked tokens) και έπειτα το μοντέλο να προσπαθεί να τα ανακτήσει. Επίσης, είναι πολύ κρίσιμο το γεγονός πώς το CodeT5 έχει την δυνατότητα να επεξεργάζεται μπλοκ κώδικα που εμπεριέχουν φυσική γλώσσα (NL) (π.χ. σε μορφή σχολίων) καθώς υπάρχουν περιπτώσεις στην ανάπτυξη λογισμικού που ο προγραμματιστής εισάγει σχόλια με σκοπό να εξηγήσει την εκάστοτε λειτουργία που υλοποιεί ένα μπλοκ κώδικα.
Ουσιαστικά, το CodeT5 εκπαιδεύεται σε δύο εργασίες:

- την ταυτοποίηση αναγνωριστικών ετικετών (identifier tagging) και την απόκρυψη τους
- την πρόβλεψη των "κρυμμένων" στοιχείων.

Ένα στοιχείο identifier μπορεί να είναι το όνομα μιας συνάρτησης ή το όνομα μιας μεταβλητής, στοιχεία πολυ γενικά ενδέχεται να εμπεριέχουν πλούσια σημασιολογία. Όμοια με τις προηγούμενες αναφορές, κατά την τροφοδότηση του μοντέλου για κάθε δείγμα το κομμάτι που είναι PL[3] μετατρέπεται σε Abstract Syntax Tree, από το οποίο εξάγονται οι τύποι των κόμβων του δέντρου (π.χ. κόμβος κλήσης συνάρτησης κ.α.) και τελικά δημιουργείται μια ακολουθία μήκους m όπου m το σύνολο των λέξεων PL που παρήγαγε ο tokenizer, η οποία αναπαριστά την πιθανότητα το αντίστοιχο στοιχείο (word token) να είναι identifier.

Διαδικασία προ-εκπαίδευσης του CodeT5

Η διαδικασία της προ-εκπαίδευσης διαχωρίζεται από τους συγγραφείς σε τρεις εργασίες. Στην πρώτη χρησιμοποιούν τη λογική αποθορυβοποίησης (denoising) ακολουθίας σε ακολουθία, μια εδραιωμένη πρακτική για την υλοποίηση εργασιών επεξεργασίας φυσικής γλώσσας. Λειτουργεί αλλοιώνοντας την αρχική ακολουθία (θορυβοποίηση) στον κωδικοποιητή και αναθέτοντας στον αποκωδικοποιητή την επαναφορά του αρχικού κειμένου. Επιπλέον, εφαρμόζεται ένας μηχανισμός απόκρυψης μιας έκτασης τυχαίου μεγέθους στην ακολουθία την οποία αποσκοπεί να συμπληρώσει στο τέλος ο αποκωδικοποιητής. Αυτή η εργασία ονομάζεται Masked Span Prediction (MSP) με τύπο συνάρτησης απώλειας (Loss):

$$L_{MSP}(\theta) = \sum_{t=1}^{k} -logP(\boldsymbol{x}_t^{mask} | \boldsymbol{x}^{/mask}, \boldsymbol{x}_{<t}^{mask})$$

όπου οι παράμετροι του μοντέλου, $\boldsymbol{x}^{/mask}$ είναι η "κρυμμένη" είσοδος, $\boldsymbol{x}_{mask}$ είναι η "κρυμμένη" ακολουθία στόχος, με μήκος k, που οφείλει να προβλέψει ο αποκωδικοποιητής και $\boldsymbol{x}_{<t}^{mask}$ είναι η ακολουθία με μήκος το τυχαίο εύρος που καθορίστηκε στο MSP μια δεδομένη χρονική

[3]PL: κομμάτι που περιέχει κώδικα

στιγμή. Στο κομμάτι αυτό έγκειται η δυνατότητα τροφοδότησης αυτής της σημασιολογίας στο μοντέλο, θυμίζει τη λογική των λειτουργιών επισήμανσης κώδικα που έχουν οι εφαρμογές ανάπτυξης κώδικα όπως Visual Studio Code κ.α.
Όπως αναφέρθηκε παραπάνω, η ανάγκη ανάκτησης της πιθανής σημασιολογίας που υποδηλώνει η δομή και η σύνταξη αποσπασμάτων κώδικα, ειδικότερα μετά την κατασκευή του Abstract Syntax Tree, εφαρμόζεται στο μοντέλο από τους συγγραφείς διαχωρίζοντάς την σε δύο υπό εργασίες. Η μία αφορά το identifier tagging, δηλαδή την εύρεση των στοιχείων που είναι identifier, το οποίο υλοποιείται στον κωδικοποιητή, αντιστοιχίζοντας την τελευταία κρυφή κατάσταση κάθε ακολουθίας εισόδου με μια ακολουθία πιθανοτήτων $p = (p_1, ..., p_m)$ με συνάρτηση απώλειας τη δυαδική διασταυρούμενη εντροπία:

$$L_{IT}(\theta_e) = \sum_{i=1}^{m} -[\boldsymbol{y_i} log \boldsymbol{p_i} + (1 - \boldsymbol{y_i}) log(1 - \boldsymbol{p_i})]$$

Η απόκρυψη των επιλεγμένων στοιχείων identifier από το προηγούμενο βήμα γίνεται με την χρήση ειδικών στοιχείων που προστέθηκαν στο λεξιλόγιο του Roberta tokenizer. Η δεύτερη υπό εργασία αφορά την πρόβλεψη των στοιχείων identifier που αποκρύφτηκαν (Masked Identifier Prediction), όπου συνδυάζοντας την ακολουθία των ”κρυμμένων” στοιχείων identifier παράγεται μια ακολουθία στόχος πάνω στην οποία προβλέπεται η ακολουθία εξόδου με αυτό - παλινδρομικό τρόπο.
Οι συντάκτες επιχειρούν να εκπαιδεύσουν το μοντέλο ταυτόχρονα σε πολλές εργασίες με τέτοιον τρόπο που τα βάρη του επαναχρησιμοποιούνται για πολλές εργασίες με αποτέλεσμα να πετυχαίνει καλύτερη γενίκευση καθώς και να μειώνει σημαντικά το υπολογιστικό κόστος. Η στρατηγική αυτή στο στάδιο της προ-εκπαίδευσης καθιστά το μοντέλο ικανό να ρυθμιστεί λεπτομερώς για μεταγενέστερες εργασίες όπως η σύνοψη κώδικα, η παραγωγή και η μετάφραση κώδικα (δηλαδή η μεταφορά παλαιού λογισμικού) και η βελτίωση κώδικα.

4.3 CodeBert

Οι Feng, Guo και συν. [3] παρουσιάζουν ένα διμορφικό προ-εκπαιδευμένο μοντέλο βασισμένο στην αρχιτεκτονική του BERT. Εκπαιδεύεται με αναπαραστάσεις γενικού χαρακτήρα που ενδέχεται να είναι ζεύγη NL[4]-PL[5] είτε απλά δείγματα PL. Έπειτα από την προ εκπαίδευση του είναι δυνατό να του εφαρμοστεί finetuning για υλοποίηση εξατομικευμένων εργασιών όπως κατανόηση και παραγωγή κώδικα. Είναι προ εκπαιδευμένο στην εργασία της ανίχνευσης στοιχείων. Η δυνατότητα του μοντέλου να επεξεργάζεται διμορφικά δείγματα (NL-PL) είναι εφικτή με τη χρήση της τεχνικής masked language modeling (απόκρυψη ενός στοιχείου στην ακολουθία εισόδου με σκοπό την ανίχνευση του κατά την ακολουθία εξόδου) όπως γίνεται και στο CodeT5.
Το μοντέλο είναι στην ουσία η βασική έκδοση του μεγάλου γλωσσικού μοντέλου Roberta, εκπαιδευμένο σε ένα μεγάλο σύνολο δεδομένων υπό τη μορφή ζευγών NL-PL. Τα βασικά χαρακτηριστικά της αρχιτεκτονικής BERT είναι πως κατά την προ-εκπαίδευσή του υλοποιούνται

[4] φυσική γλώσσα
[5] γλώσσα προγραμματισμού

δύο εργασίες. Η μία είναι το masked language modeling, και η δεύτερη είναι η πρόβλεψη της επόμενης αποκρυμμένης ακολουθίας. Οι ακολουθίες εισόδου διακρίνονται σε δύο τμήματα και έχουν τη μορφή $[CLS], x_1, ..., x_N, [SEP], y_1, ..., y_M, [EOS]$, όπου CLS είναι ένα στοιχείο ειδικός δείκτης που συγκεντρώνει πληροφορίες τόσο από τη φυσική γλώσσα (σχόλια) όσο και από τα τμήματα κώδικα. Η προκύπτουσα κρυφή αναπαράσταση του μετά την επεξεργασία όλης της ακολουθίας από τον αποκωδικοποιητή BERT πιθανότατα αποτυπώνει το συνολικό νόημα της συνδυασμένης ακολουθίας. Το στοιχείο SEP διαχωρίζει τα δύο τμήματα της ακολουθίας και το EOS σηματοδοτεί το τέλος της ακολουθίας. Η συγκεκριμένη δομή είναι απόλυτα συμβατή για την εφαρμογή του σε εργασίες κατανόησης και παραγωγής κώδικα όπου τα δείγματα μπορεί να είναι ζεύγη NL-PL. Είναι σημαντικό να σημειωθεί επίσης ότι όλες οι εκδόσεις του μοντέλου BERT αποδίδουν σε καλύτερο βαθμό όταν εκπαιδεύονται με σύνολα πολύ μεγάλου μεγέθους και χωρίς να μειώνονται τα μήκη των ακολουθιών εισόδου. Όπως και στο μοντέλο CodeT5 για το tokenization των δειγμάτων εφαρμόζεται η τεχνική κωδικοποίησης ζευγών byte, μέσω του RobertaTokenizer.
Το CodeBert έχει την δυνατότητα να επεξεργάζεται διμορφικές εισόδους εφαρμόζοντας τον μηχανισμό απόκρυψης ταυτόχρονα και στα δύο τμήματα της εισόδου και χρησιμοποιώντας ένα διαφορετικό στρώμα για κάθε τμήμα διερμηνεύει ποιες είναι οι θέσεις των κρυμμένων στοιχείων, και συνδυάζοντας τα έπειτα επιχειρεί να προβλέψει τα στοιχεία που αποκρύφτηκαν και στα δύο τμήματα. Αυτή η τακτική προ εκπαίδευσης συνίσταται ιδανική για την εφαρμογή του μοντέλου σε finetuning με σκοπό εργασίες παραγωγής κώδικα δεδομένης της περιγραφής του ή το αντίστροφο, όπως επιχειρήθηκε και από τους συντάκτες του [1], καθώς έχει την δυνατότητα να συνδυάσει σημασιολογία και συμφραζόμενα που ανακτά ξεχωριστά από τα δύο τμήματα των διμορφικών εισόδων. Μια παράγραφο για το maske language modeling.

Διαδικασία προ-εκπαίδευσης του CodeBert

Το CodeBert εκπαιδεύεται σε δύο εργασίες:

- *Masked Language Modeling*: Δεδομένης μίας ακολουθίας εισόδου που διακρίνεται σε δύο τμήματα, ένα τμήμα NL και ένα τμήμα PL (σε αυτή την εργασία εφαρμόστηκαν αποκλειστικά δείγματα που είναι NL-PL), οι θέσεις μάσκας επιλέγονται τυχαία και στη συνέχεια αντικαθίστανται με το σύμβολο $[MASK]$ και στα δύο τμήματα της ακολουθίας ($\boldsymbol{w}_{masked}$: το τμήμα NL και $\boldsymbol{c}_{masked}$ το τμήμα PL). Ο στόχος του MLM είναι να προβλέψει τα αρχικά tokens που αποκρύφτηκαν με συνάρτηση απώλειας:

$$L_{MLM}(\theta) = \sum_{i \in \boldsymbol{m}^w \cup \boldsymbol{m}^c} -logp^{D_1}(x_i|\boldsymbol{w}^{masked}, \boldsymbol{c}^{masked})$$

  όπου p^{D_1} είναι μία συνάρτηση "διευκρινιστής" που προβλέψει τα στοιχεία που ταιριάζουν σε κάθε αποκρυμμένο στοιχείο της ακολουθίας.

- *Replaced Token Detection*: Ο στόχος σε αυτή την εργασία είναι η ανίχνευση του πραγματικού στοιχείου που αποκρύφτηκε μέσω των πιθανών στοιχείων που παρήγαγε το

προηγούμενο βήμα. Συγκεκριμένα, εδώ χρησιμοποιούνται δύο γεννήτριες p^{G_w}, p^{G_c}, μία για κάθε τμήμα της ακολουθίας εισόδου, οι οποίες παράγουν πιθανά στοιχεία για κάθε αποκρυμμένο στοιχείο του τμηματός τους. Ομοίως εφαρμόζεται μία συνάρτηση "διευκρινιστής" p^{D_2} η οποία υπολογίζει την πιθανότητα ένα αποκρυμμένο στοιχείο να είναι το πραγματικό για κάθε στοιχείο από την ακολουθία που παρήγαγε το MLM. Στην συνάρτηση απώλειας του RTD χρησιμοποιείται μία δυαδική συνάρτηση (i) με τιμές $[0, 1]$ αναλόγως αν το i-οστό είναι αποκρυμμένο, και έχει τύπο:

$$L_{RTD}(\theta) = \sum_{i=1}^{|\boldsymbol{w}|+|\boldsymbol{c}|} ((i)log\ p^{D_2}(\boldsymbol{x}^{corrupt}, i) + (1-(i))(1-log\ p^{D_2}(\boldsymbol{x}^{corrupt}, i)))$$

Παρόλο που το CodeBert αρχικά δεν αναπτύχθηκε για την διόρθωση σφαλμάτων, αλλά κυρίως για την διερμήνευση κώδικα, η ικανότητα του να επεξεργάζεται διμορφικά δείγματα (NL-PL) αφήνει ανοιχτό το πλαίσιο επέκτασης του για περισσότερες εργασίες επεξεργασίας κώδικα.Τα παραπάνω, συνδυαστικά με την δυνατότητά του να προβλέπει τα κρυμμένα στοιχεία σε ένα μπλοκ κώδικα μπορεί να εφαρμοστεί σε εργασίες διόρθωσης κώδικα με κάποιες παραμετροποιήσεις, παραδείγματος χάριν μία πιο στοχευμένη επιλογή των στοιχείων που θα αποκρυφτούν στην ακολουθία εισόδου. Κάτι παρόμοιο επιχειρήθηκε στην υλοποίηση, όπου μέσω της βιβλιοθήκης *difflib* [25], μία βιβλιοθήκη της Python που υπολογίζει τις διαφορές μεταξύ δύο συμβολοσειρών σε διάφορες μορφές, εφαρμόστηκε ο μηχανισμός απόκρυψης στα στοιχεία του μπλοκ κώδικα που άλλαξαν, με στόχο το CodeBert να εκπαιδευτεί στην πρόβλεψη των στοιχείων που άλλαξαν, δηλαδή την διόρθωση του σφάλματος. Η διαδικασία της στοχευμένης απόκρυψης θα αναλυθεί εκτενέστερα στο κεφάλαιο 5

4.4 Ταξινομητής Σφαλμάτων

Όπως διαπιστώθηκε και παραπάνω, στην υλοποίηση εκτός από την χρήση των CodeBert, CodeT5 δημιουργήθηκε και ένα απλό νευρωνικό δίκτυο ταξινόμησης πολλών κλάσεων με στόχο την ανίχνευση του τύπου σφάλματος για κάθε δείγμα. Συγκεκριμένα, το δίκτυο ταξινόμησης τροφοδοτείται με την κρυφή κατάσταση του τελευταίου στρώματος του κωδικοποιητή ή αποκωδικοποιητή και αποτελείται κατά σειρά από:

1. Ένα στρώμα υποδειγματοληψίας (Pooling), το οποίο υπολογίζει τον μέσο όρο των κρυφών καταστάσεων κατά μήκος της διάστασης του μήκους της ακολουθίας, ανακτώντας σημασιολογία για κάθε δείγμα από ένα batch σε ένα διάνυσμα.

2. Ένα στρώμα κανονικοποίησης

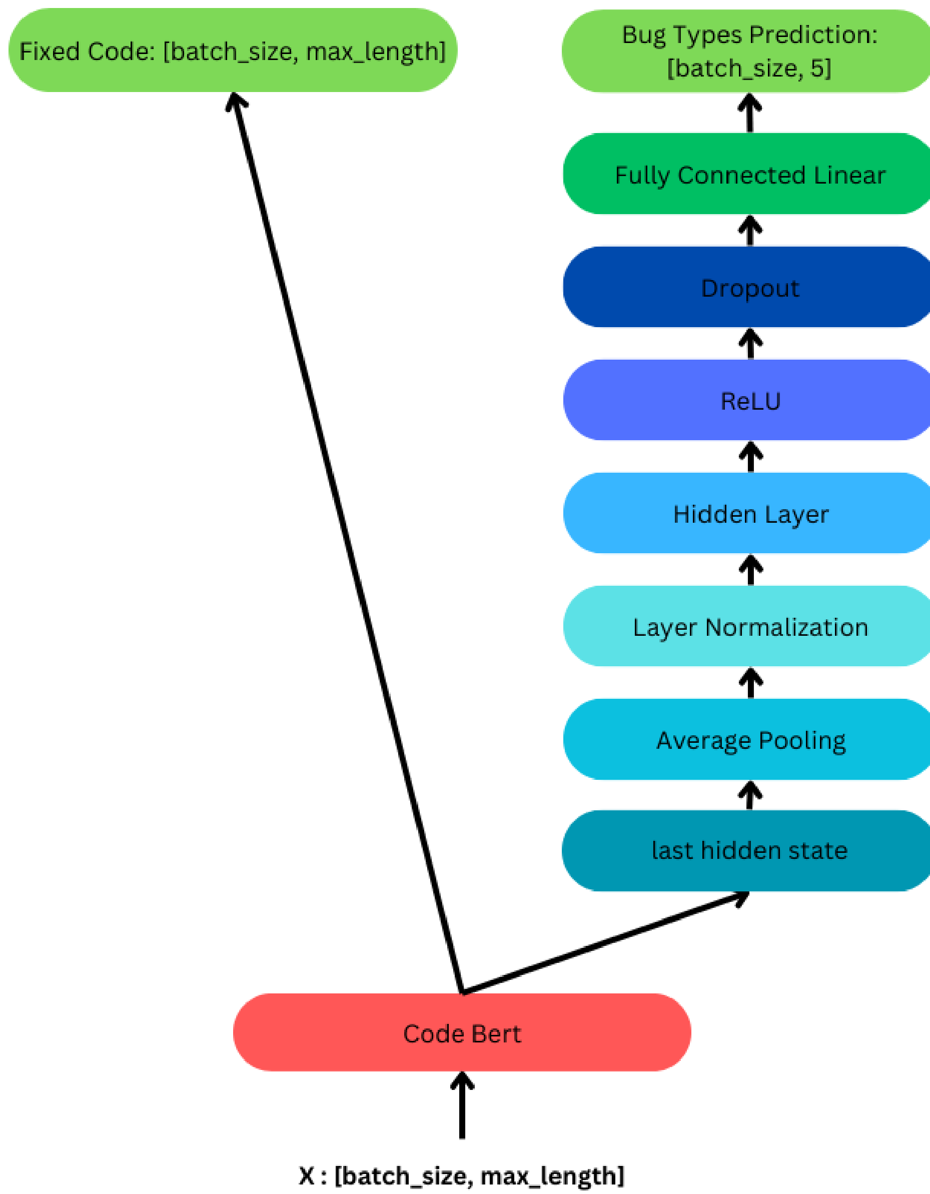
3. Ένα κρυφό βαθύ στρώμα το οποίο αναπαριστά την έξοδο του στρώματος κανονικοποίησης με διαφορετική διάσταση (συγκεκριμένα η διάσταση που επιλέχθηκε ήταν η μισή της μέγιστης διάστασης που μπορούν να επεξεργαστούν τα CodeBert, CodeT5 ανάλογα.

4. Ένα στρώμα της συνάρτησης ενεργοποίησης και συγκεκριμένα της συνάρτησης ReLU το οποίο διασφαλίζει την μετατροπή των αρνητικών τιμών σε μηδενικές το οποίο ενισχύει την δυνατότητα του μοντέλου να μαθαίνει πολύπλοκες αντιστοιχίσεις μεταξύ ακολουθίας εισόδου και τύπου σφάλματος.

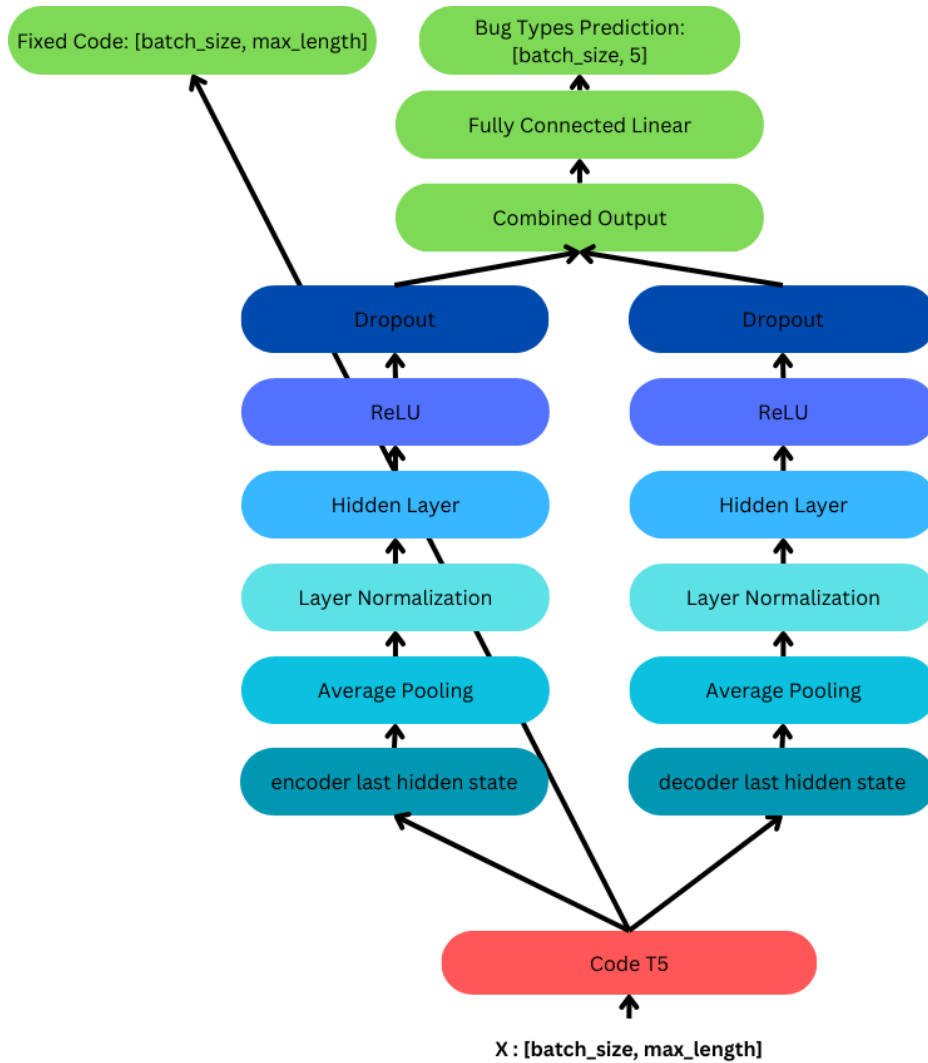
5. Ένα στρώμα τυχαίας αποκοπής (Dropout) το οποίο εφαρμόζεται για την καταπολέμηση του προβλήματος του overfitting μειώνοντας την εξάρτηση του ταξινομητή, από συγκεκριμένους νευρώνες

6. Ένα πλήρως συνδεδεμένο στρώμα το οποίο παράγει τις εξόδους, δηλαδή ένα διάνυσμα πιθανοτήτων που η κάθε θέση δηλώνει την πιθανότητα το i-οστό δείγμα να ανήκει στον τύπο σφάλματος που αντιστοιχεί στην συγκεκριμένη θέση.

Όπως καθορίζεται και από το σχήμα 4.2, δεδομένου ότι το CodeBert αποτελεί αρχιτεκτονική αποκωδικοποιητή, το δίκτυο ταξινόμησης εφαρμόστηκε αποκλειστικά στην τελευταία κρυφή κατάσταση του αποκωδικοποιητή του. Στην περίπτωση του CodeT5 (σχήμα 4.3) το οποίο εμπεριέχει κωδικοποιητή και αποκωδικοποιητή το δίκτυο ταξινόμησης εφαρμόστηκε και στις δύο κρυφές καταστάσεις των τελευταίων στρωμάτων του κωδικοποιητή και αποκωδικοποιητή ανάλογα, και στο πλήρως συνδεδεμένο στρώμα τροφοδοτήθηκε ο συνδυασμός των εξόδων που παρήγαγαν τα προηγούμενα στρώματα για τις τελευταίες κρυφές καταστάσεις κωδικοποιητή και αποκωδικοποιητή.



Σχήμα 4.2: Η αρχιτεκτονική του CodeBert με το δίκτυο ταξινόμησης πολλαπλών κλάσεων.



Σχήμα 4.3: Η αρχιτεκτονική του CodeT5 με το δίκτυο ταξινόμησης πολλαπλών κλάσεων.

Η συνάρτηση απώλειας που επιλέχθηκε για την ενημέρωση των βαρών στο δίκτυο ταξινόμησης είναι η δυαδική διασταυρούμενη εντροπία, καθώς αποτελεί την πιο διαδεδομένη συνάρτηση για εκπαίδευση νευρωνικών δικτύων ταξινόμησης. Στο πλαίσιο της ανίχνευσης τύπων σφαλμάτων, η δυαδική διασταυρούμενη εντροπία υπολογίζεται ως εξής:

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log \sigma(z_i) + (1 - y_i) \cdot (1 - (z_i))] \cdot w_i$$

, όπου N είναι ο συνολικός αριθμός του batch, z_i είναι το διάνυσμα εξόδου, δηλαδή οι τύποι σφάλματος που πρόβλεψε το δίκτυο ταξινόμησης για το δείγμα i, y_i είναι το διάνυσμα στόχος, δηλαδή το πραγματικό διάνυσμα των τύπων σφαλμάτων (ένα δείγμα μπορεί κατηγοριοποιείται με πάνω από έναν τύπο σφάλματος, επομένως το διάνυσμα στόχος παίρνει τιμή 1 στους δείκτες που αντιστοιχούν στην κλάση του δείγματος, διαφορετικά 0) , $\sigma(z_i) = \frac{1}{1+e^{-z_i}}$ είναι η τιμή της συνάρτησης ενεργοποίησης *sigmoid* και w_i είναι το βάρος του διανύσματος στόχου.

Κεφάλαιο 5

Μοντελοποίηση

Στο τρέχον κεφάλαιο θα αναλυθούν η διαδικασία προ επεξεργασίας των δειγμάτων σε μορφή συμβατή ώστε να τροφοδοτηθούν στα μοντέλα, ο τρόπος με τον οποίο τα μοντέλα μαθαίνουν καθώς θα αποδοθεί μία σύντομη επεξήγηση για το περιβάλλον και τις παραμέτρους εκπαίδευσης.

5.1 Προ-επεξεργασία των δεδομένων

Όπως αναλύθηκε στην ενότητα 4.2 το CodeT5 διαθέτει εξ' αρχής έναν μηχανισμό απόκρυψης στοιχείων που εφαρμόζεται στην ακολουθία εισόδου, αντίστοιχα οι συντάκτες του CodeBert επισημαίνουν πως εφαρμόζεται μηχανισμός απόκρυψης, με τυχαία επιλογή των στοιχείων της ακολουθίας εισόδου με την διαφορά ότι εφαρμόζεται προγραμματιστικά χωρίς να εμπλέκεται κάποιο εσωτερικό στρώμα του CodeBert. Αποσκοπώντας σε μία πιο ουσιώδης απόκρυψη στοιχείων στις ακολουθίες εισόδου ώστε το CodeBert να μετεκπαιδευτεί στην πρόβλεψη στοχευμένων στοιχείων σε μια ακολουθία, δημιουργήθηκε ένας μηχανισμός απόκρυψης που συγκρίνει σε ένα δείγμα τα δύο μπλοκ κώδικα, πριν και μετά τις διορθώσεις του προγραμματιστή και αποκρύπτει τα στοιχεία (στον κώδικα με σφάλματα) που άλλαξαν, δηλαδή τις διορθώσεις.
Συγκεκριμένα, συντάχθηκε μία συνάρτηση που χρησιμοποιεί τον tokenizer του CodeBert, για να στοιχειοποιήσει το μπλοκ κωδικά σε μια λίστα από word tokens πάνω στις οποίες εφαρμόζεται σύγκριση χαρακτήρα προς χαρακτήρα μεταξύ του κώδικα με σφάλματα και του διορθωμένου κώδικα, μέσω της βιβλιοθήκης *difflib*[25], ώστε να βρεθούν τα σημεία στο μπλοκ κώδικα που άλλαξαν, ώστε να αντικατασταθούν με το στοιχείο απόκρυψης και το CodeBert να εκπαιδευτεί στην πρόβλεψη τους. Για την αποφυγή δημιουργίας δειγμάτων που το στοιχείο απόκρυψης εμφανίζεται πολλές φορές (δηλαδή τα δείγματα που περιείχαν πολλές αλλαγές) η μηχανισμός εφαρμόστηκε μεταξύ δύο σεναρίων. Στις περιπτώσεις που το σύνολο των word tokens που άλλαξαν δεν είναι μεγαλύτερο από το $1/4$ του σύνολου των word tokens τότε εφαρμόστηκε το στοιχείο απόκρυψης του tokenizer στα word tokens που άλλαξαν. Διαφορετικά, εφαρμόστηκε το στοιχείο απόκρυψης σε ορισμένα word tokens με τυχαίο τρόπο. Τέλος η συνάρτηση επιστρέφει το μπλοκ κώδικα, αφού μετατράπηκε πάλι σε συμβολοσειρά. Η διαφοροποίηση με βάση

των αριθμό τον αλλαγών εφαρμόστηκε για την αποφυγή δημιουργίας θορύβου στο σύνολο εκπαίδευσης, παραδείγματος χάριν σε ένα δείγμα που υπήρχαν πολλές γραμμές με σχόλια και αφαιρέθηκαν, θα δημιουργούνταν ένα δείγμα που σε ένα μεγάλο κομμάτι του επαναλαμβανόταν το στοιχείο απόκρυψης.
Για την αποφυγή τροφοδότησης του μοντέλου με δείγματα που δεν θα βελτίωναν την απόδοση του εφαρμόστηκαν κάποια επιπλέον φίλτρα κατά σειρά:

1. Αγνοήθηκαν τα δείγματα που αφορούσαν σύνθεση νέου αρχείου ή συνάρτησης (δηλαδή αυτά που δεν υπήρχε κώδικας πριν τις αλλαγές)

2. Αγνοήθηκαν τα δείγματα που οι αλλαγές αφορούσαν προσθήκη σχολίων, καθώς και δείγματα που το μπλοκ περιείχε πάνω από 10 γραμμές με σχόλια

3. Αγνοήθηκαν τα δείγματα που ο προγραμματιστής διόρθωσε πάνω από τρεις περιπτώσεις σφαλμάτων ή το μπλοκ κώδικα ήταν πολύ μεγάλο (πάνω από 50 σειρές), διότι παρατηρήθηκε πως αναθέτοντας στα μοντέλα να επιχειρήσουν πολλές αλλαγές μέσα στο ίδιο δείγμα δεν βελτίωνε την απόδοση τους

5.2 Παραγωγή Διορθωμένου Κώδικα

Τα προγράμματα εκπαίδευσης (ένα για κάθε μοντέλο) περιλαμβάνουν τα ανακτημένα μοντέλα τα οποία έπειτα υλοποιήθηκαν ως υπό κλάσεις της *torch.nn.Module* η οποία προσφέρει κάποιες λειτουργικότητες σε μορφή συναρτήσεων για την υλοποίηση εκπαίδευσης νευρωνικών μοντέλων. Κάθε υπό κλάση αποτελούταν από το αντίστοιχο transformer μοντέλο, τον ταξινομητή και κάποιες ακόμη μεταβλητές που χρειάζονται για την εκπαίδευση (ο tokenizer, και οι αρχικές τιμές παραμέτρων των μοντέλων όπως ο ρυθμός εκπαίδευσης κ.α.). Όπως αναφέρθηκε και πριν, η διαδικασία της εκπαίδευσης διακρίνεται σε δύο εργασίες.

Bug Fixing

Η διανυσματική αναπαράσταση του κώδικα με σφάλματα που παρήγαγε ο tokenizer τροφοδοτείται στο μοντέλο transformer το οποίο παράγει. Στην περίπτωση του CodeT5, παράγεται μία ακολουθία μήκους [batch size, max length, vocabulary size], η οποία περιέχει τις βαθμολογίες για κάθε στοιχείο στο λεξιλόγιο του tokenizer σε κάθε θέση στη σειρά. Οι τιμές του διανύσματος συνήθως αναφέρονται ως "logits" και αντιπροσωπεύουν τις προβλέψεις του μοντέλου για το πώς θα πρέπει να διορθωθεί ο ελαττωματικός κώδικας. Μπορούν να μετατραπούν σε πιθανότητες μέσω της συνάρτησης softmax και να χρησιμοποιηθούν για τη δημιουργία της ακολουθίας του διορθωμένου κώδικα. Στην περίπτωση του CodeBert τροφοδοτείται στο δίκτυο η διανυσματική αναπαράσταση του ελαττωματικού κώδικα με αποκρυμμένα τα στοιχεία όπου συμβαίνει το σφάλμα και το μοντέλο εκπαιδεύεται στην συμπλήρωση των αποκρυμμένων στοιχείων χρησιμοποιώντας την διανυσματική αναπαράσταση του διορθωμένου κώδικα ως ακολουθία

στόχο. Η έξοδοι του έχουν ίδιο μέγεθος με αυτές του CodeT5 ωστόσο δηλώνουν την πρόβλεψη του μοντέλου για το ποια στοιχεία του λεξιλογίου του tokenizer αντιστοιχούν στα αποκρυμμένα στοιχεία της ακολουθίας.

Bug Type Detection

Στην δεύτερη εργασία χρησιμοποιούνται οι κρυφές καταστάσεις των τελευταίων στρωμάτων των CodeT5 και CodeBert όπου και τροφοδοτούνται στο δίκτυο ταξινόμησης για την πρόβλεψη του τύπου σφάλματος. Ο όρος ”τελευταία κρυφή κατάσταση” σε ένα δίκτυο transformer περιγράφει την έξοδο του τελευταίου στρώματος ενός δικτύου η οποία περιέχει εννοιολογική αναπαράσταση για κάθε στοιχείο της ακολουθίας εισόδου, αφού έχει επεξεργαστεί από τα στρώματα απόδοσης προσοχής και τα στρώματα τροφοδότησης προς τα εμπρός. Η χρήση της αποτελεί αρκετά κοινό φαινόμενο σε εργασίες finetuning μοντέλων transformer καθώς εσωτερικά της εμπεριέχει πληροφορία για το κάθε στοιχείο και τις σχέσεις που κατέχει με τα υπόλοιπα στοιχεία σε μία ακολουθία. Όπως αναλύθηκε και στην ενότητα 4.4, η ίδια στρατηγική ακολουθήθηκε όπου η τελευταία κρυφή κατάσταση του CodeBert και ο συνδυασμός των δύο τελευταίων κρυφών καταστάσεων του CodeT5 εφαρμόστηκαν στο δίκτυο ταξινόμησης για την ανίχνευση των τύπων σφάλματος σε ένα δείγμα.

Διαδικασία Εκπαίδευσης

Η εκπαίδευση των CodeBert, CodeT5 λειτουργεί, αρχικά δημιουργώντας ένα αντικείμενο tokenizer και τροφοδοτώντας του τις προ επεξεργασμένες ακολουθίες από το σύνολο εκπαίδευσης ώστε να τις μετατρέψει σε διανυσματικές αναπαραστάσεις, με βάση το λεξιλόγιο του. Αυτή η διαδικασία εφαρμόστηκε δύο φορές για κάθε μοντέλο, μία για τις ακολουθίες εισόδου που θα τροφοδοτηθούν στο μοντέλο, και μία για τις ακολουθίες του διορθωμένου κώδικα όπου θα χρησιμοποιηθούν για τον υπολογισμό της συνάρτησης απώλειας, τις ακολουθίες στόχου. Μία ακόμη σημαντική παράμετρος η οποία καθορίζει την επίδοση του transformer καθώς και του ταξινομητή είναι το μέγιστο πλήθος ακολουθιών (max length), όπως αναλύθηκε και στην ενότητα 4.1.2. Είναι σημαντικό, για την επιλογή της τιμής του να εξεταστούν τα μήκη των ακολουθιών στο σύνολο εκπαίδευσης, ώστε η τιμή του *max length* να εκφράζει την κατανομή τους. Στην συνέχεια το σύνολο εκπαίδευσης διαιρείται σε υποσύνολα (batches) 64 δειγμάτων τα οποία τροφοδοτούνται στα CodeBert, CodeT5. Σε αυτό το σημείο θεωρείται αρκετά χρήσιμη η δυνατότητα που προσφέρει η βιβλιοθήκη της πλατφόρμας Hugging Face η οποία επιστρέφει τις προβλέψεις των μοντέλων σαν πολυδιάστατα διανύσματα καθώς και την τιμή της συνάρτησης απώλειας. Στο σημείο αυτό ανακτούνται οι κρυφές καταστάσεις των τελευταίων στρωμάτων των μοντέλων όπου και τροφοδοτούνται στο δίκτυο ταξινόμησης. Συμπερασματικά, αφού τα δείγματα επεξεργαστούν και από το δίκτυο ταξινόμησης, είναι διαθέσιμες δύο τιμές απώλειας αυτή του αντίστοιχου μοντέλου L_{NL} και αυτή του δικτύου ταξινόμησης L_{BCE} . Αναλυτικότερα, όσον αφορά το CodeT5 η απώλεια υπολογίζεται με βάση την συνάρτηση αρνητικής λογαριθμικής πιθανοφάνειας (negative log-likelihood loss). Η τιμή της συνάρτησης απώλειας αρνητικής

λογαριθμικής πιθανοφάνειας δεν είναι πιθανότητα αλλά θετικός πραγματικός αριθμός που αναπαριστά την μέση αρνητική λογαριθμική πιθανότητα για όλα τα στοιχεία της ακολουθίας στόχου σε ένα υποσύνολο (batch), για αυτόν το λόγο είναι πολύ σημαντική η τιμή *max length* που θα αποδοθεί στο tokenizer καθώς μία πολύ μεγάλη τιμή μπορεί να επηρεάσει αρνητικά την τιμή της συνάρτησης λόγω του μεγαλύτερου συνόλου πιθανοτήτων. Παρόλο που έχει αρνητικό πρόσημο η τελική της τιμή είναι πάντα θετική καθώς οι λογαριθμικές πιθανότητες έχουν πάντα αρνητικό πρόσημο, και διατυπώνεται ως εξής:

$$L_{NL} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\log P(y_t|y_{>t}, x)$$

, όπου T είναι το μήκος της ακολουθίας στόχου, y_t είναι το στοιχείο της ακολουθίας στόχου την χρονική στιγμή t, $y_{>t}$ τα προηγούμενα στοιχεία της ακολουθίας στόχου ώστε να εφαρμοστεί αυτό παλινδρομική αποκωδικοποίηση ,x η ακολουθία εισόδου και $P(y_t|y_{>t}, x)$ είναι η πιθανότητα του στοιχείου y_t στην ακολουθία στόχου δεδομένων των προηγούμενων στοιχείων της ακολουθίας στόχου και της ακολουθίας εισόδου. Η συνάρτηση απώλειας αρνητικής λογαριθμικής πιθανοφάνειας εκφράζει την απόκλιση του κειμένου που παρήγαγε το CodeT5 από την πραγματικό κείμενο, και ο στόχος είναι η ελαχιστοποίηση της. Όσον αφορά το CodeBert, το οποίο αποτελεί αρχιτεκτονική transformer μόνο με κωδικοποιητή, εκπαιδεύεται στην πρόβλεψη των αποκρυμμένων στοιχείων. Ουσιαστικά για κάθε στοιχείο της ακολουθίας εισόδου το CodeBert παράγει ένα διάνυσμα μήκους ίσου με το λεξιλόγιο του tokenizer το οποίο περιέχει την κατανομή πιθανότητας το στοιχείο της ακολουθίας εισόδου (x_i) να είναι το τρέχον στοιχείο του λεξιλογίου με συνάρτηση απώλειας ομοίως την αρνητική λογαριθμική πιθανοφάνεια.

5.3 Συνδυασμός Απώλειας Transformer και Ταξινομητή

Στα μοντέλα μηχανικής μάθησης που υλοποιούν πάνω από μία εργασίες οι τιμές απώλειας κάθε εργασίας ενδέχεται να εκφράζονται σε διαφορετικές κλίμακες, όπως στην περίπτωση της αποσφαλμάτωσης από το transformer που η τιμή απώλειας εκφράζει την απόκλιση της παραγόμενης ακολουθίας από την ακολουθία στόχου και στην περίπτωση του ταξινομητή, όπου το μοντέλο προβλέπει τους τύπους σφάλματος στον κώδικα που διόρθωσε το transformer και η τιμή απώλειάς του είναι πιθανότητα. Επομένως εφαρμόστηκε συνάρτηση απώλειας που συνδυάζει τις απώλειες από τις δύο εργασίες χρησιμοποιώντας μια προσέγγιση βασισμένη σε βάρη, που μπορούν είτε να έχουν στατικές τιμές, είτε να υπολογίζονται δυναμικά, οδηγώντας το μοντέλο σε ποιοτικότερη γενίκευση. Επιπλέον η συνδυαστική απώλεια με δυναμικά βάρη επιφέρει σταθερότητα στις περιπτώσεις που η μία εργασία συγκλίνει ταχύτερα από την άλλη καθυστερώντας έτσι την εκπαίδευση του μοντέλου. Με αυτό τον τρόπο κανονικοποιείτε η κατανομή των απωλειών για κάθε εργασία διασφαλίζοντας πως δεν υποσκιάζεται η έξοδος κάποιας εργασίας. Η συνδυαστική απώλεια με στατικά βάρη μπορεί να μην προσφέρει την ελαστικότητα που προσφέρουν τα δυναμικά βάρη, ωστόσο μπορεί να αποτελέσει σημαντικό παράγοντα σε περιπτώσεις που υπάρχει χαμηλή διαθεσιμότητα υπολογιστικών πόρων

ή οι δύο εργασίες έχουν την ίδια σημαντικότητα. Εν κατακλείδι, η καθολική συνάρτηση απώλειας του συνολικού μοντέλου με δυναμικά βάρη σε κάθε ξεχωριστή απώλεια διατυπώνεται ως εξής:

$$L_{combined} = \alpha \cdot L_{Transformer} + \beta \cdot L_C$$

, όπου α, β τα βάρη που αποδίδονται στις ανάλογες απώλειες, $L_{Transformer}$ η απώλεια του ανάλογου μοντέλου transformer και L_C η απώλεια του δικτύου ταξινόμησης.
Όσον αφορά των υπολογισμό των δυναμικών βαρών για την καθολική συνάρτηση απώλειας τα βάρος α για την απώλεια του transformer υπολογίζεται ως εξής :

$$\alpha = \frac{||\nabla transformer_loss||}{||\nabla transformer_loss|| + ||\nabla classification_loss|| + e}$$

, όπου $||\nabla classification_loss||$ αποτελεί το κανονικοποιημένο μέτρο των βαθμίδων (gradients) του ταξινομιτή που υπολογίζεται από το μήκος (L2 norm) όλων των βαθμίδων των παραμέτρων που επηρεάζονται από την απώλεια του ταξινομητή, $||\nabla transformer_loss||$ είναι το κανονικοποιημένο μέτρο των βαθμίδων (gradients) του transformer και e είναι ένας πολύ μικρός πραγματικός αριθμός για την αποφυγή διαίρεσης με το μηδέν. Ομοίως το βάρος για την απώλεια του δικτύου ταξινόμησης είναι:

$$\beta = \frac{||\nabla classification_loss||}{||\nabla transformer_loss|| + ||\nabla classification_loss|| + e}$$

Η συνάρτηση απώλειας με δυναμικά βάρη δίνει μεγαλύτερη έμφαση στην απώλεια του μοντέλου transformer καθώς το α αυξάνεται και ομοίως για το β στην απώλεια του ταξινομητή εξασφαλίζοντας την συνεισφορά και των δύο απωλειών με βάση την επίδοσή τους. Με αυτόν τον τρόπο, η διαδικασία δυναμικών βαρών επιτυγχάνει την ισορροπία ανάμεσα στις δύο εργασίες (εντοπισμός σφαλμάτων και επισκευή κώδικα), δίνοντας έμφαση στην εργασία που απαιτεί περισσότερη προσοχή σε κάθε στάδιο εκπαίδευσης, ωστόσο αυξάνει τις προϋποθέσεις υπολογιστικής ισχύς σε αρκετά μεγάλο βαθμό.

5.4 Περιβάλλον Εκπαίδευσης & Καθορισμός Παραμέτρων

Η διαδικασία της εκπαίδευσης πραγματοποιήθηκε ξεχωριστά για κάθε μοντέλο μέσω της πλατφόρμας Google Colab σε μία μονάδα επεξεργασίας γραφικών NVIDIA A100 Tensor Core GPU με μέγεθος μνήμης 40GB. Σχετικά με το CodeT5 το οποίο περιέχει τρεις εκδόσεις με διαφορετικό αριθμό εκπαιδεύσιμων παραμέτρων (222 M), εφαρμόστηκε αυτή με τον μεγαλύτερο αριθμό σε υποσύνολα 64 δειγμάτων για 10 εποχές. Το μέγεθος των υποσυνόλων και ο αριθμός των εποχών στην περίπτωση του CodeBert ήταν ίδια με το CodeT5 με την διαφορά πως το CodeBert διαθέτει λιγότερες εκπαιδεύσιμες παραμέτρους (124 M). Ομοίως και στα δύο μοντέλα ως μέγιστο μήκος ακολουθιών προς επεξεργασία ορίστηκε το 512 λόγω τον περιορισμών μνήμης που είχε η πλατφόρμα Google Colab. Όσον αφορά παραμέτρους που απαιτούνται για την εκπαίδευση, για τον ρυθμό μάθησης δοκιμάστηκαν τιμές στο διάστημα $[10^{-2}, 10^{-5}]$ ενώ για το ποσοστό αποκοπής (dropout rate) επιλέχθηκαν τιμές στο διάστημα $[0.1, 0.5]$. Στο πρόγραμμα

```
Type a bug type to run inference on (general,network-security,functionality, ui-ux, compatibility-performance): general
--------------------- Buggy Code ---------------------
/* Write a function to display the Fibonacci sequence using recursion */
function fibonacci(n) {
  if (n <= 1) {
    return n;
  } else {
    return fibonacci(n + 1) + fibonacci(n + 2);
  }
}
------------------- Masked Buggy Code ----------------
/* Write a function to display the Fibonacci sequence using recursion */
function fibonacci(n) {
  if (n <= 1) {
    return n;
  } else {
    return fibonacci(n<extra_id_0> 1) + fibonacci(n<extra_id_1> 2);
  }
}
-------------------- Correct Code --------------------
/* Write a function to display the Fibonacci sequence using recursion */
function fibonacci(n) {
  if (n <= 1) {
    return n;
  } else {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}
--------------------- Predicted Code -----------------
<extra_id_0> :  -
<extra_id_1> :  -
---------------- Classification ----------------------
Real Classes: ['general', 'functionality'], Predicted classes: ['functionality', 'ui-ux', 'compatibility-performance']
```

Σχήμα 5.1: Ανάκληση στο CodeT5 1

εκπαίδευσης επιλέχθηκε η συνδυαστική απώλεια στο βήμα επικύρωσης (validation step) ως η μετρική απόδοσης του μοντέλου για αποθήκευση, καθώς και εφαρμόστηκε η λογική πρώιμης διακοπής της εκπαίδευσης στις περιπτώσεις που η μετρική υπό παρακολούθηση δεν βελτιωνόταν.

5.5 Ανάκληση

Περαιτέρω, εκτός από προγράμματα εκπαίδευσης και αξιολόγησης, αναπτύχθηκε και ένα πρόγραμμα σε μορφή python notebook με το οποίο δύναται να εκτελεστεί ανάκληση στα μοντέλα, προσπελάζοντας το σημείο αποθήκευσης μοντέλου (checkpoint). Όπως φαίνεται και στα σχήματα 5.1, 5.2 παρακάτω το μοντέλο δέχεται δείγμα κώδικα με το διορθωμένο μπλοκ, το ελαττωματικό μπλοκ και τον τύπο σφάλματος, αρχικά παράγεται ο ελαττωματικός κώδικας όπου τα στοιχεία που άλλαξαν στην διόρθωσή έχουν αποκρυφτεί και έπειτα τα μοντέλα παράγουν τον διορθωμένο κώδικα και τους προβλεπόμενους τύπους σφάλματος.

```
Type a bug type to run inference on (general,network-security,functionality, ui-ux, compatibility-performance): functionality
--------------------- Buggy Code ---------------------
/* Given an array of integers return only the even numbers */
function getEvenNumbers(arr) {
  return arr.filter(num => num % 2 === 1);
  }
------------------ Masked Buggy Code ---------------
/* Given an array of integers return only the even numbers */
function getEvenNumbers(arr) {
  return arr.filter(num => num % 2 ===<mask>);
  }
------------------- Correct Code -------------------
/* Given an array of integers return only the even numbers */
function getEvenNumbers(arr) {  return arr.filter(num => num % 2 === 0);
  }
--------------------- Predicted Code ----------------
<mask> :  0
--------------- Classification ---------------------
Real Classes: ['functionality'], Predicted classes: ['functionality', 'compatibility-performance']
```

Σχήμα 5.2: Ανάκληση στο CodeBert 1

```
Type a bug type to run inference on (general,network-security,functionality, ui-ux, compatibility-performance): ui-ux
---------------------- Buggy Code ---------------------
/* Write a function that logs the string 'Hello' when the user clicks a button */
function buttonAction() {
  const button = document.getElementById('myButton');
button.addEventListener('clock', () => {
  console.log('Hello');
  });
}
------------------- Masked Buggy Code ----------------
/* Write a function that logs the string 'Hello' when the user clicks a button */
function buttonAction() {
  const button = document.getElementById('myButton');
button.addEventListener('<mask>', () => {
  console.log('Hello');
  });
}
-------------------- Correct Code --------------------
/* Write a function that logs the string 'Hello' when the user clicks a button */
function buttonAction() {
  const button = document.getElementById('myButton');
button.addEventListener('click', () => {
  console.log('Hello');
  });
}
--------------------- Predicted Code -----------------
<mask> : click
---------------- Classification ----------------------
Real Classes: ['ui-ux'], Predicted classes: ['functionality', 'ui-ux', 'network-security']
```

Σχήμα 5.3: Ανάκληση στο CodeBert 2

```
Type a bug type to run inference on (general,network-security,functionality, ui-ux, compatibility-performance): ui-ux
--------------------- Buggy Code ---------------------
/* Write a function that logs the string 'Hello' when the user clicks a button */
function buttonAction() {
  const button = document.getElementById('myButton');
button.addEventListener('clock', () => {
  console.log('Hello');
  });
}
------------------- Masked Buggy Code ----------------
/* Write a function that logs the string 'Hello' when the user clicks a button */
function buttonAction() {
  const button = document.getElementById('myButton');
button.addEventListener('<extra_id_0>', () => {
  console.log('Hello');
  });
}
-------------------- Correct Code --------------------
/* Write a function that logs the string 'Hello' when the user clicks a button */
function buttonAction() {
  const button = document.getElementById('myButton');
button.addEventListener('click', () => {
  console.log('Hello');
  });
}
--------------------- Predicted Code -----------------
<extra_id_0> : click
---------------- Classification ----------------------
Real Classes: ['ui-ux'], Predicted classes: ['functionality', 'ui-ux', 'compatibility-performance', 'network-security']
```

Σχήμα 5.4: Ανάκληση στο CodeT5 2

```
Type a bug type to run inference on (general,network-security,functionality, ui-ux, compatibility-performance): network-security
-------------------- Buggy Code ---------------------
/* Write a function that displays a user input on an html element */
function displayUserInput(input) {
  const output = document.getElementById('output');
output.innerHTML = `<p>${input}</p>`;
  }
------------------ Masked Buggy Code ----------------
/* Write a function that displays a user input on an html element */
function displayUserInput(input) {
  const output = document.getElementById('output');
output.<ex<extra_id_1> = `<p>${input}</p>`;
  }
------------------- Correct Code --------------------
/* Write a function that displays a user input on an html element */
function displayUserInput(input) {
  const output = document.getElementById('output');
output.textContent  = input;
  }
-------------------- Predicted Code -----------------
<extra_id_0> : Content
---------------- Classification ---------------------
Real Classes: ['network-security'], Predicted classes: ['functionality', 'ui-ux', 'network-security']
```

Σχήμα 5.5: Ανάκληση στο CodeT5 3

```
Type a bug type to run inference on (general,network-security,functionality, ui-ux, compatibility-performance): compatibility-performance
-------------------- Buggy Code --------------------
/* Given a list of dom element ids with arbitrary length, write a function that changes their background color to yellow */
function highlightItems(ids) {
  ids.forEach(id => {
    const element = document.getElementById(id);
    if (element) {
      element.style.backgroundColor = "yellow";
    }
  });
}
------------------ Masked Buggy Code ---------------
/* Given a list of dom element ids with arbitrary length, write a function that changes their background color to yellow */
function highlightItems(ids) {
  ids.<extra_id_0>(id => {<extra_id_1>    const element = document.getElementById(id);
    if (element) {
      element.style.backgroundColor = "yellow";
    }
  });
}
------------------- Correct Code -------------------
/* Given a list of dom element ids with arbitrary length, write a function that changes their background color to yellow */
function highlightItems(ids) {
  const elements = ids.map(id => document.getElementById(id));
  elements.forEach(element => {
    if (element) {
      element.style.backgroundColor = "yellow";
    }
  });
}
-------------------- Predicted Code ----------------
<extra_id_0> :  =
<extra_id_1> : id
--------------- Classification --------------------
Real Classes: ['compatibility-performance'], Predicted classes: ['functionality', 'ui-ux', 'compatibility-performance', 'network-security']
```

Σχήμα 5.6: Ανάκληση στο CodeT5 4

```
Type a bug type to run inference on (general,network-security,functionality, ui-ux, compatibility-performance): ui-ux
--------------------- Buggy Code ---------------------
/* Write a function that returns a boolean value whether a person is an adult */
function isAdult(age) {
  return age >= 17;
}
------------------- Masked Buggy Code ----------------
/* Write a function that returns a boolean value whether a person is an adult */
function isAdult(age) {
  return age >=<extra_id_0>;
}
-------------------- Correct Code --------------------
/* Write a function that returns a boolean value whether a person is an adult */
function isAdult(age) {
  return age >= 18;
}
--------------------- Predicted Code -----------------
<extra_id_0> :  0
---------------- Classification ---------------------
Real Classes: ['functionality'], Predicted classes: ['functionality', 'compatibility-performance']
```

Σχήμα 5.7: Ανάκληση στο CodeT5 όπου δεν πέτυχε την σωστή διόρθωση

Κεφάλαιο 6

Αποτελέσματα

6.1 Αξιολόγηση

6.1.1 ROUGE

Το ROUGE[26] είναι ένα σύνολο μετρικών που αξιολογεί την συνοχή ενός κομματιού κειμένου, που παράχθηκε από ένα μοντέλο, συγκριτικά με άλλα αντίστοιχα κομμάτια κειμένου που φτιάχτηκαν από ανθρώπους (ground truth data), μετρώντας τον αριθμό των επικαλυπτόμενων στοιχείων όπως ακολουθίες, φράσεις και λέξεις μεταξύ του κειμένου που παράχθηκε από ένα μοντέλο και του κειμένου που παράχθηκε από ανθρώπους. Αποτελεί μία από τις πιο διαδεδομένες μετρικές αξιολόγησης για μοντέλα παραγωγής κειμένου και προσφέρεται σαν πακέτο λογισμικού στην γλώσσα Python.

ROUGE-N

Το ROUGE-N υπολογίζει την επικάλυψη n συνεχόμενων ακολουθιών λέξεων ($n - grams$) μεταξύ του παραγόμενου κειμένου και του πραγματικού κειμένου με τύπο:

$$\frac{\sum_{S \in References} \sum_{gram_n \in S} CountMatch(gram_n)}{\sum_{S \in References} \sum_{gram_n \in S} Count(gram_n)}$$

όπου n είναι το μήκος των ακολουθιών που συγκρίνονται (ROUGE-1, ROUGE-2 κ.α.), $CountMatch(gram_n)$ είναι το σύνολο των n συνεχόμενων επικαλυπτόμενων στοιχείων μεταξύ παραγόμενου και πραγματικού κειμένου και $Count(gram_n)$ είναι το σύνολο των n συνεχόμενων στοιχείων που εμφανίζονται στο πραγματικό κείμενο. Επιπλέον, δεδομένων των συνόλων $CountMatch(gram_n)$ και $Count(gram_n)$ δύναται να υπολογισθούν και οι μετρικές ακρίβειας (Precision) , ανάκλη-

σης (Recall) και F1 ως:

$$P = \frac{\sum CountMatch(gram_n)}{\sum Count(G, gram_n)}$$

όπου $Count(G, gram_n)$ το σύνολο των $n - grams$ του παραγόμενου κειμένου,

$$R = \frac{\sum CountMatch(gram_n)}{\sum Count(R, gram_n)}$$

όπου $Count(R, gram_n)$ το σύνολο των $n - grams$ του πραγματικού κειμένου και

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

ROUGE-L

Το ROUGE-L επιτυγχάνει να δώσει περισσότερο έμφαση στην σειρά και την δομή των υπό ακολουθιών χρησιμοποιώντας την μεγαλύτερου μεγέθους κοινή υπό ακολουθία (LCS) μεταξύ παραγόμενου και πραγματικού κειμένου, όπου οι τιμές P, R, F_1 υπολογίζονται ως:

$$P = \frac{LCS_{length}}{G_{length}}$$

όπου G_{length} είναι το μήκος του παραγόμενου κειμένου,

$$R = \frac{LCS_{length}}{R_{length}}$$

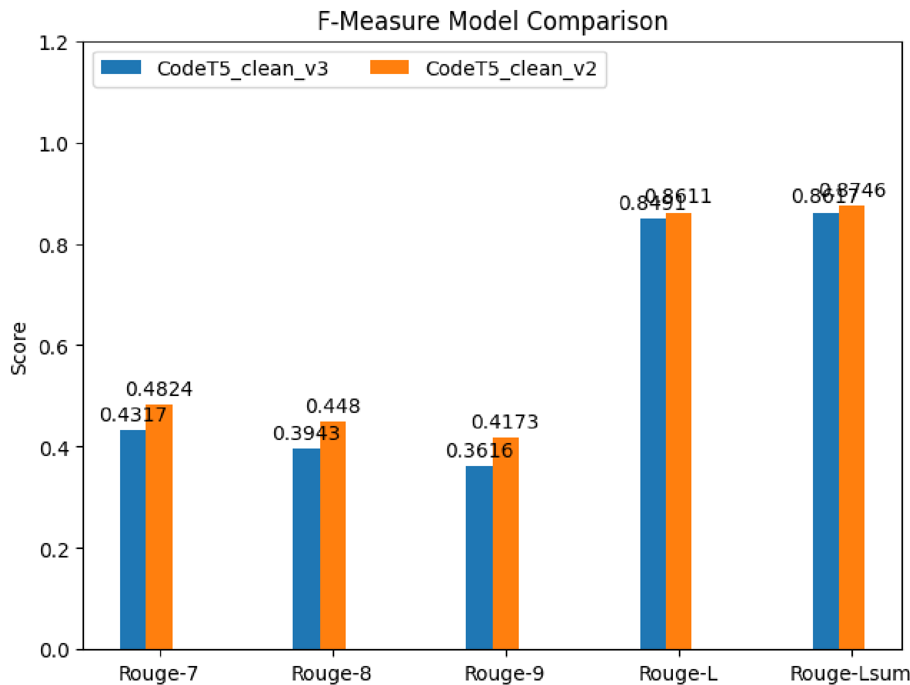
όπου R_{length} είναι το μήκος του πραγματικού κειμένου και το F_1 υπολογίζεται με τον ίδιο τύπο όπως στο ROUGE-N.
Το ROUGE-L διαφέρει απο το ROUGE-N στο γεγονός πως εφαρμόζεται πάνω σε μεγαλύτερες μη διαδοχικές ακολουθίες που διατηρούν τη σειρά των πληροφοριών. Ουσιαστικά, αξιολογεί την συνοχή του παραγόμενου κειμένου με το πραγματικό όσον αφορά την δομή και το περιεχόμενο και όχι την ακριβής εμφάνιση ακολουθιών όπως στο ROUGE-N. Το γεγονός αυτό καθιστά το ROUGE-L αποδοτικότερη μετρική για την αποσφαλμάτωση καθώς πολλές φορές κατά την διόρθωση προβλημάτων οι προγραμματιστές μπορεί να πραγματοποιήσουν αλλαγές στο συντακτικό (π.χ. αλλαγή ονόματος μεταβλητής) χωρίς ωστόσο να αλλάξει η δομή και το περιεχόμενου του κώδικα.
Από την κοινότητα των ερευνητών έχει αναπτυχθεί και μία παραλλαγή του ROUGE-L, το ROUGE-Lsum το οποίο εφαρμόζει τη μέθοδο υπολογισμού ROUGE-L σε επίπεδο προτάσεων και στη συνέχεια συγκεντρώνει όλα τα αποτελέσματα με αθροιστικό τρόπο για την τελική βαθμολογία. Δηλαδή ενώ το ROUGE-L εξετάζει την περίληψη ως σύνολο, το ROUGE-Lsum λαμβάνει υπόψη πληροφορίες σε επίπεδο πρότασης, συγκεκριμένα μέσω του ειδικού συμβόλου "*n*" που δηλώνει νέα σειρά, παρέχοντας ενδεχομένως μεγαλύτερη λεπτομέρεια σε ορισμένες περιπτώσεις χρήσης.

6.1.2 Υλοποίηση

Οι τρεις μετρικές που αναλύθηκαν παραπάνω εφαρμόστηκαν στον κώδικα που παρήγαγαν τα CodeT5, CodeBert στο σύνολο δειγμάτων αξιολόγησης που δημιουργήθηκε από την πηγή COMMITPACK. Όσον αφορά τα δείγματα, παρατηρήθηκε πως το εύρος τον σειρών κυμαίνονταν περίπου στις τριάντα, επομένως επιλέχθηκε να εφαρμοστεί το ROUGE-N για $N = [7, 8, 9]$ καθώς υπάρχουν λέξεις και φράσεις που χρησιμοποιούνται καθολικά και εμφανίζονται αρκετά συχνά στην ανάπτυξη λογισμικού, όπως μεταβλητές με όνομα data, response κ.α.

6.2 Αποτελέσματα CodeT5

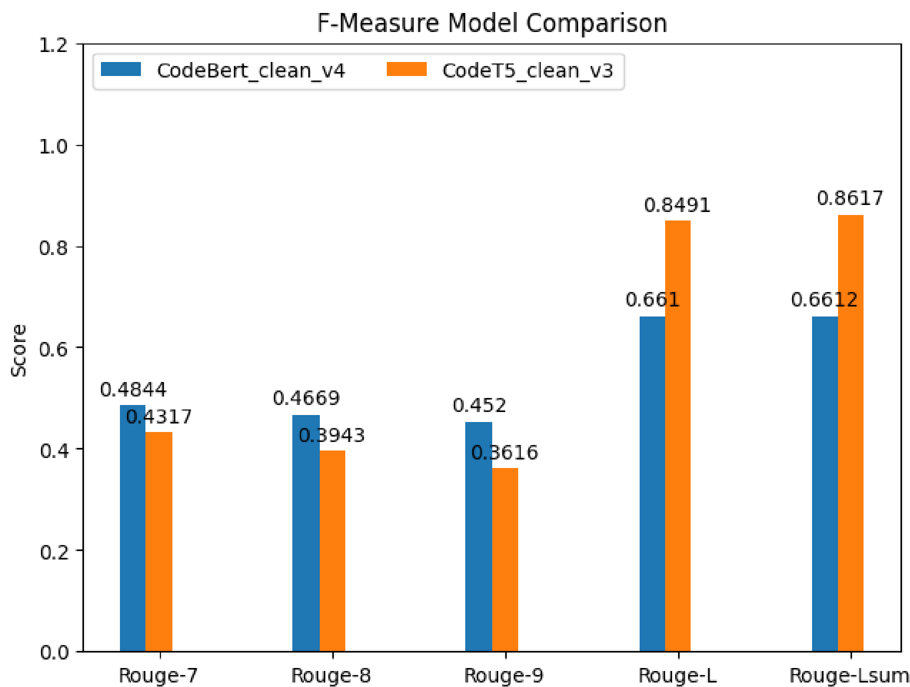
Το μοντέλο που χρησιμοποιούσε το transformer CodeT5 επέφερε την καλύτερη γενική επίδοση με τιμή dropout rate $dr = 0.2$ και ρυθμό εκπαίδευσης $lr = 10^{-5}$, η τιμή της μετρικής προς παρακολούθηση, δηλαδή της συνδυαστικής απώλειας στο βήμα επικύρωσης ήταν $L_{combined} \approx 0.15$. Όσον αφορά το σύνολο αξιολόγησης η καλύτερη μέση επίδοση με βάση την μετρική ROUGE επιτεύχθηκε με μήκος $n - gram = 7$ και τιμή $rougeLsum \approx 0.85$, και για την εργασία ανίχνευσης σφάλματος η καλύτερη επίδοση επιτεύχθηκε στον τύπο ”functionality” και ”general”, το οποίο βασίζεται στο γεγονός πως τα περισσότερα σφάλματα που εμφανίζονται στην JavaScript έχουν να κάνουν με την ορθή λειτουργία του κώδικα, κάτι που σχετίζεται απόλυτα με τις ασύγχρονες δυνατότητες που προσφέρει η JavaScript.



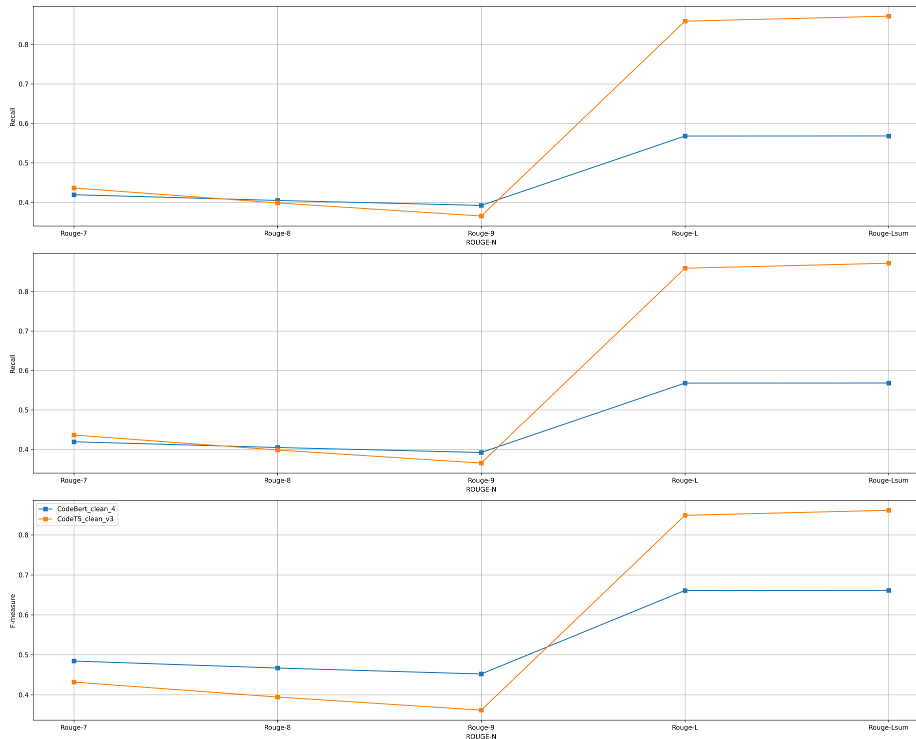
Σχήμα 6.1: Σύγκριση της απόδοσης δύο εκδόσεων του CodeT5 με βάση την μετρική ROUGE

6.3 Αποτελέσματα CodeBert

Το transformer CodeBert πέτυχε καλύτερη επίδοση συνδυαστικής απώλειας $L_{combined} \approx 0.15$,με τιμή dropout rate $dr = 0.2$ και ρυθμό εκπαίδευσης $lr = 10^{-5}$. Στο σύνολο αξιολόγησης η επιδόσεις του μοντέλου με βάση την μετρική ROUGE ήταν ανάλογες με αυτές του CodeT5 όπου παρουσιάστηκε μεγαλύτερη ακρίβεια σε $n - gram$ μικρότερων μεγεθών (<8) κάτι που εξηγείται από το γεγονός πως από την δομή τους δύο μπλοκ κώδικα ενδέχεται να έχουν εκτελούν την ίδια διαδικασία αλλά να έχουν διαφορές σε λεπτομέρειες (π.χ. ονόματα μεταβλητών κ.α.). Ομοίως η καλύτερη επίδοση της ανίχνευσης τύπων σφαλμάτων επιτεύχθηκε στην κατηγορία ”functionality” και ”general”.



Σχήμα 6.2: Σύγκριση επιδόσεων CodeT5 και CodeBert

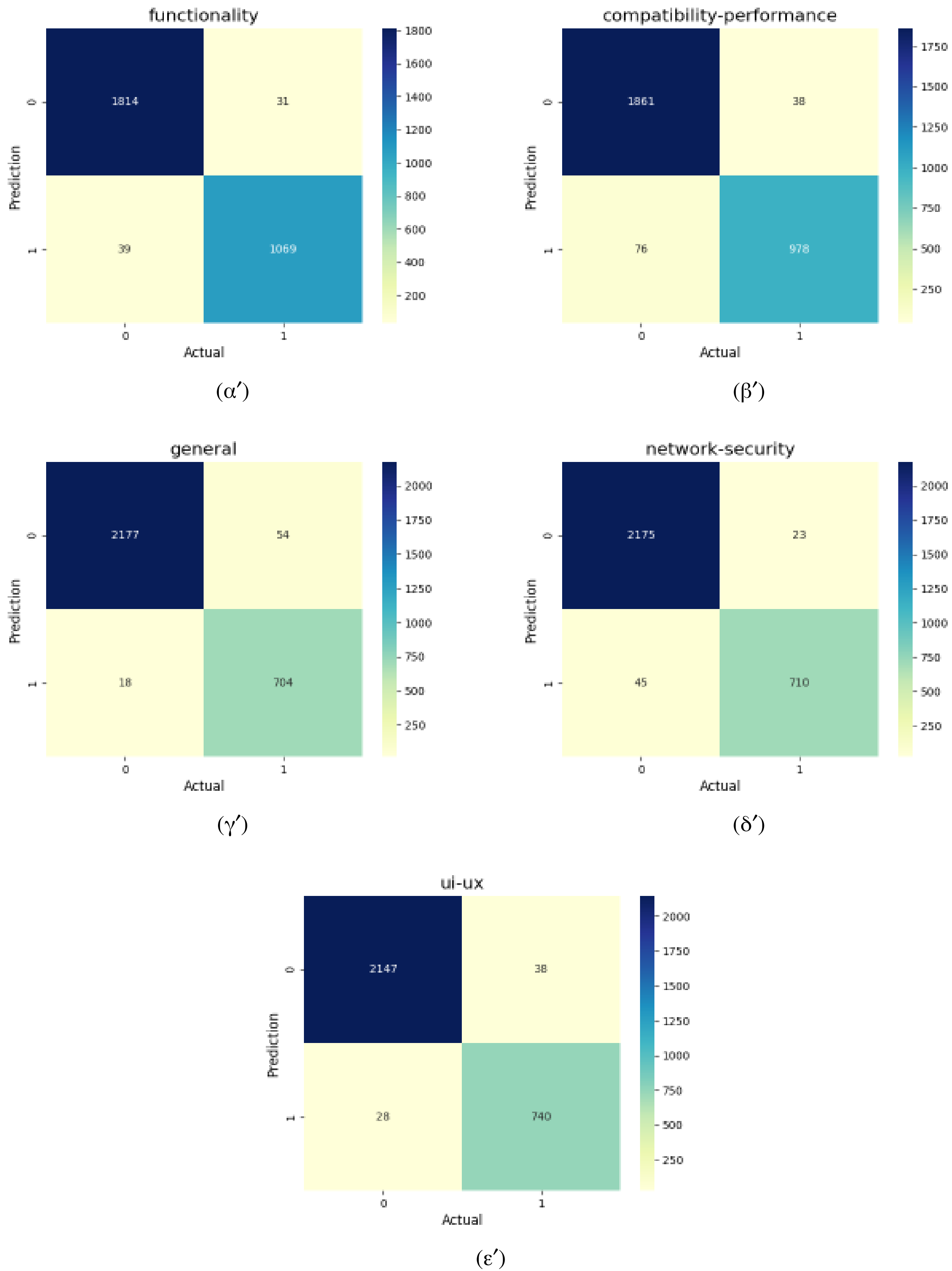


Σχήμα 6.3: Σύγκριση επιδόσεων CodeT5 και CodeBert

Όπως είναι διακριτό από το σχήμα 6.2 και 6.3 η επίδοση των μοντέλων παρουσιάζει αρκετά ενδιαφέροντα στοιχεία. Συγκεκριμένα, στο διάγραμμα τιμών των διαφόρων μετρικών ROUGE για το F-measure πως η συνοχή των παραγόμενων εξόδων του CodeBert είναι καλύτερη από αυτή του CodeT5 στις μετρικές ROUGE με $n - gram = [7, 8, 9]$ γεγονός το οποίο αντιστράφηκε στις μετρικές RougeLsum και RougeL. Η καλύτερη επίδοση του CodeBert με $n - gram$ μικρού μεγέθους αιτιολογείται από το γεγονός πως είναι προ εκπαιδευμένο στην εργασία masked language modeling, όπου το μοντέλο συμπληρώνει αποκρυμμένα στοιχεία, έτσι που βελτιστοποιείτε ώστε να ανιχνεύει συσχετίσεις και νόημα μικρότερα μπλοκ κειμένου. Αναλόγως η αρχιτεκτονική κωδικοποιητή αποκωδικοποιητή που περιέχει το CodeT5 αποδίδει καλύτερα στις μετρικές με βάση την μεγαλύτερου μεγέθους κοινή υπό ακολουθία καθώς επωφελείται από την ικανότητα του να παράγει ακριβή αποτελέσματα με συνοχή αξιοποιώντας τον μηχανισμό αποκωδικοποίησης του. Ο αποκωδικοποιητής μαθαίνει να παράγει τα στοιχεία της ακολουθίας εξόδου αυτό παλινδρομικά καθιστώντας το ικανό να ανακτά συσχετίσεις μεταξύ ακολουθιών μεγάλου μήκους. Συνοψίζοντας, τα αποτελέσματα που επέφεραν τα δύο μοντέλα αποδεικνύουν ότι το CodeBert μπορεί να αποδώσει καλύτερα σε εργασίες διόρθωσης σχετικά μικρών μπλοκ κώδικα, ενώ το CodeT5 ταιριάζει περισσότερο σε εργασίες που απαιτούν διατήρηση της συνοχής, όπως η επιδιόρθωση ακολουθίας, η ανακατασκευή μεγάλων μπλοκ κώδικα ή η αποσφαλμάτωση με πολλές αλληλοσυνδεόμενες εξαρτήσεις.

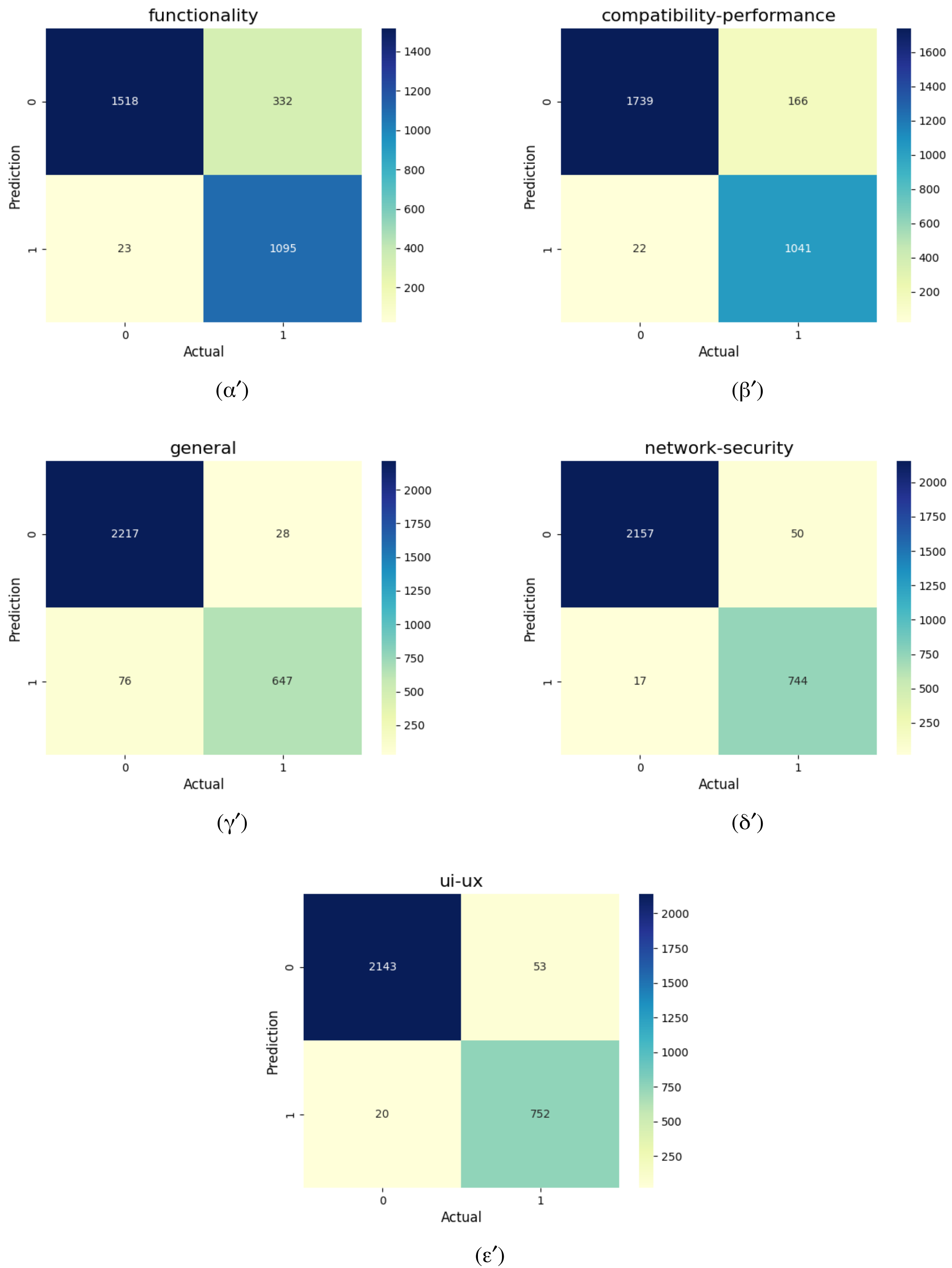
6.4 Αποτελέσματα Ταξινομητή

Όσον αφορά τα δίκτυα ταξινομητή που συνδυάστηκαν με το CodeT5, στο σύνολο επαλήθευσης επιτεύχθηκε ακρίβεια 0.15 ενώ στον ταξινομητή που συνδυάστηκε με το CodeBert επιτεύχθηκε ακρίβεια 0.2. Αναλόγως στο σύνολο αξιολόγησης για τον ταξινομητή του CodeT5 επιτεύχθηκε ακρίβεια 0.26 ενώ στον ταξινομητή του CodeBert επιτεύχθηκε ακρίβεια 0.27. Επιπρόσθετα, ο ταξινομητής του CodeBert και του CodeT5 πέτυχε την καλύτερη ακρίβεια στο σύνολο αξιολόγησης στα δείγματα που ανήκαν στην κλάση ”functionality”, όπως φαίνεται και στα παρακάτω σχήματα η δεύτερη καλύτερη επίδοση και για τους δύο ταξινομητές παρατηρήθηκε στην κλάση ”compatibility-performance”, ενώ αναλόγως η χειρότερη ακρίβεια που επέφεραν και οι δύο ταξινομητές εμφανίστηκε στην κλάση general.



(α′) (β′) (γ′) (δ′) (ε′)

Σχήμα 6.4: Πίνακες Σύγχυσης για κάθε τύπο σφάλματος στον ταξινομητή του CodeBert



Σχήμα 6.5: Πίνακες Σύγχυσης για κάθε τύπο σφάλματος στον ταξινομητή του CodeT5

Κεφάλαιο 7

Θέματα Συζήτησης

7.1 Προκλήσεις

Από την κυκλοφορία της η JavaScript δεν αποτελούσε μία γλώσσα που οι εφαρμογές και χρήσεις της ήταν ξεκάθαρες, ειδικότερα μετά την ένταξη της και σε εφαρμογές server μέσω του *Node.js* [5], όπου θεωρήθηκε μία γλώσσα γενικού σκοπού. Ωστόσο, είναι γνωστό πως δεν κατασκευάστηκε με την σκοπιά μιας γλώσσας γενικού σκοπού, το οποίο σε συνδυασμό με την απουσία μεταγλωττιστή και αυστηρών κανόνων σύνταξης καθώς και την χρήση από προγραμματιστές με ελάχιστη εμπειρία δημιούργησε ένα χώρο εμφάνισης σφαλμάτων και ανεξήγητων συμπεριφορών κατά την εκτέλεση ενός προγράμματος, ιδίως σε συστήματα που η γλώσσα χρησιμοποιείται για διάφορες λειτουργίες πέραν τις εφαρμογές πελάτη, παρά όλη την προσπάθεια της κοινότητας για την αντιμετώπιση τους μέσω πακέτων ή βιβλιοθηκών. Εξαιτίας των παραπάνω, η JavaScript θεωρείται μία από τις πιο δύσκολες γλώσσες για αποσφαλμάτωση, καθώς το εύρος σφαλμάτων είναι πελώριο, και τα περισσότερα δεν εμφανίζονται κατά το χρόνο εκτέλεσης ή δεν δημιουργούν αντικείμενα εξαιρέσεων, αναγκάζοντας έτσι τον προγραμματιστή να τα αναζητήσει σειρά προς σειρά.
Η διαφορετική φύση της γλώσσας, συγκριτικά με τις υπόλοιπες διαδεδομένες γλώσσες σε συνδυασμό με τις δυσκολίες αποσφαλμάτωσης που προκύπτουν καθιστούν δύσκολη την προσαρμογή μοντέλων transformer πάνω στην εργασία αποσφαλμάτωσης, γεγονός που φαίνεται και με την ελάχιστη έρευνα και βιβλιογραφία που υπάρχει πάνω στο θέμα. Διότι οι περισσότεροι ερευνητές επιχειρούν την δημιουργία ή επέκταση μοντέλων στην διόρθωση κώδικα με σκοπό την γενίκευση και εξακρίβωση τύπων σφαλμάτων που είναι καθολικά και συναντιούνται σε πολλές γλώσσες υψηλού επιπέδου. Επιπρόσθετα, η πλειονότητα της ερευνητικής κοινότητας που ασχολείται με την επεξεργασία κώδικα μέσω μοντέλων NLP, έχει αφιερωθεί κυρίως στην ανάπτυξη μοντέλων transformer πάνω στις εργασίες διερμήνευσης κώδικα είτε παραγωγής κώδικα από φυσική γλώσσα (code-to-summary, summary-to-code). Αυτό μπορεί να αποτιμηθεί στο γεγονός πως η διαδικασία της αποσφαλμάτωσης, ειδικότερα στην JavaScript, είναι πολύπλοκη και τις περισσότερες φορές η πλήρης γνώση των λειτουργιών του συστήματος προς αποσφαλμάτωση είναι αναγκαία. Για αυτούς τους λόγους παρατηρούμε την δημοσίευση εργαλείων και εφαρμογών για την υποβοήθηση της αποσφαλμάτωσης όπως το GithubCopilot, τα

οποία ενσωματώνονται στα προγράμματα ανάπτυξης κώδικα και συνήθως έχουν πρόσβαση σε όλη την βάση κώδικα του προγραμματιστή που τα χρησιμοποιεί.
Ένα ακόμη ζήτημα που προκύπτει στην ανάπτυξη γλωσσικών μοντέλων για αποσφαλμάτωση κώδικα αφορά τα σύνολα δεδομένων για εκπαίδευση. Συγκεκριμένα, η κατασκευή συνόλων σε μορφή έτοιμη ώστε να τροφοδοτηθούν σε μοντέλα αποσφαλμάτωσης κώδικα εμπεριέχει διάφορες δυσκολίες οι οποίες σχετίζονται με την φύση των σφαλμάτων στην JavaScript, και διακρίνονται ως:

- Ελλιπής γνώση του συστήματος: Τις περισσότερες φορές ένα σφάλμα (που δεν είναι απλό σφάλμα σύνταξης) απαιτεί την επίγνωση εννοιών και οντοτήτων που δεν εμπεριέχονται στο μπλοκ κώδικα προς αποσφαλμάτωση
- Δεν υπάρχει ακριβής προσδιορισμός ενός σφάλματος στην JavaScript: Εκτός από τα σφάλματα που εμφανίζονται στην ροή εκτέλεσης (π.χ. διαίρεση με το 0), ένα σφάλμα μπορεί να αφορά λανθασμένη λογική, βελτιστοποίηση του χρόνου εκτέλεσης κ.α., κάνοντας τον εντοπισμό του σημείου του σφάλματος αλλά και την κατηγοριοποίηση του δύσκολη. Επιπλέον, ενδέχεται το λάθος να μην βρίσκεται στο ίδιο αρχείο, με το σημείο που συμβαίνει το σφάλμα, ή ακόμη ενδέχεται ένα σφάλμα να τυγχάνει μόνο σε συγκεκριμένες περιπτώσεις (π.χ. όταν ο χρήστης εισάγει συγκεκριμένη τιμή)
- Όταν οι προγραμματιστές διορθώνουν ένα σφάλμα και το αποθηκεύουν σε ένα αποθετήριο όπως το GitHub, συνήθως το μήνυμα δεν εμπεριέχει αρκετή πληροφορία για την αναγνώριση του σφάλματος, καθώς και σε αρκετές περιπτώσεις για την ανίχνευση ενός σφάλματος σε ένα μπλοκ κώδικα είναι αναγκαία η γνώση της πρόθεσης ενός προγραμματιστή

Αρκετά δύσκολη είναι και η διαδικασία κατηγοριοποίησης του σφάλματος σε πολλά δείγματα κώδικα, διότι τις περισσότερες φορές το μήνυμα του προγραμματιστή δεν είναι αρκετό για την πλήρη αναγνώριση και κατηγοριοποίηση του σφάλματος σε κάποιον τύπο. Όπως αναλύθηκε και στην ενότητα 3.1.2 ένας μηχανισμός κατηγοριοποίησης των μηνυμάτων των προγραμματιστών ή κάποια άλλη λύση είναι αναγκαία είτε εν τέλει η προσπέλαση των δειγμάτων από ανθρώπινο μάτι.
Υπάρχουν διάφοροι τρόποι για την βελτίωση των αποτελεσμάτων των μοντέλων καθώς και την αποδοτικότερη εκπαίδευση του μέσω επεξεργασίας των μοντέλων είτε μέσω επεξεργασίας του αρχικού συνόλου δεδομένων. Όσον αφορά το σύνολο δεδομένων, η δημιουργία φίλτρων που θα αποθορυβοποιούσαν το σύνολο, παραδείγματος χάριν την αποφύγη χρήσης στην εκπαίδευση δειγμάτων που παρόλο το μήνυμα υποδήλωνε διόρθωση σφάλματος η αλλαγή ήταν πάνω σε μπλοκ με σχόλια, ή ακόμη και περιπτώσεις που οι αλλαγές στο μπλοκ κώδικα ήταν πάνω από τρεις καθώς παρατηρήθηκε πως και τα δύο μοντέλα δεν απέδιδαν καλά σε δείγματα που χρειαζόταν πάνω από μία η δύο αλλαγές. Συνδυάζοντας την κατασκευή ποιοτικότερων φίλτρων με τον εμπλουτισμό του λεξιλογίου με περισσότερες λέξεις και φράσεις κλειδιά (όπως αναφέρεται στην ενότητα 3.1.2) θα οδηγούσε σε βελτίωση της απόδοσης των μοντέλων.
Μία ακόμη πρόκληση αποτέλεσε η αναγκαία υπολογιστική ισχύς για την αποδοτικότερη εκπαίδευση. Συγκεκριμένα, σε ένα σενάριο που θα ήταν διαθέσιμες πολλαπλές μονάδες επεξεργασίας (GPUs) θα ήταν δυνατό να εφαρμοστεί παράλληλη εκπαίδευση πολλών μοντέλων με

την ίδια αρχιτεκτονική, όπου κάθε μονάδα επεξεργασίας θα εκπαιδευόταν στην επιδιόρθωση συγκεκριμένων τύπων σφαλμάτων (π.χ. μία έκδοση του CodeT5 θα εκπαιδευόταν στην διόρθωση λειτουργικών σφαλμάτων, μία άλλη έκδοση στα σφάλματα διεπαφής χρήστη και ούτω καθεξής) και την χρήση ενός μοντέλου για την ταξινόμηση του σφάλματος. Τέλος, μία διαφορετική στρατηγική που είναι δεδομένο πως θα επέφερε καλύτερα αποτελέσματα καθώς έχει εφαρμοστεί από πολλές εταιρείες και οργανισμού ήδη όπως η ενσωμάτωση του GithubCopilot στο πρόγραμμα ανάπτυξης κώδικα Visual Studio Code, είναι η υλοποίηση της εκπαίδευσης των μοντέλων χρησιμοποιώντας όλη την βάση κώδικα (π.χ. όλο το αποθετήριο git) για καλύτερη διερμήνευση των λειτουργιών που συνεπάγεται και ποιοτικότερη διόρθωση.
Ακόμη μία στρατηγική που ενδέχεται να επέφερε καλύτερα αποτελέσματα θα ήταν η δημιουργία ενός νέου στοχευμένου tokenizer το οποίο θα περιείχε στο λεξιλόγιο του ειδικά token που θα αντιστοιχίζονταν με συγκεκριμένα μοτίβα σφαλμάτων ώστε τα μοντέλα να είναι ικανά να τα ανιχνεύσουν ευκολότερα.

7.2 Τρόποι επέκτασης της υλοποίησης

Η χρήση πλαισίων εφαρμογής, όπως το *LangChain*, παρέχει μια επιστημονική προσέγγιση για τη δημιουργία μιας βάσης δεδομένων για διανύσματα (vector store) τα οποία θα αφορούν δείγματα κώδικα, η οποία μπορεί να αξιοποιηθεί ως μία βάση γνώσης (context) για την ανάπτυξη εφαρμογών RAG (Retrieve, Augment, Generate) με σκοπό την αποσφαλμάτωση εφαρμογών JavaScript ή ακόμη για τη σχεδίαση ολοκληρωμένων εφαρμογών *chatbot* για μια πιο γενικευμένη ανάλυση μιας εφαρμογής JavaScript. Αρχικά, τα δείγματα κώδικα υποβάλλονται σε επεξεργασία μέσω κατάλληλων αλγορίθμων εξαγωγής embeddings, χρησιμοποιώντας μοντέλα όπως τα OpenAI Embeddings ή Hugging Face Transformers. Τα παραγόμενα embeddings αποθηκεύονται στην βάση δεδομένων διανυσμάτων, για την εξασφάλιση ταχείας ανάκτησης σχετικών πληροφοριών. Στη συνέχεια, το *LangChain* επιτρέπει τη διασύνδεση της βάσης δεδομένων για διανύσματα με ένα από τα μοντέλα που αναπτύχθηκαν, διευκολύνοντας την ανάκτηση σχετικού νοήματος, τον εμπλουτισμό του και την παραγωγή απαντήσεων ή συστάσεων. Αυτή η μεθοδολογία καθιστά δυνατή την ανάπτυξη συστημάτων που εντοπίζουν και αναλύουν προβλήματα κώδικα ή παρέχουν στοχευμένες προτάσεις επίλυσης, βασισμένες σε προηγούμενα παραδείγματα, συμβάλλοντας έτσι στην αύξηση της αποδοτικότητας και της ακρίβειας στις διαδικασίες ανάλυσης και επιδιόρθωσης λογισμικού.

Κεφάλαιο 8

Συμπεράσματα

Αρχικά, αναλύθηκε η ανάπτυξη της JavaScript και των χαρακτηριστικών της, καθώς και τις ευπάθειες που δημιουργούνται από τον τρόπο με τον οποίο δομείται. Έπειτα αναλύθηκαν διάφορες τεχνικές και μεθοδολογίες για αποσφαλμάτωση και ανίχνευση μοτίβων σφαλμάτων σε εφαρμογές JavaScript με συμβατικούς τρόπους αλλά και με μοντέλα μηχανικής μάθησης. Συνοψίζοντας, σε αυτή την εργασία εφαρμόστηκε η τεχνική του finetuning πάνω σε δύο μοντέλα αρχιτεκτονικής transformer CodeT5 και CodeBert με σκοπό την επιδιόρθωση μπλοκ κώδικα JavaScript και την ταξινόμηση τους σε πέντε τύπους σφαλμάτων. Τα μοντέλα έχουν προ εκπαιδευτεί σε διάφορες εργασίες που αφορούν την ανάπτυξη λογισμικού σε πολλές γλώσσες. Καθώς και τα δύο μοντέλα χρησιμοποιούν τον τυχαίο μηχανισμό απόκρυψης στοιχείων στις ακολουθίες εισόδου τους, στην συγκεκριμένη περίπτωση επιλέχθηκε μία διαφορετική λογική στην απόκρυψη στοιχείων βασισμένη πάνω στις αλλαγές που προέκυψαν στον κώδικα από την διόρθωση ώστε το μοντέλα να εκπαιδευτούν στην ανίχνευση των στοιχείων που άλλαξαν κατά την διόρθωση του κώδικα. Για αξιολόγηση των μοντέλων εφαρμόστηκε η μέθοδος ROUGE [26] η οποία αποτελεί μία διαδεδομένη μετρική για μοντέλα παραγωγής κειμένου. Όσον αφορά τις επιδόσεις τον μοντέλων αρχικά στην διόρθωση του κώδικα η μετρική ROUGE απέδειξε πως το CodeBert είναι ισχυρότερο στην ανίχνευση μοτίβων και συνεπώς διορθώσεων σε μικρότερα μπλοκ κώδικα λόγω της αρχιτεκτονικής τους. Αναλόγως το CodeT5 αναδείχθηκε ισχυρότερο στην ανάκτηση μοτίβων μεγαλύτερου μεγέθους χωρίς να χάνεται η συνοχή του κώδικα.

8.1 Θέματα Ασφάλειας & προστασίας δεδομένων

Παρόλο που η ανάπτυξη μεγάλων γλωσσικών μοντέλων για αποσφαλμάτωση εφαρμογών αποτελεί πολυάσχολο θέμα έρευνας τα τελευταία χρόνια, ελλοχεύουν κάποια ρίσκα και κίνδυνοι στην ανάπτυξη και χρήση τους. Ένα από τα κύρια θέματα είναι η πιθανότητα παραγωγής λανθασμένων αποτελεσμάτων, καθώς τα γλωσσικά μοντέλα μαθαίνουν μοτίβα με βάση τα δεδομένα εκπαίδευσης τους που τις περισσότερες φορές προέρχονται από δημόσια αποθετήρια, τα οποία μπορεί να μην είναι λειτουργικά ή και απαρχαιωμένα. Ακόμη μεγαλύτερα είναι τα ρίσκα όταν δεν γίνεται επαρκής αξιολόγηση των παραγόμενων κειμένων οδηγώντας σε λύσεις

που αυξάνουν την πολυπλοκότητα και τον χρόνο εκτέλεσης, δημιουργούν νέες ευπάθειες ή/και οδηγούν σε ανεξήγητες συμπεριφορές οδηγώντας ακόμα και σε νομικά ζητήματα.
Ακόμη ένας τεχνικός κίνδυνος που έγκειται κυρίως στο κομμάτι ανάκλησης των μοντέλων είναι η αδυναμία επεξήγησης των αποτελεσμάτων που παράγουν τα μοντέλα. Ειδικότερα σε εφαρμογές υψηλής σημασίας, ένα μοντέλο διόρθωσης κώδικα, το οποίο δεν κατέχει την εννοιολογική σκοπιά της εφαρμογής ενδέχεται να παράξει εσφαλμένες λύσεις ή/και να παραμελήσει σφάλματα. Σε αυτό το κομμάτι είναι κρίσιμο να παρατεθεί και ο κίνδυνος της υπερβολικής εξάρτησης που ενδέχεται να έχει ένας προγραμματιστής από το μοντέλο, θεωρώντας τις εξόδους του αλάνθαστες παραμελώντας την αξιολόγηση τους και την διαδικασία δοκιμών της εφαρμογής που υλοποιείται. Επομένως είναι πολύ σημαντικό ο κάθε χρήστης γλωσσικού μοντέλου για εργασίες ανάπτυξης λογισμικού να συνδυάζει την βοήθεια που παρέχεται από το μοντέλο σε συνδυασμό με της διαδεδομένες πρακτικές ανάπτυξης λογισμικού.

Βιβλιογραφία

[1] Y. Hu, X. Shi, Q. Zhou και L. Pike, "Fix bugs with transformer through a neural-symbolic edit grammar", *arXiv preprint arXiv:2204.06643*, 2022.

[2] Y. Wang, W. Wang, S. Joty και S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation", *arXiv preprint arXiv:2109.00859*, 2021.

[3] Z. Feng, D. Guo, D. Tang κ.ά., "Codebert: A pre-trained model for programming and natural languages", *arXiv preprint arXiv:2002.08155*, 2020.

[4] A. Wirfs-Brock και B. Eich, "JavaScript: the first 20 years", *Proc. ACM Program. Lang.*, τόμ. 4, αρθμ. HOPL, Ιούν. 2020. doi: `10.1145/3386327`. διεύθν.: `https://doi.org/10.1145/3386327`.

[5] OpenJS Foundation, *Node.js*, `https://nodejs.org/`, 2009.

[6] K. Sun και S. Ryu, "Analysis of JavaScript Programs: Challenges and Research Trends", *ACM Comput. Surv.*, τόμ. 50, αρθμ. 4, Αύγ. 2017, issn: 0360-0300. doi: `10.1145/3106741`. διεύθν.: `https://doi.org/10.1145/3106741`.

[7] A. Turcotte, M. D. Shah, M. W. Aldrich και F. Tip, "DrAsync: identifying and visualizing anti-patterns in asynchronous JavaScript", στο *Proceedings of the 44th International Conference on Software Engineering*, σειρά ICSE '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, σσ. 774–785, isbn: 9781450392211. doi: `10.1145/3510003.3510097`. διεύθν.: `https://doi.org/10.1145/3510003.3510097`.

[8] A. Vaswani, "Attention is all you need", *Advances in Neural Information Processing Systems*, 2017.

[9] J. Hutchins, "The first public demonstration of machine translation: the Georgetown-IBM system, 7th January 1954", *noviembre de*, 2005.

[10] S. Hochreiter, "Long Short-term Memory", *Neural Computation MIT-Press*, 1997.

[11] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition", *Proceedings of the IEEE*, τόμ. 77, αρθμ. 2, σσ. 257–286, 1989.

[12] C. Cortes, "Support-Vector Networks", *Machine Learning*, 1995.

[13] T. Mikolov, "Efficient estimation of word representations in vector space", *arXiv preprint arXiv:1301.3781*, τόμ. 3781, 2013.

[14] OpenJS Foundation, *jQuery*, `https://jquery.com/`, 2006.

[15] Meta Platforms, Inc., *reactJS*, `https://react.dev`, 2013.

[16] E. Arteca, M. Schäfer και F. Tip, "Learning how to listen: Automatically finding bug patterns in event-driven javascript apis", *IEEE Transactions on Software Engineering*, τόμ. 49, αρθμ. 1, σσ. 166–184, 2022.

[17] M. Madsen, F. Tip και O. Lhoták, "Static analysis of event-driven Node. js JavaScript applications", *ACM SIGPLAN Notices*, τόμ. 50, αρθμ. 10, σσ. 505–519, 2015.

[18] M. Madsen, O. Lhoták και F. Tip, "A model for reasoning about JavaScript promises", *Proceedings of the ACM on Programming Languages*, τόμ. 1, αρθμ. OOPSLA, σσ. 1–24, 2017.

[19] M. Rabinovich, M. Stern και D. Klein, "Abstract syntax networks for code generation and semantic parsing", *arXiv preprint arXiv:1704.07535*, 2017.

[20] U. Alon, M. Zilberstein, O. Levy και E. Yahav, "code2vec: learning distributed representations of code", *Proc. ACM Program. Lang.*, τόμ. 3, αρθμ. POPL, Ιαν. 2019. doi: `10.1145/3290353`. διεύθν.: `https://doi.org/10.1145/3290353`.

[21] Hugging Face, Inc., *Hugging Face: State-of-the-Art Machine Learning Models*, `https://huggingface.co/`, Accessed: 2024-12-24, 2024.

[22] N. Muennighoff, Q. Liu, A. Zebaze κ.ά., "Octopack: Instruction tuning code large language models", *arXiv preprint arXiv:2308.07124*, 2023.

[23] S. Lu, D. Guo, S. Ren κ.ά., "Codexglue: A machine learning benchmark dataset for code understanding and generation", *arXiv preprint arXiv:2102.04664*, 2021.

[24] M. Chen, J. Tworek, H. Jun κ.ά., "Evaluating large language models trained on code", *arXiv preprint arXiv:2107.03374*, 2021.

[25] P. S. Foundation, *difflib: Helpers for computing deltas*, έκδ. 3.x, Part of the Python Standard Library, 1990. διεύθν.: `https://docs.python.org/3/library/difflib.html`.

[26] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries", στο *Text summarization branches out*, 2004, σσ. 74–81.