ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΕΓΑΛΑ ΓΛΩΣΣΙΚΑ ΜΟΝΤΕΛΑ ΣΤΗΝ ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

Διπλωματική Εργασία

Δημήτριος Κράβαρης

Επιβλέπων: Δημήτριος Κατσαρός

Ιούνιος 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΕΓΑΛΑ ΓΛΩΣΣΙΚΑ ΜΟΝΤΕΛΑ ΣΤΗΝ ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

Διπλωματική Εργασία

Δημήτριος Κράβαρης

Επιβλέπων: Δημήτριος Κατσαρός

Ιούνιος 2025

[3]

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

LARGE LANGUAGE MODELS AND SOFTWARE ENGINEERING

Diploma Thesis

Dimitrios Kravaris

Supervisor: Dimitris Katsaros

June 2025

[5]

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπων **Δημήτριος Κατσαρός**

Αναπληρωτής καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Δημήτριος Ραφαηλίδης**

Αναπληρωτής καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Γεώργιος Θάνος**

Ε.ΔΙ.Π., Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας.

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελούν αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλουν οποιασδήποτε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχουν έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Δηλώνω επίσης ότι τα αποτελέσματα της εργασίας δεν έχουν χρησιμοποιηθεί για την απόκτηση άλλου πτυχίου. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο/Η Δηλών/ούσα

Δημήτριος Κράβαρης

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The Declarant

Dimitrios Kravaris

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον καθηγητή του Πανεπιστημίου Θεσσαλίας και επιβλέπων αυτής της διπλωματικής, κύριο Δημήτριο Κατσαρό για όλη τη καθοδήγηση που μου προσέφερε, λύνοντάς μου απορίες και προβληματισμούς καθ' όλη τη διάρκεια συγγραφής της διπλωματικής. Επιπλέον, θα ήθελα να ευχαριστήσω τους καθηγητές, κύριο Δημήτριο Ραφαηλίδη και κύριο Γεώργιο Θάνο που υπήρξαν μέλη της επιτροπής εξέτασης. Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου που υπάρχουν δίπλα μου σε κάθε βήμα.

Διπλωματική Εργασία

ΜΕΓΑΛΑ ΓΛΩΣΣΙΚΑ ΜΟΝΤΕΛΑ

ΣΤΗΝ ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

Δημήτριος Κράβαρης

Περίληψη

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι να διερευνήσει το τρόπο με τον οποίο τα μεγάλα γλωσσικά μοντέλα ασκούν επιρροή στο τομέα ανάπτυξης λογισμικού. Αρχικά θα δοθεί μια αναλυτική περιγραφή του ρόλου ενός software developer και της διαδικασίας που ακολουθεί σε όλα τα στάδια ανάπτυξης λογισμικού και τα προβλήματα που αντιμετωπίζει. Στην συνέχεια, θα αναφερθούμε στο τομέα της τεχνητής νοημοσύνης, Natural Language Processing (NLP), και θα παρουσιάσουμε κάποια Large Language Models (LLMs), όπως το GPT, το Codex και το CodeT5 εστιάζοντας στις εφαρμογές τους στην συγγραφή και διόρθωση κώδικα. Στο πλαίσιο της διπλωματικής θα αναλύσουμε το κώδικα που έχει παραχθεί από έναν προγραμματιστή σε σύγκριση με κώδικα που παράγουν τα LLMs, ελέγχοντας την αποτελεσματικότητα, την ταχύτητα και την ακρίβεια, αλλά και την ικανότητα των μοντέλων αυτών να εντοπίζουν και να διορθώνουν λάθη στο κώδικά τους ή στο κώδικα των προγραμματιστών. Τέλος, θα αξιολογήσουμε την επίδραση των LLMs στην τεχνολογία λογισμικού, υπογραμμίζοντας ότι με τη σωστή εκπαίδευση και έλεγχο πάνω στη χρήση τους μπορούν να λειτουργούν ως εργαλεία για τους προγραμματιστές και όχι αντικαταστάτες τους.

Λέξεις-κλειδιά:

μεγάλα γλωσσικά μοντέλα, ChatGPT, παραγωγή κώδικα, ανίχνευση λαθών

Diploma Thesis

LARGE LANGUAGE MODELS AND SOFTWARE ENGINEERING

Dimitrios Kravaris

Abstract

The aim of the thesis is to explore the way large language models impact software development. First, we will analyze the role of the software developer and the process he follows, including all the stages of developing software and the problems that may occur. Next, we will talk about Artificial Intelligence and Natural Language Processing (NLP) and mention some Large Language Models like GPT, Codex and CodeT5 while showcasing their applications in code generation and correction. In the context of the thesis, we will analyze the code produced from an LLM compared to human-written code, examining the effectiveness, writing speed and accuracy, but also the capability of the models to debug code produced by them or by programmers. Finally, we will assess the impact of LLMs in the field of software engineering, emphasizing that with correct training and control over their usage, Large Language Models can be tools to help programmers instead of replacing them.

Keywords:

large language models, ChatGPT, code generation, bug detection

Περιεχόμενα

Κατάλογος εικόνων

Συντομογραφίες

NLP Natural Language Processing

LLMs Large Language Models

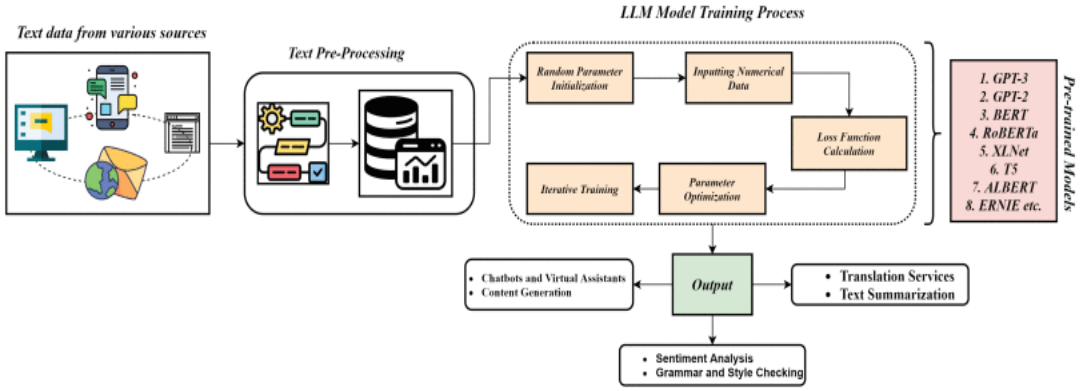
κλπ. και λοιπά

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Τα τελευταία χρόνια, η ταχεία εξέλιξη του τομέα της τεχνητής νοημοσύνης και ειδικότερα των μεγάλων γλωσσικών μοντέλων (Large Language Models) έχει φέρει ριζικές αλλαγές στο τομέα ανάπτυξης λογισμικού. Κατά συνέπεια, ο ρόλος του προγραμματιστή, που περιλαμβάνει κατά κύριο λόγο τη συγγραφή, τη σχεδίαση, την υλοποίηση και τη συντήρηση του κώδικα, πλέον επηρεάζεται από τα μοντέλα αυτά. Τα μεγάλα γλωσσικά μοντέλα προσφέρουν νέες δυνατότητες και αυξημένη παραγωγικότητα στο τομέα ανάπτυξης λογισμικού, αλλά ταυτόχρονα δημιουργούν ερωτήματα που αφορούν την ακρίβεια και την αξιοπιστία τους και πόσο ηθική είναι η χρήση τους.

Τι είναι όμως τα μεγάλα γλωσσικά μοντέλα και από τι αποτελούνται; Τα μεγάλα γλωσσικά μοντέλα η αλλιώς LLMs, είναι μοντέλα βαθιάς μάθησης που έχουν εκπαιδευτεί σε τεράστια ποσότητα δεδομένων. Η αρχιτεκτονική τους αποτελείται από νευρωνικά δίκτυα, Transformer, τα οποία χρησιμοποιούν έναν κωδικοποιητή (encoder) και έναν αποκωδικοποιητή (decoder), μέσω των οποίων εξάγουν το νόημα από μια πρόταση κειμένου και κατανοούν τις σχέσεις μεταξύ των λέξεων και των φράσεων[1]. Τα LLMs έχουν τη δυνατότητα να εκπαιδεύουν μόνα τους τον εαυτό τους μαθαίνοντας να κατανοούν βασική γραμματική, διάφορες γλώσσες και γενικές γνώσεις. Επίσης, μέσω της αρχιτεκτονικής Transformer μπορούν να επεξεργάζονται παράλληλα ακολουθίες δεδομένων με αποτέλεσμα να εκπαιδεύονται πολύ πιο γρήγορα σε αντίθεση με παλαιότερα νευρωνικά δίκτυα(RNNs), τα οποία επεξεργάζονταν το πλήθος δεδομένων ακολουθιακά.



Εικόνα 1: Διάγραμμα της διαδικασίας εκπαίδευσης των Μεγάλων Γλωσσικών Μοντέλων[2].

1.1 Αντικείμενο της διπλωματικής

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι να διερευνήσει το τρόπο με τον οποίο τα μεγάλα γλωσσικά μοντέλα ασκούν επιρροή στο τομέα ανάπτυξης λογισμικού. Αρχικά θα δοθεί μια αναλυτική περιγραφή του ρόλου ενός software developer και της διαδικασίας που ακολουθεί σε όλα τα στάδια ανάπτυξης λογισμικού και τα προβλήματα που αντιμετωπίζει. Στην συνέχεια, θα αναφερθούμε στο τομέα της τεχνητής νοημοσύνης, Natural Language Processing (NLP), και θα παρουσιάσουμε κάποια Large Language Models (LLMs), όπως το GPT, το Codex και το CodeT5 εστιάζοντας στις εφαρμογές τους στην συγγραφή και διόρθωση κώδικα. Στο πλαίσιο της διπλωματικής θα αναλύσουμε το κώδικα που έχει παραχθεί από έναν προγραμματιστή σε σύγκριση με κώδικα που παράγουν τα LLMs, ελέγχοντας την αποτελεσματικότητα, την ταχύτητα και την ακρίβεια, αλλά και την ικανότητα των μοντέλων αυτών να εντοπίζουν και να διορθώνουν λάθη στο κώδικά τους ή στο κώδικα των προγραμματιστών. Τέλος, θα αξιολογήσουμε την επίδραση των LLMs στην τεχνολογία λογισμικού, υπογραμμίζοντας ότι με τη σωστή εκπαίδευση και έλεγχο πάνω στη χρήση τους μπορούν να λειτουργούν ως εργαλεία για τους προγραμματιστές και όχι αντικαταστάτες τους.

ΚΕΦΑΛΑΙΟ 2

Ο ΡΟΛΟΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ ΛΟΓΙΣΜΙΚΟΥ

2.1 Εισαγωγή

Ο προγραμματιστής αποτελεί έναν από τους πιο κρίσιμους ρόλους στον τομέα της τεχνολογίας λογισμικού, δίνοντας ιδέες και παράγοντας λύσεις στα προβλήματα που προκύπτουν. Ο ρόλος του εκτός από τη συγγραφή κώδικα είναι και η σχεδίαση, ανάπτυξη, δοκιμή, συντήρηση και βελτίωση λογισμικού που εξυπηρετεί συγκεκριμένες ανάγκες και απαιτήσεις[3]. Η εργασία ενός προγραμματιστή ξεκινά από την κατανόηση των απαιτήσεων του εκάστοτε project που έχει αναλάβει. Σε αυτή τη φάση χρειάζεται να συνεργαστεί με διάφορα τμήματα όπως σχεδιαστές, αναλυτές και πελάτες για να προσδιοριστούν οι στόχοι και οι προδιαγραφές που πρέπει να ακολουθηθούν. Στη συνέχεια, ο προγραμματιστής αρχίζει να σχεδιάζει τον κώδικα, δημιουργώντας λειτουργικές λύσεις και διαλέγοντας τις κατάλληλες τεχνολογίες για να τις υλοποιήσει. Κατά την υλοποίηση, ο προγραμματιστής γράφει τον αρχικό κώδικα, ύστερα ελέγχει για τυχόν λάθη ή δυσλειτουργίες και τέλος τον βελτιστοποιεί διασφαλίζοντας ότι λειτουργεί σωστά σύμφωνα με τις απαιτήσεις που υφίστανται. Μετά την υλοποίηση του κώδικα, ξεκινάει τη διαδικασία δοκιμών για να εντοπίσει και να διορθώσει σφάλματα, ενώ εξίσου σημαντικό είναι να φροντίσει την ασφάλεια και ανθεκτικότητα του προγράμματος που δημιουργεί[4]. Ωστόσο, ο ρόλος του προγραμματιστή δεν τελειώνει εδώ, καθώς θα πρέπει να συντηρεί το λογισμικό που δημιούργησε διασφαλίζοντας ότι παραμένει λειτουργικό και ενημερωμένο για να μπορεί να ανταπεξέλθει στις νέες απαιτήσεις. Εκτός από τεχνικές δεξιότητες που πρέπει να έχει ένας σύγχρονος προγραμματιστής, είναι σημαντικό να διαθέτει επικοινωνιακές ικανότητες και ομαδικό πνεύμα καθώς χρειάζεται να εργάζεται συχνά σε ομαδικά project πιο σύνθετης δυσκολίας και υπό τη πίεση χρόνου. Με τη ραγδαία εξέλιξη της τεχνολογίας, αναμενόμενα ο ρόλος του προγραμματιστή δυσκολεύει καθώς απαιτεί συνεχή μάθηση, ενημέρωση και προσαρμογή στα νέα δεδομένα.

2.2 Τα καθήκοντα του προγραμματιστή

2.2.1 Σχεδιασμός και ανάπτυξη λογισμικού

Οι προγραμματιστές είναι ο σημαντικότερος παράγοντας στη σχεδίαση και ανάπτυξη λογισμικού που εξυπηρετεί τις ανάγκες της εκάστοτε εταιρείας ή των χρηστών. Πρέπει να μπορούν να χρησιμοποιούν διάφορες γλώσσες προγραμματισμού και εργαλεία αναλόγως το λογισμικό που έχουν σκοπό να δημιουργήσουν. Καθ' όλη τη διάρκεια της διαδικασίας σχεδιασμού του λογισμικού είναι απαραίτητη η συνεργασία με άλλους προγραμματιστές αλλά και με άλλα τμήματα της εταιρείας. Οι προγραμματιστές ασχολούνται τόσο με τη δημιουργία νέων εφαρμογών όσο και με την βελτίωση ή την συντήρηση των ήδη υπαρχουσών[5]. Κάθε επιχείρηση και κάθε πελάτης έχει διαφορετικές ανάγκες οπότε ο κύριος στόχος των προγραμματιστών είναι να φροντίσουν το λογισμικό που παράγουν να ανταποκρίνεται σε αυτές τις ανάγκες. Αυτό σημαίνει ότι οι προγραμματιστές πρέπει να μπορούν να κατανοήσουν πλήρως τα προβλήματα και τις απαιτήσεις του κάθε πελάτη και να δρουν αναλόγως. Συνεπώς, μετά από συσκέψεις μεταξύ προγραμματιστών και πελατών και αναλύσεις των προβλημάτων αποφασίζουν ποια λειτουργικά χαρακτηριστικά θα υλοποιήσουν μέσα στο λογισμικό και με ποιον τρόπο θα καλύψουν τις ανάγκες των χρηστών.

Εικόνα 2.2.1: Διάφορες γλώσσες προγραμματισμού[6].

Ανάλογα τη φύση του κάθε project, οι προγραμματιστές χρησιμοποιούν και την αντίστοιχη γλώσσα προγραμματισμού όπως η Python, η JavaScript, η R και άλλες. Ποια γλώσσα θα επιλέξουν να χρησιμοποιήσουν εξαρτάται από τις απαιτήσεις που έχει το συγκεκριμένο project, δηλαδή η ταχύτητα, η ασφάλεια, η ευκολία χρήσης και αν μπορεί να ενσωματωθεί σε άλλα συστήματα. Για παράδειγμα, αν θέλουν να δημιουργήσουν μια εφαρμογή για κινητά τηλέφωνα θα χρησιμοποιήσουν τις γλώσσες Java ή JavaScript καθώς αυτές τους δίνουν τα κατάλληλα εργαλεία, ενώ αν θέλουν να δημιουργήσουν μια ιστοσελίδα θα χρησιμοποιήσουν τις γλώσσες HTML, CSS και JavaScript. Πέρα όμως από την επιλογή της γλώσσας προγραμματισμού, η διαδικασία της ανάπτυξης του λογισμικού περιέχει και τη χρήση διαφόρων περιβαλλόντων ανάπτυξης (Integrated development environment, IDE) τα οποία προσφέρουν επιπρόσθετα εργαλεία που συμβάλλουν στην ανάλυση του κώδικα, στη βελτιστοποίησή του και στον εντοπισμό λαθών[7].

2.2.2 Συντήρηση και βελτίωση λογισμικού

Μετά από τη διαδικασία ανάπτυξης του λογισμικού οι προγραμματιστές οφείλουν να το συντηρούν επιλύοντας σφάλματα που προκύπτουν και ενημερώνοντας το για νέες λειτουργίες ή νέες ανάγκες που δημιουργούνται. Είναι απαραίτητο για τη μακροζωία του λογισμικού να συνεχίσει να δέχεται αναβαθμίσεις στο κώδικα που θα βελτιώσουν την απόδοσή του και την ασφάλεια ώστε να παραμένει λειτουργικό, αποδοτικό και αξιόπιστο ακόμη και μετά τη κυκλοφορία του.

Αρχικά η συντήρηση λογισμικού αφορά κυρίως την επίλυση σφαλμάτων ή προβλημάτων που μπορεί να προκύψουν με την πάροδο του χρόνου. Τα προβλήματα αυτά είναι διαφόρων ειδών όπως για παράδειγμα σφάλματα στο κώδικα που δεν αναγνώρισαν οι προγραμματιστές όταν έκαναν τις απαραίτητες δοκιμές ή να υπάρξει ασυμβατότητα και απρόβλεπτες αλληλεπιδράσεις του λογισμικού με κάποια νέα τεχνολογία λειτουργικών συστημάτων. Για αυτό, τα σφάλματα αυτά πρέπει να εντοπίζονται και να επιλύονται γρήγορα με τέτοιο τρόπο έτσι ώστε να μην επηρεάζουν αρνητικά τις υπόλοιπες προυπάρχουσες λειτουργίες του λογισμικού. Ένα ακόμη σημαντικό κομμάτι της συντήρησης είναι και η παρακολούθηση της απόδοσης του λογισμικού σε πραγματικές συνθήκες χρήσης, δηλαδή ένα πρόγραμμα να αρχίσει να

μην αποδίδει όπως θα έπρεπε λόγω της υψηλής κατανάλωσης πόρων και να καθυστερεί. Τότε οι προγραμματιστές οφείλουν να τροποποιήσουν κατάλληλα τον κώδικα ώστε να διασφαλιστεί η ομαλή και αποτελεσματική λειτουργία του προγράμματος[8].

Από την άλλη εξίσου σημαντικό είναι οι προγραμματιστές, εκτός από τη συντήρηση, να συμβάλλουν ενεργά και στη βελτίωση του λογισμικού. Όσο οι απαιτήσεις αυξάνονται τόσο το λογισμικό πρέπει να εξελίσσεται και να προσαρμόζεται στις νέες συνθήκες, για το οποίο φροντίζουν οι προγραμματιστές ενημερώνοντας συνεχώς τον κώδικα. Για παράδειγμα, ένα λογισμικό που αναπτύχθηκε πριν από μία δεκαετία δεν θα είναι πια το ίδιο αποδοτικό με πιο πρόσφατες κυκλοφορίες αντίστοιχων λογισμικών. Κατά συνέπεια, μπορεί να χρειαστεί να γίνουν ενέργειες με σκοπό την αναβάθμιση του κώδικα είτε με τον ανασχεδιασμό κομματιών του αρχικού κώδικα είτε με τη βελτίωση των αλγορίθμων είτε ακόμη και με την αντικατάσταση παλιών τεχνολογιών με νεότερες και πιο αποδοτικές.

2.2.3 Αναβάθμιση ασφάλειας

Η ασφάλεια είναι ένα από τα πιο σημαντικά ζητήματα στη διαδικασία της ανάπτυξης και συντήρησης του λογισμικού. Οι προγραμματιστές οφείλουν να φροντίσουν για την ανθεκτικότητα του λογισμικού, δηλαδή να μην είναι ευάλωτο από νέες απειλές που μπορεί να εμφανιστούν στο μέλλον. Για αυτό είναι απαραίτητο να γίνονται συνεχείς ενημερώσεις ασφαλείας όπως η διόρθωση ευπαθών σημείων στον κώδικα ή η προσθήκη καινούριων μηχανισμών ελέγχου με σκοπό την προστασία των δεδομένων. Κάποιοι βασικοί τρόποι για να διατηρείται η ασφάλεια και η ακεραιότητα του λογισμικού είναι η κρυπτογράφηση των δεδομένων, συχνή ενημέρωση του λογισμικού για να μην είναι ευάλωτο σε νέους κινδύνους και ο τακτικός έλεγχος της ταυτότητας των χρηστών. Τελικώς, η συντήρηση και βελτίωση του λογισμικού είναι μια διαδικασία που δεν σταματάει καθώς και οι τεχνολογίες και οι απαιτήσεις της αγοράς εξελίσσονται συνεχώς[9]. Οπότε για να παραμείνει το λογισμικό αποδοτικό και να συμβαδίζει με τις τεχνολογικές εξελίξεις χρειάζεται να γίνονται συχνές ενημερώσεις του κώδικα και προσαρμογή των μέτρων ασφαλείας.

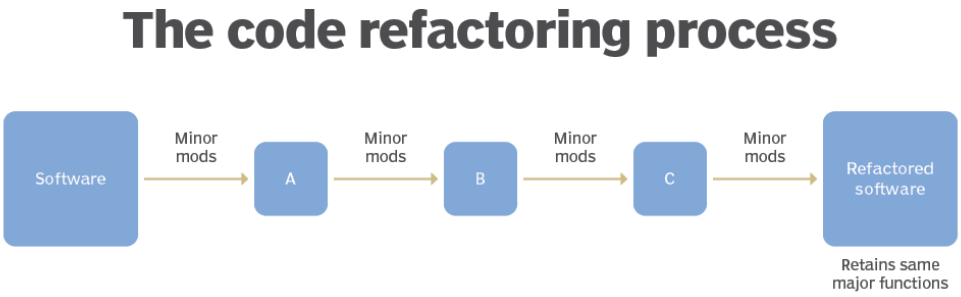
Εικόνα 2.2.3: Η σημαντικότητα και τα επίπεδα της ασφάλειας λογισμικού[10].

2.2.4 Επίλυση προβλημάτων

Όσο περνάνε τα χρόνια και η τεχνολογία εξελίσσεται είναι λογικό να εμφανίζονται νέα τεχνικά ζητήματα που επηρεάζουν την αποδοτικότητα και την ασφάλεια των συστημάτων. Οι προγραμματιστές οφείλουν να αναγνωρίζουν γρήγορα τις ιδιαιτερότητες του κάθε προβλήματος και να δίνουν λύσεις που θα συμβάλλουν στη βελτίωσή του αλλά και την προστασία του από παρόμοιες καταστάσεις. Το πρώτο βήμα αυτής της διαδικασίας αντιμετώπισης προβλημάτων είναι να βρεθεί η φύση του προβλήματος και αυτό επιτυγχάνεται με τη χρήση εργαλείων όπως το JIRA, Bugzilla και άλλα, που τους δίνουν τη δυνατότητα να φτάσουν στη ρίζα του προβλήματος. Σαν δεύτερο βήμα, ακολουθεί η ανάλυση και διεξοδική εξέταση του κώδικα κατά την οποία οι προγραμματιστές αφού έχουν προσδιορίσει τα σφάλματα, προβαίνουν σε διορθώσεις που λύνουν τα επικείμενα προβλήματα αλλά φροντίζουν και για τη μακροχρόνια ασφάλεια και σταθερότητα του συστήματος.

Στη συνέχεια, για να εξασφαλίσουν την ελαχιστοποίηση της εμφάνισης των ίδιων λαθών και στο μέλλον σχεδιάζουν επιπρόσθετα συστήματα που μπορούν να εντοπίζουν και να επιλύουν αυτομάτως παρόμοια ζητήματα με αυτά που έχουν αντιμετωπίσει οι ίδιοι οι προγραμματιστές στο παρελθόν. Επιπρόσθετα, με τη διαδικασία refactoring[11], κάνοντας δηλαδή μικρές αλλαγές εσωτερικά στο κώδικα χωρίς να

αλλάζουν τον πυρήνα, βελτιστοποιούν τη λειτουργικότητα και την αποτελεσματικότητα του συστήματος, κάνοντάς το ταυτόχρονα και πιο ανθεκτικό.



Εικόνα 2.2.4: Διαδικασία του code refactoring[12].

Τέλος, σε περίπτωση που μπορεί να προκύψει κάποιο ξαφνικό πρόβλημα στο σύστημα, είναι σημαντικό να γίνεται έγκαιρα η αποκατάστασή του ώστε να μην είναι υπάρξουν μεγάλες διακοπές λειτουργίας. Για αυτό είναι χρήσιμο μέσω της συνεχούς ανατροφοδότησης, οι προγραμματιστές να δημιουργούν προληπτικές λύσεις για τη συντήρηση των συστημάτων ώστε να προλαμβάνουν την εμφάνιση ίδιων η και όχι προβλημάτων.

2.2.5 Εκπαίδευση

Η τεχνολογία δε σταματάει να εξελίσσεται και νέες εξειδικευμένες τεχνικές συνεχίζουν να εμφανίζονται οπότε οι προγραμματιστές οφείλουν να ενημερώνονται. Αν θέλει μια επιχείρηση να παραμείνει ανταγωνιστική και καινοτόμα θα πρέπει να φροντίζει οι προγραμματιστές της να συνεχίσουν να εκπαιδεύονται παρακολουθώντας τις νέες τάσεις και ενσωματώνοντάς τες στη καθημερινή τους εργασία. Για παράδειγμα, στις μέρες μας έχει αποδειχθεί πολύ χρήσιμη η χρήση τεχνητής νοημοσύνης και machine learning στη διαδικασία παραγωγής, διόρθωσης και βελτιστοποίησης κώδικα. Η συνεχής εκπαίδευση του προγραμματιστή μπορεί να επιτευχθεί μέσω της συμμετοχής του σε ειδικά σεμινάρια και εργαστήρια, που θα το προσφέρουν νέες γνώσεις και εμπειρίες για να ενσωματώσει αργότερα στη δουλειά του[13].

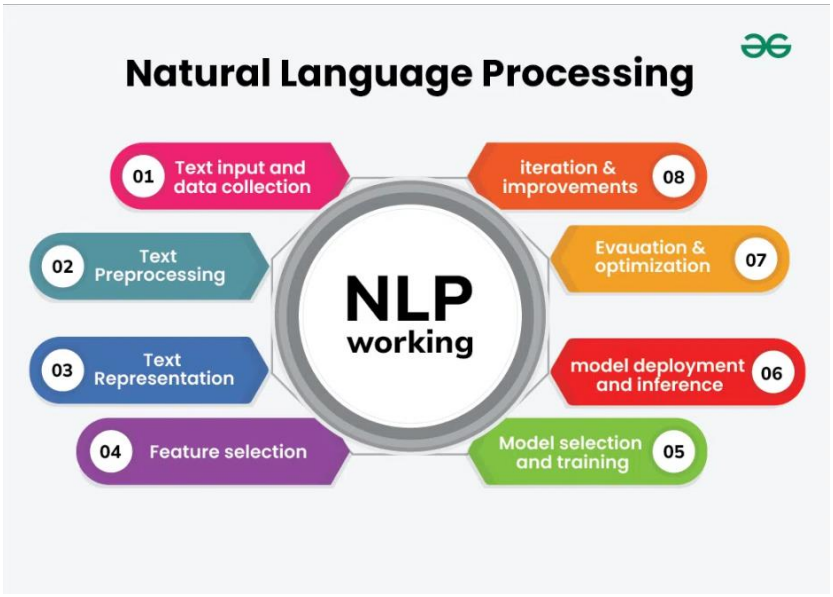
Ωστόσο είναι εξίσου σημαντικό, ο προγραμματιστής να προσπαθεί και προσωπικά να βελτιωθεί, είτε μέσω ατομικής του έρευνας είτε ανταλλάσσοντας ιδέες με άλλους προγραμματιστές εντός της εταιρείας[14]. Αυτό έχει ως αποτέλεσμα μέσω της αλληλοβοήθειας να ενισχύεται η συνολική ικανότητα της εταιρείας. Έτσι με τη συνεχή ενημέρωση για νέες τεχνολογίες και εφαρμογή καινοτόμων μεθόδων, οι προγραμματιστές φροντίζουν η επιχείρηση να παραμένει ανταγωνιστική και να μπορεί να ανταποκρίνεται στις ανάγκες της αγοράς που αλλάζουν συνεχώς.

ΚΕΦΑΛΑΙΟ 3

Τομέας NLP και Large Language Models

3.1 Εισαγωγή

Ο τομέας τεχνητής νοημοσύνης αποτελεί πλέον μέρος της καθημερινότητας των ανθρώπων είτε στο τομέα της πληροφορικής είτε όχι καθώς οι δυνατότητες που προσφέρει έχουν πολλές χρήσιμες εφαρμογές. Το NLP (Natural Language Processing) [15] είναι μεταξύ άλλων ένας σημαντικός και πρωτοποριακός κλάδος του τομέα τεχνητής νοημοσύνης. Το αντικείμενο του NLP είναι η ανάπτυξη τεχνολογιών και συστημάτων που επιτρέπουν στους υπολογιστές να επεξεργάζονται, να αναλύουν και να παράγουν δεδομένα γραμμένα σε ανθρώπινη γλώσσα. Συνεπώς, είναι πολύ χρήσιμο εργαλείο σε περιπτώσεις όπως η αυτόματη μετάφραση, τα Chatbot(πχ. ChatGPT της OpenAI) και ψηφιακών βοηθών (πχ. Alexa της Amazon). Η επιρροή της τεχνολογίας NLP στο τομέα ανάπτυξης λογισμικού είναι εξαιρετικά σημαντική, καθώς παρουσιάζει νέες δυνατότητες αυτοματοποίησης και βελτίωσης της παραγωγικότητας μέσω της συνεργασίας ανθρώπου και μηχανής.



Εικόνα 3.1: Εφαρμογές του NLP[16].

Στη βιομηχανία ανάπτυξης λογισμικού, υπάρχουν πολλά στάδια μέχρι να επιτευχθεί το τελικό αποτέλεσμα, όπως η σχεδίαση, η υλοποίηση, η συντήρηση και ο έλεγχος του λογισμικού. Τα στάδια αυτά πρέπει να χρειάζονται ακρίβεια και πολλές δοκιμές ώστε να μειωθούν τα λάθη που μπορεί να προκύψουν. Οπότε η τεχνολογία NLP μπορεί να συμβάλλει στις διαδικασίες αυτές διευκολύνοντας τη παραγωγή, τη διαχείριση και τη διόρθωση κώδικα, αλλά και την ανάλυση δεδομένων.

Στις μέρες μας, η πιο ευρέως διαδεδομένη εφαρμογή του NLP στο τομέα της πληροφορικής είναι η αυτόματη παραγωγή κώδικα. Αυτό γίνεται κάνοντας χρήση μεγάλων γλωσσικών μοντέλων βασισμένων στο NLP που προσφέρουν αυτή τη δυνατότητα όπως το ChatGPT της OpenAI, το Tabnine, το CodeT5 και άλλα. Οι προγραμματιστές μπορούν να περιγράψουν σε φυσική γλώσσα την εντολή που θέλουν να εκτελέσει το μοντέλο και ανάλογα τη λειτουργικότητα που έχουν ζητήσει μέσω της εντολής, θα δεχτούν τον αντίστοιχο κώδικα ως απάντηση. Η γρήγορη παραγωγή κώδικα από τα μοντέλα μειώνει κατά πολύ το χρόνο σχεδίασης λογισμικού ειδικά σε περιπτώσεις που χρειάζεται να γίνει επανάληψη κώδικα. Επιπρόσθετα, τα μοντέλα αυτά που χρησιμοποιούν τη τεχνολογία NLP, μπορούν να προσφέρουν τεκμηρίωση πάνω στον κώδικα[17], δημιουργώντας αυτόματα σχόλια σε κάθε βήμα κώδικα που εξηγούν τη λειτουργία του

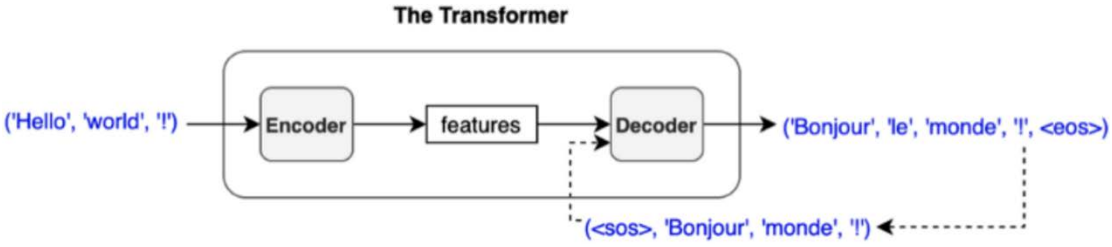
Μία άλλη εξίσου σημαντική εφαρμογή της τεχνολογίας NLP είναι ο εντοπισμός και η διόρθωση λαθών. Κάποια μεγάλα γλωσσικά μοντέλα όπως το ChatGPT και το Copilot έχουν την δυνατότητα να αναλύουν τον κώδικα και να εντοπίζουν λάθη είτε αυτά είναι λογικά είτε είναι συντακτικά, ενώ ταυτόχρονα προτείνουν λύσεις με τεκμηρίωση. Αυτή η δυνατότητα βοηθάει τη διαδικασία του debugging, προσφέροντας στους προγραμματιστές ένα επιπλέον εργαλείο για να βελτιώσουν την ποιότητα του κώδικά τους.

Επιπρόσθετα, πολλές εφαρμογές και ιστοσελίδες έχουν ήδη ενσωματώσει εικονικούς βοηθούς και Chatbots στο σύστημα τους, οι οποίοι είναι βασισμένοι στη τεχνολογία NLP[18]. Μέσω αυτών των βοηθών, οι χρήστες μπορούν να επικοινωνούν το πρόβλημά τους σε φυσική γλώσσα και να δέχονται απαντήσεις, κάνοντας έτσι τη χρήση των συστημάτων πιο προσιτή και φιλική ακόμη και για χρήστες που δεν έχουν τεχνικές γνώσεις.

Συνοψίζοντας, η τεχνολογία Natural Language Processing όταν ενσωματωθεί στη τεχνολογία λογισμικού, μπορεί να αποτελέσει σημαντικό εργαλείο για τον προγραμματιστή, διευκολύνοντας τη διαδικασία σχεδίασης λογισμικού. Με τη σημαντική μείωση του χρόνου και του κόστους ανάπτυξης του λογισμικού και την αύξηση της ποιότητάς του, συνδυάζοντας ταυτόχρονα τον άνθρωπο με τη μηχανή, είναι αναμενόμενο ότι στο μέλλον η ενσωμάτωσή του στο τομέα τεχνολογίας λογισμικού θα είναι μεγαλύτερη.

3.2 Large Language Models

Τα μεγάλα γλωσσικά μοντέλα (LLMs) είναι μοντέλα τεχνητής νοημοσύνης που έχουν εκπαιδευτεί από έναν μεγάλο όγκο δεδομένων, και είναι ικανά να καταλαβαίνουν και να παράγουν φυσική γλώσσα και άλλες πολλές εργασίες. Αυτά τα μοντέλα έχουν εφαρμογές και στον τομέα σχεδίασης λογισμικού, αφού προσφέρουν στους προγραμματιστές τη δυνατότητα να παράγουν και να διορθώνουν κώδικα αυτόματα. Κάποια από τα πιο γνωστά και αποτελεσματικά LLMs που χρησιμοποιούνται στο τομέα ανάπτυξης λογισμικού είναι τα GPT-4, Codex, CodeT5, Tabnine και CoGram[19], το καθένα με τις δικές του ιδιαιτερότητες και εφαρμογές.



Εικόνα 3.2: Μετάφραση ακολουθίας με κωδικοποιητή-αποκωδικοποιητή του μοντέλου Transformer[20].

3.2.1 GPT-4

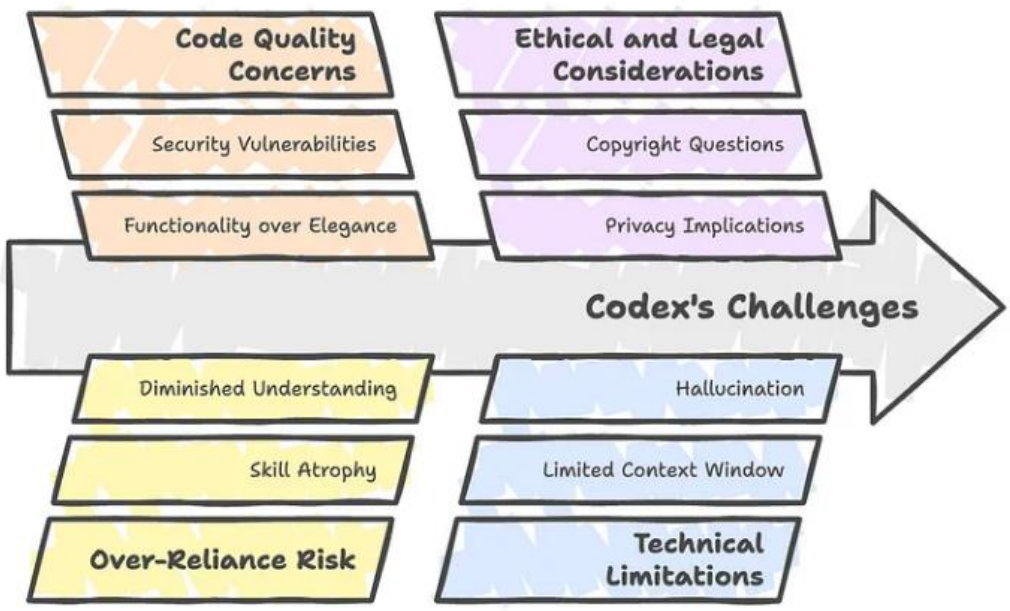
Το GPT-4 είναι η τέταρτη εκδοχή του GPT που αναπτύχθηκε από την OpenAI[21] και είναι ένα από τα πιο εξελιγμένα μεγάλα γλωσσικά μοντέλα που υπάρχουν στην εποχή μας. Το μοντέλο έχει εκπαιδευτεί χρησιμοποιώντας έναν τεράστιο όγκο δεδομένων που απέσπασε από δημόσιες πηγές στο διαδίκτυο, άρθρα, βιβλία, κώδικες και άλλα. Έχοντας τέτοια διεξοδική και εκτενή εκπαίδευση το GPT-4 μπορεί να κατανοεί εντολές και να παράγει απαντήσεις σε φυσική γλώσσα με μεγάλη ακρίβεια και ευκολία.

Η κύρια χρήση του GPT-4[22] που γίνεται από προγραμματιστές είναι η παραγωγή κώδικα έχοντας λάβει συγκεκριμένες εντολές σε φυσική γλώσσα. Δηλαδή, ο προγραμματιστής μπορεί να ζητήσει από το μοντέλο να του παράξει κώδικα για την ακολουθία Fibonacci και αυτό να του δώσει τον αντίστοιχο κώδικα στη προγραμματιστική γλώσσα που επιθυμεί. Ταυτόχρονα, το GPT-4 προσφέρει τη δυνατότητα εντοπισμού λαθών σε κώδικα, τεκμηριώνοντας τα βήματα και τέλος προτείνοντας λύσεις που θα βελτίωναν τον κώδικα. Ωστόσο οι δυνατότητές του είναι περιορισμένες καθώς σε πολύπλοκα προβλήματα μπορεί να παράξει ελαττωματικό κώδικα που δεν ανταποκρίνεται στη λειτουργία που του ζητήθηκε.

Εικόνα 3.2.1: Σύγκριση μοντέλων τεχνητής νοημοσύνης[23].

3.2.2 Codex

Ένα άλλο LLM-based εργαλείο τεχνητής νοημοσύνης είναι και το Codex από την ίδια εταιρεία OpenAI. Το Codex[24] είναι μια πιο εξειδικευμένη εκδοχή του GPT με κύριο σκοπό την δημιουργία κώδικα. Χρησιμοποιήθηκε σαν βάση για να δημιουργηθεί το GitHub Copilot, το οποίο είναι ενσωματωμένο σε IDEs όπως το Visual Studio Code σαν εργαλείο αυτόματης συμπλήρωσης κώδικα. Είναι πολύ χρήσιμο για τους προγραμματιστές καθώς μπορεί να δημιουργήσει κώδικα σε αρκετές γλώσσες όπως Java, JavaScript, Python, C# και άλλες. Επιπρόσθετα, το Codex έχει και αυτό την δυνατότητα να εντοπίσει σφάλματα στο κώδικα και να δώσεις λύσεις αλλά όπως και το GPT-4 έχει αδυναμία στην επίλυση πολύπλοκων λογικών προβλημάτων.

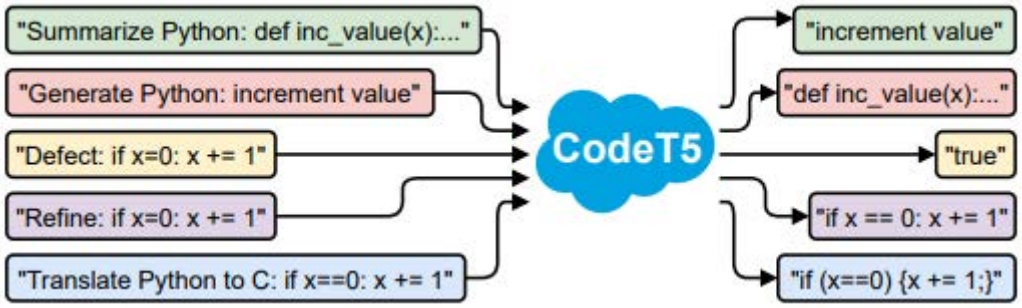


Εικόνα 3.2.2: Περιορισμοί στη χρήση του Codex[25].

3.2.3 CodeT5

Το CodeT5[26, 27] είναι ένα μοντέλο που αναπτύχθηκε από την Salesforce Research και επικεντρώνεται στο να κατανοεί, να παράγει και να αναδιαμορφώνει

κώδικα. Το μοντέλο βασίζεται στην αρχιτεκτονική T5, η οποία χρησιμοποιείται ευρέως σε εργασίες που σχετίζονται με το NLP. Αυτό που το ξεχωρίζει από άλλα μεγάλα γλωσσικά μοντέλα είναι ότι χρησιμοποιεί μια τεχνική που λέγεται identifier-aware pre-training για να διαβάσει κώδικα. Δηλαδή μπορεί να αναγνωρίσει κάποιους identifiers στο κώδικα όπως ονόματα μεταβλητών και συναρτήσεων, κατανοώντας έτσι καλύτερα τον κώδικα. Μπορεί, επίσης, να εντοπίσει και διορθώσει σφάλματα, να μεταφράσει κώδικα από μία γλώσσα σε μια άλλη, αλλά και να προσφέρει πλήρη περιγραφή του παραγόμενου κώδικα για να γίνει πιο κατανοητός. Ωστόσο, είναι εκπαιδευμένο σε λίγες προγραμματιστικές γλώσσες όπως C και C# οπότε δεν είναι τόσο αποδοτικό σε γλώσσες που δεν έχει εκπαιδευτεί.



Εικόνα 3.2.3: Παράδειγμα χρήσης του CodeT5 για συγγραφή κώδικα[28].

3.2.4 Tabnine

Το Tabnine[29, 30] λειτουργεί σαν AI code assistant , δηλαδή βοηθά τον προγραμματιστή δίνοντας προτάσεις κώδικα που ταιριάζουν στο περιβάλλον που γράφει ο χρήστης, μειώνοντας έτσι τον χρόνο γραφής. Επίσης, είναι πολύ χρήσιμο για εργασίες πάνω σε μεγάλες βάσεις κώδικα, καθώς προσφέρει γρήγορη ανάλυση των δεδομένων αλλά και αυτόματη συγγραφή κώδικα που επαναλαμβάνεται στο σύστημα. Το Tabnine χρησιμοποιείται ήδη από αρκετά περιβάλλοντα ανάπτυξης λογισμικού όπως το Visual Studio Code, το PyCharm και το Android Studio. Η ιδιαιτερότητα του Tabnine είναι ότι λειτουργεί τοπικά πάνω στο κάθε υπολογιστικό σύστημα διασφαλίζοντας έτσι ότι ο κώδικας δεν θα διαρρεύσει στο διαδίκτυο. Ωστόσο η λειτουργία του είναι κυρίως βοηθητική καθώς χρειάζεται την παρέμβαση του προγραμματιστή για την τελική συγγραφή του κώδικα.

3.2.5 CoGram

Το CoGram είναι ένα μοντέλο τεχνητής νοημοσύνης που μπορεί να παραγάγει κώδικα και χρησιμοποιείται κυρίως από προγραμματιστές που γράφουν σε γλώσσα Python με SQL queries. Το μοντέλο μπορεί να ενσωματωθεί σε IDEs όπως το Jupyter Notebooks[31] για να συμπληρώνει αυτόματα κώδικα και υποστηρίζει συστήματα SQLite, PostgreSQL και MySQL. Επίσης το CoGram είναι πολύ χρήσιμο σε επαγγελματικές διαδικτυακές συναντήσεις, καθώς μπορεί να παράγει σημειώσεις αυτόματα δίνοντας έμφαση σε σημαντικές πληροφορίες, ακούγοντας απλά την συνομιλία[32]. Βέβαια, φροντίζει για την ασφάλεια των πληροφοριών που άντλησε ώστε να μην διαρρεύσουν στο διαδίκτυο.

3.2.6 Llama3

Το Llama3 είναι το τελευταίο μοντέλο που έχει δημιουργήσει η Meta AI[33, 34] και είναι γνωστό για τις ικανότητές του στη παραγωγή και επεξήγηση σύνθετου κώδικα. Το μοντέλο αυτό είναι πολύ χρήσιμο και ιδανικό για μεγάλα και πολυσύνθετα project καθώς μπορεί να διαχειριστεί τεράστιες βάσεις δεδομένων. Εκτός από τη παραγωγή κώδικα λειτουργεί και σαν εργαλείο συμπλήρωσης κώδικα και υποστηρίζει πολλές γλώσσες προγραμματισμού όπως Python, C++, Java και άλλες. Επιπλέον, μπορεί να κατανοεί και να τεκμηριώνει πολύπλοκα προβλήματα κώδικα ενώ ταυτόχρονα έχει και δυνατότητες debugging. Είναι το κατάλληλο μοντέλο τεχνητής νοημοσύνης για project μεγάλης κλίμακας που χρειάζεται να επεξεργαστούν μεγάλες βάσεις κώδικα.

3.2.7 DeepSeek-R1

Ένα εξίσου σημαντικό LLM ,που δημιουργήθηκε από τη κινέζικη εταιρεία DeepSeek, είναι το DeepSeek-R1[35] το οποίο είναι το τελευταίο μοντέλο της εταιρείας. Έχει πολλές εφαρμογές σε διάφορους κλάδους αλλά στο τομέα τεχνολογίας λογισμικού επιτυγχάνει τα ίδια αποτελέσματα απόδοσης με τα μοντέλα AI της OpenAI,

μειώνοντας όμως το κόστος εκπαίδευσης και χρήσης του μοντέλου. Μπορεί να ανταπεξέλθει στην σύνθεση πολύπλοκων προβλημάτων κώδικα αλλά και να επεξεργαστεί τεράστιο όγκο δεδομένων σε πολύ μικρό χρόνο. Επιπλέον ο κώδικας του DeepSeek είναι open source, δηλαδή δίνεται πρόσβαση της βάσης του κώδικα στο ευρύτερο κοινό.

Feature	DeepSeek-R1	GPT-o1
Architecture	Mixture-of-Experts (MoE), 671B params	Dense Transformer, 175B params
Efficiency	Activates only 37B params per token	Uses all parameters per task
Training Cost	Lower GPU hours, cost-effective	Higher GPU demand, proprietary
Strengths	Coding, structured reasoning, logic	Multi-step reasoning, creativity
Customization	Open-source, full control	Proprietary, limited customization
Best For	Developers, technical tasks	Research, creative & analytical tasks

Εικόνα 3.2.7: Σύγκριση του DeepSeek-R1 και του GPT-o1[36].

3.2.8 Συμπέρασμα

Η εξέλιξη των μεγάλων γλωσσικών μοντέλων και η δημιουργία LLM-based μοντέλων σε συνδυασμό με τη χρήση τεχνητής νοημοσύνης έχει προσφέρει νέες ευκαιρίες και εφαρμογές στη τεχνολογία λογισμικού. Όλα τα παραπάνω μοντέλα και άλλα που δεν αναφέραμε, συμβάλλουν σημαντικά στην διευκόλυνση της διαδικασίας σχεδίασης λογισμικού μειώνοντας το χρόνο παραγωγής και βελτιώνοντας τη αποδοτικότητά του. Συνεπώς, μελλοντικά όσο εξελίσσονται και εκπαιδεύονται περισσότερα αυτά τα μοντέλα τόσο θα βελτιώνεται η ποιότητα και η αξιοπιστία του κώδικα που βοηθάνε να δημιουργηθεί.

3.3 Δυνατότητες των Large Language Models

Όπως είδαμε και παραπάνω τα μεγάλα γλωσσικά μοντέλα έχουν πολλές εφαρμογές στη τεχνολογία λογισμικού και προσφέρουν τέτοιες δυνατότητες που τα καθιστούν σημαντικά εργαλεία για κάθε προγραμματιστή. Είτε με τη παραγωγή κώδικα είτε με την ανίχνευση σφαλμάτων και την επεξήγηση του κώδικα που παράγουν, τα LLMs δίνουν προτάσεις για τη βελτίωση της αποδοτικότητας και της τελικής ποιότητας του λογισμικού.

3.3.1 Παραγωγή κώδικα

Η κυριότερη και η πιο ωφέλιμη δυνατότητα που προσφέρουν τα LLMs είναι η αυτόματη παραγωγή κώδικα όταν δεχτούν μια εντολή σε φυσική γλώσσα που περιγράφει τι επιθυμεί ο χρήστης[37].Τα περισσότερα μεγάλα γλωσσικά μοντέλα έχουν εκπαιδευτεί από τεράστια μάζα δεδομένων του διαδικτύου που περιλαμβάνουν και αμέτρητα παραδείγματα κώδικα, συνεπώς μπορούν να παράγουν μέσα σε λίγα δευτερόλεπτα έτοιμο κώδικα που αντιστοιχεί στο prompt του χρήστη. Για παράδειγμα, αν ένας χρήστης χρειάζεται να υλοποιήσει την ακολουθία Fibonacci, μπορεί να δώσει την ίδια εντολή γραμμένη σε φυσική γλώσσα σε ένα μοντέλο όπως το ChatGPT[38, 39], και αυτό θα του επιστρέψει αμέσως έτοιμη υλοποίηση ενώ ταυτόχρονα θα προσθέσει σχόλια και τεκμηρίωση σε κάθε βήμα του κώδικα εξηγώντας τη λειτουργία του. Με αυτό το τρόπο ο προγραμματιστής μειώνει τον χρόνο που χρειάζεται για τη συγγραφή του αρχικού κώδικα, πριν προβεί σε διορθώσεις και διαμορφώσεις που θα βελτιστοποιήσουν την αποτελεσματικότητά του.[40, 41]

Επιπλέον, τα LLMs δεν περιορίζονται στη συγγραφή κώδικα σε μία μόνο γλώσσα προγραμματισμού. Έχουν εκπαιδευτεί σε μεγάλο εύρος γλωσσών οπότε μπορούν να προσαρμόζουν τον κώδικα ανάλογα τη γλώσσα που επιθυμεί ο χρήστης αλλά και να μεταφράζουν κώδικα από μία γλώσσα προγραμματισμού σε άλλη χωρίς να χάνετε η λειτουργικότητά του[42, 43]. Ωστόσο, όπως είναι λογικό ο κώδικας που παράγουν τα μεγάλα γλωσσικά μοντέλα δεν είναι πάντα αξιόπιστα καθώς ενδέχεται να έχουν συντακτικά και λογικά λάθη, ειδικά αν τους ζητηθεί να υλοποιηθεί ένα πιο

σύνθετο και εξειδικευμένο πρόβλημα. Συνεπώς, η παρέμβαση του προγραμματιστή είναι απαραίτητη ώστε το τελικό αποτέλεσμα να είναι ένας σωστά δομημένος και λειτουργικός κώδικας.

3.3.2 Debugging

Μια άλλη εξίσου σημαντική λειτουργία των LLMs είναι και σαν εργαλείο debugging[44, 45, 46] για τους προγραμματιστές. Τα μοντέλα μπορούν να εντοπίσουν και να διορθώσουν σφάλματα στον κώδικα που θα τους δοθεί ή που θα έχουν παραγάγει τα ίδια, είτε αυτά τα σφάλματα είναι συντακτικά είτε λογικά. Δηλαδή, αν τους δοθεί ένα κομμάτι κώδικα σαν prompt από τον χρήστη και τους ζητηθεί να αναγνωρίσει αν υπάρχουν σφάλματα, το μοντέλο θα αναλύσει τον επικείμενο κώδικα και αφού υπογραμμίσει που βρίσκονται τα σφάλματα, θα προτείνει πιθανές λύσεις.

Επιπρόσθετα, τα LLMs εφόσον εντοπίσουν τα λάθη έχουν την δυνατότητα να επεξηγήσουν πώς αυτά τα λάθη επηρεάζουν τη λειτουργία του κώδικα[47, 48]. Αυτή η λειτουργία είναι πολύ χρήσιμη για έμπειρους και μη προγραμματιστές καθώς τους βοηθά να κατανοήσουν τις λεπτομέρειες της διαδικασίες του debugging σε πολύ μικρό χρόνο. Από την άλλη, όπως και η παραγωγή του κώδικα από LLMs έτσι και τα αποτελέσματα του debugging σε πολύπλοκες δομές κώδικα δεν είναι πάντα ακριβή και οι λύσεις που προτείνουν τα μοντέλα δεν είναι κατάλληλες για τη λειτουργία που επιθυμεί ο χρήστης.

3.3.3 Επεξήγηση κώδικα

Η προσθήκη επεξήγησης μέσω σχολίων στον κώδικα είναι μια χρονοβόρα διαδικασία αλλά απαραίτητη γιατί βοηθά στη κατανόηση των σταδίων του κώδικα από τρίτους ή όταν χρειαστεί ο δημιουργός να επισκεφτεί ξανά κάποιο σημείο για να το επεξεργαστεί[49]. Τα LLMs προσφέρουν αυτή την δυνατότητα της αυτόματης τεκμηρίωσης κώδικα, εξοικονομώντας έτσι χρόνο στους προγραμματιστές. Επίσης, μπορούν να δημιουργήσουν περιλήψεις[50, 51, 52] και τεχνικές αναφορές για μεγάλες βάσεις κώδικα αλλά και να προτείνουν βελτιώσεις της τεκμηρίωσης και της

αναγνωσιμότητας του κώδικα, διαμορφώνοντας τα σχόλια και την ονοματολογία των μεταβλητών και των συναρτήσεων. Πάντα όμως θα χρειάζεται η εποπτεία από τους προγραμματιστές, καθώς η αξιοπιστία και η ακρίβεια της επεξήγησης που παράγει το μοντέλο εξαρτώνται κυρίως από την ποιότητα του κώδικα που του δόθηκε και την εκπαίδευση του μοντέλου.

3.4 Περιορισμοί και προβλήματα των LLMs

Έχοντας δει τις δυνατότητες που προσφέρουν τα μεγάλα γλωσσικά μοντέλα όπως το GPT-4, Codex και άλλα, είναι εύλογο να αναρωτηθεί κάποιος και για αρνητικά της χρήσης τέτοιων μοντέλων. Οπότε πριν ενσωματωθούν πλήρως στο τομέα τεχνολογίας λογισμικού, οι προγραμματιστές πρέπει να λάβουν υπόψη τους περιορισμούς και τα ζητήματα που προκύπτουν όπως η μεροληψία των μοντέλων στη παραγωγή απαντήσεων, η αδυναμία επίλυσης πολύπλοκων προβλημάτων αλλά και τα λάθη που μπορεί να κάνουν πάνω στον κώδικα που παράγουν.

3.4.1 Μεροληψία

Αρχικά, τα περισσότερα LLMs έχουν εκπαιδευτεί και συνεχίζουν να εκπαιδεύονται από τεράστια ποσότητα δεδομένων του διαδικτύου όπως βιβλία, άρθρα, παραδείγματα κώδικα, επιστημονικές πηγές και άλλα. Συνεπώς, οι απαντήσεις που προσφέρουν δεν είναι αμερόληπτες[53] αφού και τα δεδομένα από τα οποία έχουν εκπαιδευτεί μπορεί να παρουσιάσουν ανακρίβειες και προκαταλήψεις. Δηλαδή, όταν τα δεδομένα εκπαίδευσης του LLM είναι περιορισμένα ή όχι ενημερωμένα στις απαιτήσεις του σήμερα, τότε οι λύσεις που θα προτείνει θα βασίζονται παλιότερες πρακτικές και όχι στα βέλτιστα πρότυπα σχεδίασης λογισμικού.

Για να αντιμετωπιστεί αυτό το πρόβλημα, χρειάζεται να ελεγχθούν και να διαμορφωθούν τεράστια σύνολα δεδομένων που χρησιμοποιούνται για την εκπαίδευση των μεγάλων γλωσσικών μοντέλων. Έτσι, είναι απαραίτητη η παρέμβαση των προγραμματιστών για να εντοπίζουν πότε οι απαντήσεις που δίνει ένα μοντέλο δεν

ακολουθούν τις κατάλληλες τεχνικές ανάπτυξης λογισμικού ή εμφανίζουν σημεία μεροληψίας.

3.4.2 Αδυναμία σε σύνθετα προβλήματα

Όπως θα δείξουμε και στη συνέχεια τα μεγάλα γλωσσικά μοντέλα ενώ είναι ικανά να παράγουν κώδικα και να διορθώσουν σφάλματα, δυσκολεύονται όταν πρόκειται για πιο σύνθετα ή εξειδικευμένα προβλήματα. Αυτό συμβαίνει διότι τα μοντέλα αυτά έχουν εκπαιδευτεί από ήδη υπάρχοντα δεδομένα του διαδικτύου και δεν έχουν την ικανότητα να κατανοήσουν πραγματικά τη λογική και τις μαθηματικές πράξεις που ίσως απαιτούνται για την επίλυση πολύπλοκων προβλημάτων. Δηλαδή, όταν τους ζητηθεί να υλοποιήσουν έναν αλγόριθμο που απαιτεί βαθιά κατανόηση άλγεβρας και γεωμετρίας, τότε το μοντέλο μπορεί να δώσει μια λύση η οποία μπορεί να φαίνεται συντακτικά σωστή αλλά να έχει θεμελιώδη λογικά σφάλματα.

Αυτή η συμπεριφορά των LLMs είναι αναμενόμενη καθώς οι ικανότητές τους εξαρτώνται από την μάζα πληροφοριών που χρησιμοποιήθηκε για την εκπαίδευσή τους. Συνεπώς, όταν πρόκειται για προβλήματα που χρειάζονται καινοτόμες και σύνθετες λύσεις, τα μοντέλα αυτά αδυνατούν να ανταπεξέλθουν καθώς συνήθως επαναλαμβάνουν υπάρχουσες λύσεις χωρίς να αναλύουν και να κατανοούν την μοναδικότητα και την φύση του κάθε προβλήματος ξεχωριστά.

3.4.3 Αστοχίες και λάθη

Σε όλα τα λογισμικά μπορεί να εμφανιστούν λάθη, τα λεγόμενα bugs, έτσι και τα LLMs αν και είναι εξαιρετικά αποτελεσματικά στη γρήγορη παραγωγή κώδικα, σε κάποιες περιπτώσεις έχουν και αυτά αστοχίες στις λύσεις που προτείνουν. Συνήθως τα σφάλματα αυτά είναι συντακτικά, λογικά, μη ολοκληρωμένος κώδικας, αλλά και λανθασμένη κατανόηση του προβλήματος και της λειτουργίας που του ζητήθηκε να υλοποιήσει[54, 55]. Επίσης, μπορεί τα LLMs όταν δεν διαθέτουν επαρκή δεδομένα να επινοήσουν πληροφορίες που θα τα οδηγήσουν να κατασκευάσουν μία φαινομενικά ορθή απάντηση που στη πραγματικότητα είναι αναληθής.

Αυτή η συμπεριφορά των LLMs είναι γνωστή ως “hallucination” και μπορεί να εμφανιστεί για διάφορους λόγους[56]. Αρχικά, μπορεί η βάση δεδομένων πάνω στην οποία εκπαιδεύτηκε το μεγάλο γλωσσικό μοντέλο να είναι ημιτελή, να περιέχει μεροληπτικό περιεχόμενο ή και παραπληροφόρηση, κατά συνέπεια οι απαντήσεις του μοντέλου να μην είναι αξιόπιστες. Επίσης, αν του δοθεί από τον προγραμματιστή μία εντολή που δεν περιγράφει λεπτομερώς το πρόβλημα, τότε το μοντέλο θα προσπαθήσει να υποθέσει τι χρειάζεται ο προγραμματιστής και θα του δώσει την ανάλογη απάντηση η οποία θα είναι σαφώς λανθασμένη όσον αφορά τη λειτουργία της. Ακόμη, τα LLMs συνήθως αποτυγχάνουν να όταν έρχονται αντιμέτωπα με εργασίες που απαιτούν ανάλυση μεγάλου εύρους δεδομένων ή όταν χρειάζεται να διατηρήσουν τη συνεκτικότητα ενός κώδικα που αποτελείται από πολλές σειρές, συνεπώς καταλήγουν να σε ανακρίβειες ή και παραλείψεις.

3.5 Συμπέρασμα

Εν κατακλείδι, τα μεγάλα γλωσσικά μοντέλα αποτελούν σημαντικό εργαλείο για τους προγραμματιστές, συμβάλλοντας κυρίως στη μείωση του χρόνου ανάπτυξης ενός λογισμικού, με τις δυνατότητες που προσφέρουν όπως η παραγωγή και αυτόματη συμπλήρωση κώδικα, ο εντοπισμός και η διόρθωση σφαλμάτων αλλά και η τεκμηρίωση που προσφέρουν όταν τους ζητηθεί. Από την άλλη, φαίνεται πως δεν θα έπρεπε να γίνεται τυφλή χρήση των μοντέλων καθώς δεν είναι πάντοτε αξιόπιστα λόγω των περιορισμών που είδαμε όπως η μεροληψία των δεδομένων εκπαίδευσης που χρησιμοποιήθηκαν, η αδυναμία τους να επιλύσουν πολύπλοκα προβλήματα που απαιτούν σύγχρονες λύσεις και οι αστοχίες που μπορεί να παρουσιάσουν σε ειδικές περιπτώσεις. Για αυτό, είναι απαραίτητη η ανθρώπινη παρέμβαση ώστε να ελέγχονται και να αντιμετωπίζονται τα προβλήματα που προκύπτουν και τέλος να αξιοποιούνται στο έπακρο οι δυνατότητές τους στη τεχνολογία λογισμικού. Με τη συνεχή εκπαίδευση και βελτίωση των μοντέλων θα επιτευχθεί η αύξηση της αξιοπιστίας τους και θα αποφευχθούν μελλοντικές αστοχίες, δίνοντας έτσι ελπίδα ότι στο μέλλον θα γίνει μια πιο ολοκληρωμένη και αποδοτική ενσωμάτωση των μεγάλων γλωσσικών μοντέλων στο τομέα ανάπτυξης λογισμικού.

ΚΕΦΑΛΑΙΟ 4

ΣΥΓΚΡΙΣΗ ΑΝΘΡΩΠΟΥ ΚΑΙ ΜΗΧΑΝΗΣ

4.1 Εισαγωγή

Η διαδικασία ανάπτυξης λογισμικού έχει αρχίσει να επαναπροσδιορίζεται με την ένταξη των μεγάλων γλωσσικών μοντέλων στο τομέα τεχνολογίας λογισμικού. Παλιότερα έπρεπε ο προγραμματιστής να χρησιμοποιήσει τη λογική, τη δημιουργικότητα και τις τεχνολογικές του ικανότητες για τη σχεδίαση ενός λογισμικού, ενώ τώρα έχει στην διάθεσή του εργαλεία όπως τα LLMs, τα οποία προσφέρουν τη δυνατότητα παραγωγής και διόρθωσης κώδικα σε πολύ λιγότερο χρόνο. Σε αυτό το κεφάλαιο θα δοκιμάσουμε τη δυνατότητα των μεγάλων γλωσσικών μοντέλων στην παραγωγή κώδικα για διάφορα προβλήματα αυξανόμενης πολυπλοκότητας. Ύστερα θα συγκρίνουμε τη ποιότητα του κώδικα που παρήγαγε το LLM με αυτή ενός ανθρώπου προγραμματιστή, δίνοντας έμφαση στους εξής παράγοντες. Το χρόνο γραφής και τη ταχύτητα υλοποίησης του κώδικα, την αποτελεσματικότητα και την ακρίβεια της λύσης, την ευκρίνεια και τεκμηρίωση, και τέλος τη συχνότητα εμφάνισης σφαλμάτων.

4.2 Επίλυση προβλημάτων με κώδικα από LLM

Κάνοντας χρήση ενός από το πολλά large language models που μπορούν να παράξουν κώδικα, το ChatGPT, θα δούμε τις δυνατότητές του να λύσει κάποια προβλήματα που θα του δοθούν ως εντολή. Τα προβλήματα που επιλέξαμε να ζητήσουμε από το ChatGPT να λύσει, είναι αυξανόμενης δυσκολίας και έχουν τις εξής ιδιαιτερότητες. Αρχικά η ταξινόμηση ενός δισδιάστατου πίνακα χρειάζεται σχετικά λίγα βήματα υλοποίησης και είναι εύκολα κατανοητή η λογική του προβλήματος. Στη συνέχεια, η συμπίεση Huffman έχει εξίσου σαφή λογική αλλά για να λυθεί χρειάζεται

περισσότερα βήματα κώδικα και σωστή χρήση κόμβων. Το πρόβλημα minimum spanning tree απαιτεί τη κατανόηση σύνθετων δομών δεδομένων, αφού είναι απαραίτητη η χρήση γραφημάτων και αλγορίθμων όπως Kruskal ή Prim. Ύστερα, το πρόβλημα εύρεσης της συντομότερης διαδρομής Dijkstra συνδυάζει λίγο πιο περίπλοκη κατανόηση της υλοποίησης του προβλήματος με τη χρήση ουρών προτεραιότητας. Ο υπολογισμός της μέγιστης ροής είναι πολύπλοκο πρόβλημα και χρειάζεται αρκετές γραμμές κώδικα για να υλοποιηθεί, χρησιμοποιώντας δυναμικά γραφήματα, αναδρομικές αναζητήσεις διαδρομών και διαχείριση βρόχων και ακμών. Τέλος, το πρόβλημα με την τριγωνοποίηση πολυγώνου ελέγχουμε αν το LLM μπορεί να κατανοήσει μαθηματικές έννοιες και να εφαρμόσει με ακρίβεια τους αντίστοιχους γεωμετρικούς ελέγχους.

4.2.1 Ταξινόμηση δισδιάστατου Πίνακα

Εντολή: Να ταξινομηθεί ένας δισδιάστατος πίνακας αριθμών με αύξουσα σειρά

Λύση:

Κώδικας:

python Αντιγραφή Επεξεργασία

```python
def sort_2d_array(matrix):
    flat_list = [item for row in matrix for item in row]
    flat_list.sort()
    rows = len(matrix)
    cols = len(matrix[0]) if rows > 0 else 0
    sorted_matrix = [flat_list[i * cols:(i + 1) * cols] for i in range(rows)]
    return sorted_matrix

matrix = [
    [3, 8, 7],
    [5, 2, 9],
    [4, 6, 1]
]

sorted_matrix = sort_2d_array(matrix)
print("Ταξινομημένος πίνακας:", sorted_matrix)
```

Output κώδικα:

```
Αρχικός πίνακας: [[3, 8, 7], [5, 2, 9], [4, 6, 1]]
Ταξινομημένος πίνακας: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

4.2.2 Συμπίεση Huffman

Εντολή: Να συμπιεσθεί το κείμενο με τη χρήση του αλγορίθμου Huffman

Λύση:

python Αντιγραφή Επεξεργασία

```python
import heapq
from collections import defaultdict

class Node:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq

def build_huffman_tree(text):
    frequency = defaultdict(int)
    for char in text:
        frequency[char] += 1
    priority_queue = [Node(char, freq) for char, freq in frequency.items()]
    heapq.heapify(priority_queue)
    while len(priority_queue) > 1:
        left = heapq.heappop(priority_queue)
        right = heapq.heappop(priority_queue)
        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(priority_queue, merged)
    return priority_queue[0]
```

```python
def build_huffman_codes(root, current_code="", codes={}):
    if root is None:
        return
    if root.char is not None:
        codes[root.char] = current_code
    build_huffman_codes(root.left, current_code + "0", codes)
    build_huffman_codes(root.right, current_code + "1", codes)
    return codes

def huffman_encode(text):
    root = build_huffman_tree(text)
    codes = build_huffman_codes(root)
    encoded_text = "".join(codes[char] for char in text)
    return encoded_text, codes

def huffman_decode(encoded_text, codes):
    reverse_codes = {v: k for k, v in codes.items()}
    decoded_text = ""
    current_code = ""
    for bit in encoded_text:
        current_code += bit
        if current_code in reverse_codes:
            decoded_text += reverse_codes[current_code]
            current_code = ""
    return decoded_text
```

```python
# Παράδειγμα χρήσης
text = "hello huffman"
encoded_text, codes = huffman_encode(text)
decoded_text = huffman_decode(encoded_text, codes)

print("Αρχικό Κείμενο:", text)
print("Κωδικοποιημένο Κείμενο:", encoded_text)
print("Λεξικό Κωδικοποίησης:", codes)
print("Αποκωδικοποιημένο Κείμενο:", decoded_text)
```

Output κώδικα:

```
Αρχικό Κείμενο: hello huffman
Κωδικοποιημένο Κείμενο: 001111100100101011100011000110110101011101
Λεξικό Κωδικοποίησης: {'h': '00', 'm': '010', 'f': '011', 'l': '100', 'o': '1010', 'a': '1011', 'u': '1100', 'n': '1101', ' ': '1110', 'e': '1111'}
Αποκωδικοποιημένο Κείμενο: hello huffman
```

4.2.3 Εύρεση Minimum Spanning Tree (MST)

Εντολή: Να βρεθεί το minimum spanning tree

Λύση:

```python
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.edges = []

    def add_edge(self, u, v, weight):
        self.edges.append((weight, u, v))

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1
```

```
    def kruskal_mst(self):
        self.edges.sort()
        parent = [i for i in range(self.vertices)]
        rank = [0] * self.vertices
        mst = []
        for weight, u, v in self.edges:
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                mst.append((u, v, weight))
                self.union(parent, rank, x, y)
        return mst

# Παράδειγμα χρήσης
g = Graph(4)
g.add_edge(0, 1, 10)
g.add_edge(0, 2, 6)
g.add_edge(0, 3, 5)
g.add_edge(1, 3, 15)
g.add_edge(2, 3, 4)

mst = g.kruskal_mst()
print("Ακμές του MST:", mst)
```

Output:

```
Ακμές του MST: [(2, 3, 4), (0, 3, 5), (0, 1, 10)]
```

4.2.4 Εύρεση συντομότερης διαδρομής (Dijkstra)

Εντολή: Να υπολογιστεί η συντομότερη διαδρομή από την πηγή στους υπόλοιπους κόμβους.

Λύση:

```python
import heapq

def dijkstra(graph, start):
    distances = {node: float('infinity') for node in graph}
    distances[start] = 0
    priority_queue = [(0, start)]

    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)
        if current_distance > distances[current_node]:
            continue
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))
    return distances

# Παράδειγμα χρήσης
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 5},
    'C': {'A': 4, 'B': 2, 'D': 1},
    'D': {'B': 5, 'C': 1}
}

start_node = 'A'
shortest_paths = dijkstra(graph, start_node)
print("Αποστάσεις από τον κόμβο-πηγή:", shortest_paths)
```

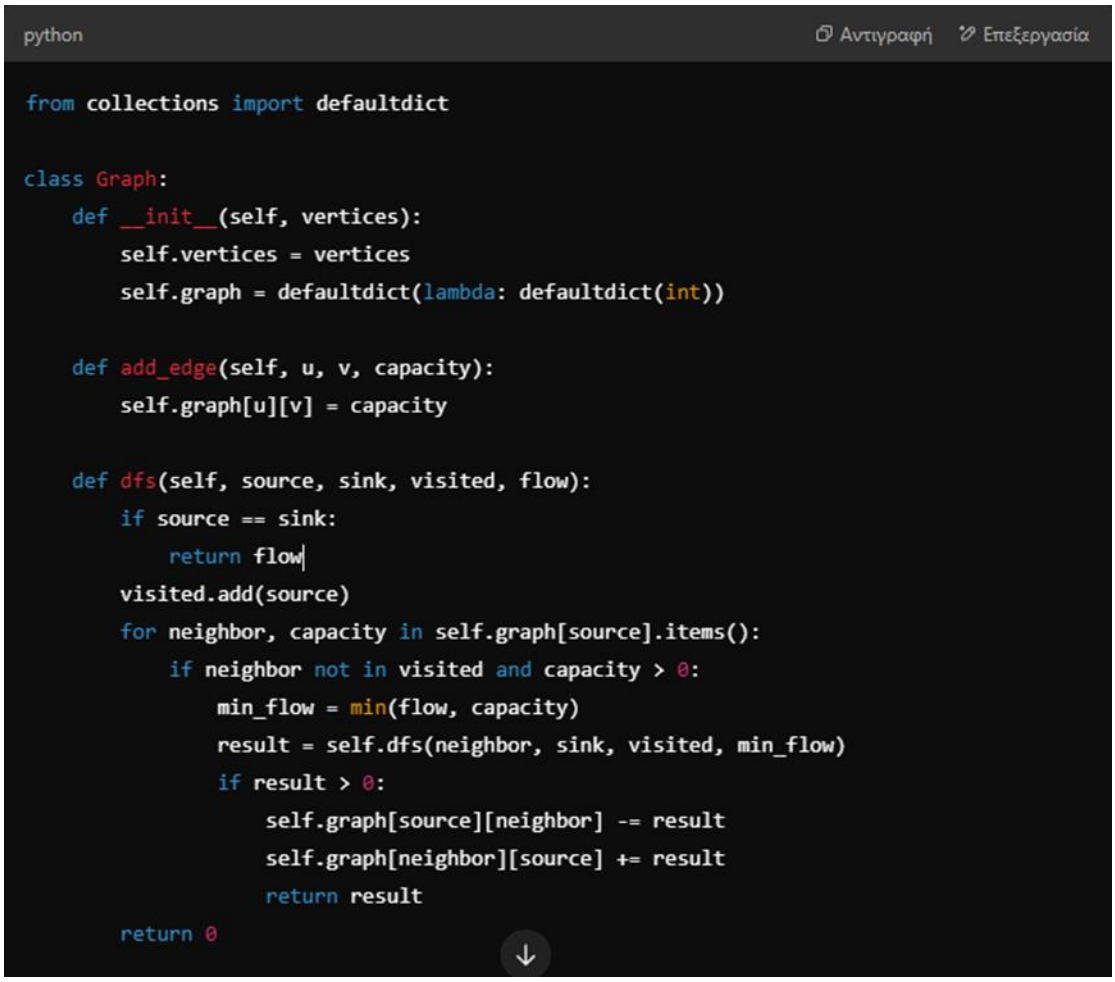
Output:

```
Αποστάσεις από τον κόμβο-πηγή: {'A': 0, 'B': 1, 'C': 3, 'D': 4}
```

4.2.5 Υπολογισμός Max Flow

Εντολή: Να υπολογιστεί η μέγιστη ροή από έναν κόμβο-πηγή σε έναν κόμβο-κατάληξη.

Λύση:



python Αντιγραφή Επεξεργασία

```python
from collections import defaultdict

class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = defaultdict(lambda: defaultdict(int))

    def add_edge(self, u, v, capacity):
        self.graph[u][v] = capacity

    def dfs(self, source, sink, visited, flow):
        if source == sink:
            return flow
        visited.add(source)
        for neighbor, capacity in self.graph[source].items():
            if neighbor not in visited and capacity > 0:
                min_flow = min(flow, capacity)
                result = self.dfs(neighbor, sink, visited, min_flow)
                if result > 0:
                    self.graph[source][neighbor] -= result
                    self.graph[neighbor][source] += result
                    return result
        return 0
```

```
    def ford_fulkerson(self, source, sink):
        max_flow = 0
        while True:
            visited = set()
            flow = self.dfs(source, sink, visited, float('Inf'))
            if flow == 0:
                break
            max_flow += flow
        return max_flow

# Παράδειγμα χρήσης
g = Graph(6)
g.add_edge(0, 1, 16)
g.add_edge(0, 2, 13)
g.add_edge(1, 2, 10)
g.add_edge(1, 3, 12)
g.add_edge(2, 4, 14)
g.add_edge(3, 5, 20)
g.add_edge(4, 3, 7)
g.add_edge(4, 5, 4)

source, sink = 0, 5
max_flow = g.ford_fulkerson(source, sink)
print(f"Μέγιστη ροή: {max_flow}")
```

Output:

```
Μέγιστη ροή: 23
```

4.2.6 Polygon Triangulation

Εντολή: Να υλοποιηθεί ο αλγόριθμος για το Polygon Triangulation.

Λύση:

python Αντιγραφή Επεξεργασία

```python
def is_convex(p1, p2, p3):
    return (p2[0] - p1[0]) * (p3[1] - p1[1]) - (p2[1] - p1[1]) * (p3[0] - p1[0]) > 0

def is_point_in_triangle(pt, v1, v2, v3):
    def sign(p1, p2, p3):
        return (p1[0] - p3[0]) * (p2[1] - p3[1]) - (p2[0] - p3[0]) * (p1[1] - p3[1])
    b1 = sign(pt, v1, v2) < 0.0
    b2 = sign(pt, v2, v3) < 0.0
    b3 = sign(pt, v3, v1) < 0.0
    return b1 == b2 == b3

def is_ear(p1, p2, p3, polygon):
    if not is_convex(p1, p2, p3):
        return False
    for point in polygon:
        if point != p1 and point != p2 and point != p3:
            if is_point_in_triangle(point, p1, p2, p3):
                return False
    return True
```

```python
def ear_clipping_triangulation(polygon):
    triangles = []
    polygon = polygon[:]
    while len(polygon) > 3:
        for i in range(len(polygon)):
            p1 = polygon[i]
            p2 = polygon[(i + 1) % len(polygon)]
            p3 = polygon[(i + 2) % len(polygon)]
            if is_ear(p1, p2, p3, polygon):
                triangles.append((p1, p2, p3))
                del polygon[(i + 1) % len(polygon)]
                break
    triangles.append((polygon[0], polygon[1], polygon[2]))
    return triangles

# Παράδειγμα χρήσης
polygon = [(0, 0), (4, 0), (4, 3), (2, 5), (0, 3)]
triangles = ear_clipping_triangulation(polygon)
print("Τρίγωνα του τριγωνισμού:", triangles)
```

Output:

```
Τρίγωνα του τριγωνισμού: [((0, 0), (4, 0), (4, 3)), ((0, 0), (4, 3), (2, 5)), ((0, 0), (2, 5), (0, 3))]
```

4.3 Σύγκριση κώδικα μεταξύ προγραμματιστή και LLM

Ο κώδικας που παράγει ένα Large Language Model διαφέρει με τον κώδικα που είναι γραμμένος από έναν άνθρωπο σε διάφορα σημεία ανάλογα την πολυπλοκότητα του προβλήματος που λύνει. Αυτό μπορούμε να το δούμε παρακάτω συγκρίνοντας κάποια από τα παραπάνω παραδείγματα κώδικα από το ChatGPT με κώδικα γραμμένο από εμένα.

4.3.1 Ταξινόμηση Δισδιάστατου Πίνακα

Ανθρώπινος Κώδικας:

```
def sorting(matrix, n) :
    k = 0
    tempMatrix = [0] * (n*n)

    for i in range (0 , n):
        for j in range(0, n):
            tempMatrix[k] = matrix[i][j]
            k = k + 1

    tempMatrix.sort()
    k = 0

    for i in range(0, n):
        for j in range (0, n):
            matrix[i][j] = tempMatrix[k]
            k = k + 1

def printMatrix(matrix, n):

    for i in range(0, n):
        for j in range(0, n):
            print(matrix[i][j], end = " ")
        print(" ")

matrix = [[3, 8, 7],
          [5, 2, 9],
          [4, 6, 1]]
n = 3
print("Αρχικός Πίνακας:")
printMatrix(matrix, n)
sorting(matrix, n)
print("Τελικός Πίνακας")
printMatrix(matrix, n)
```

4.4 Κριτήρια σύγκρισης

Τα κριτήρια σύγκρισης του κώδικα που γράφει ένας άνθρωπος με τον κώδικα που παράγει ένα Large Language Model όπως το ChatGPT στη περίπτωσή μας, είναι ο χρόνος γραφής, η ταχύτητα υλοποίησης, η αποτελεσματικότητα, η ευκρίνεια και τεκμηρίωση, τα σφάλματα και οι περιορισμοί του κώδικα.

4.4.1 Χρόνος γραφής

Ο χρόνος που χρειάστηκα για να γράψω τον κώδικα υλοποίησης της ταξινόμησης ενός δισδιάστατου πίνακα ήταν περίπου 15 λεπτά συμπεριλαμβάνοντας τον χρόνο που χρειάστηκα για να ελέγξω τη λειτουργικότητα του κώδικα και να κάνω τις απαραίτητες διορθώσεις. Ο χρόνος που χρειάστηκε το ChatGPT για να παράξει τον αντίστοιχο κώδικα υλοποίησης του ίδιου προβλήματος ήταν περίπου 3 δευτερόλεπτα, δηλαδή το 1/300 του χρόνου που χρειάζεται ένας μέσος προγραμματιστής.

4.4.2 Ταχύτητα υλοποίησης

Και οι δύο κώδικες ακολουθούν την ίδια γραμμή υλοποίησης, μετατρέπουν τον δισδιάστατο πίνακα σε μονοδιάστατο, κάνουν την ταξινόμηση και τέλος κατασκευάζουν τον ταξινομημένο δισδιάστατο πίνακα. Συνεπώς, η πολυπλοκότητα είναι $O(n^2 \log(n))$ και στις δύο περιπτώσεις άρα έχουν παρόμοια ταχύτητα υλοποίησης

4.4.3 Αποτελεσματικότητα

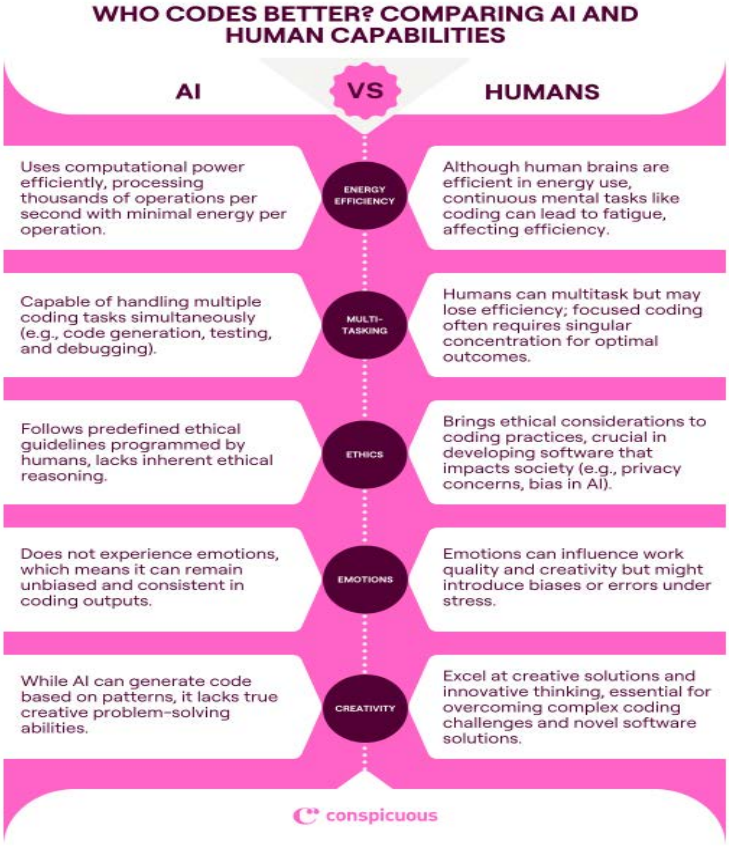
Ο δικός μου κώδικας επιστρέφει το σωστό αποτέλεσμα όμως είναι λειτουργικός μόνο για τετράγωνες πίνακες (n x n) ενώ ο κώδικας που μας έδωσε το ChatGPT λειτουργεί σωστά για οποιονδήποτε δισδιάστατο πίνακα.

4.4.4 Ευκρίνεια και Τεκμηρίωση

Και οι δύο κώδικες είναι απλοί και καθαροί στη σύνταξή τους, ωστόσο ο κώδικας που πήραμε από το LLM περιέχει σχόλια που εξηγούν το κάθε βήμα αλλά και πιο περιγραφικές ονομασίες στις μεταβλητές του. Κατά συνέπεια, ο κώδικας του ChatGPT είναι πιο ευανάγνωστος και κατανοητός από κάποιον τρίτο που μπορεί να τον διαβάσει.

4.4.5 Σφάλματα και Περιορισμοί

Ο κώδικας που έχω γράψει λειτουργεί ορθά για το συγκεκριμένο παράδειγμα που έχουμε τετράγωνο πίνακα, αλλά δεν φρόντισα να λειτουργεί για περιπτώσεις που το μήκος των γραμμών του πίνακα δεν θα ήταν το ίδιο. Αντιθέτως, ο κώδικας από το LLM περιλαμβάνει έλεγχο για τον αριθμό των στηλών και των γραμμών του πίνακα οπότε είναι πιο ευέλικτος.



Εικόνα 4.4: Σύγκριση ανθρώπου και μηχανής στη παραγωγή κώδικα[57].

4.5 Συμπέρασμα

Εν κατακλείδι, συγκρίνοντας τον κώδικα που παράγεται από ένα LLM με τον κώδικα που γράφεται από έναν προγραμματιστή διαπιστώσαμε κάποιες διαφορές αλλά και κάποιες ομοιότητες. Αρχικά, η σημαντικότερη διαφορά είναι το πόσο πιο χρονοβόρα είναι η γραφή του κώδικα από έναν άνθρωπο σε σχέση με έναν μεγάλο γλωσσικό μοντέλο το οποίο μπορεί να παράξει κώδικα σε λίγα δευτερόλεπτα. Σε απλά προβλήματα η λειτουργικότητα και η αποτελεσματικότητα του κώδικα από LLM μπορεί να είναι επαρκής, αλλά όταν τα προβλήματα γίνονται πιο πολύπλοκα χρειάζεται η παρέμβαση του προγραμματιστή καθώς υπερτερεί στη κατανόηση των προβλημάτων και στη βελτιστοποίηση του κώδικα. Οπότε η ιδανική χρήση των μεγάλων γλωσσικών μοντέλων, όσον αφορά την παραγωγή κώδικα, είναι σαν βοηθητικό εργαλείο δέχοντας τις απαραίτητες εντολές και καθοδήγηση από τον άνθρωπο προγραμματιστή.

Κριτήρια	**Προγραμματιστής**	**Large Language Model**
Χρόνος γραφής	Αργή διότι χρειάζεται ανάλυση της λογικής του προβλήματος	Πολύ γρήγορη καθώς γίνεται αυτόματα η ανάλυση του προβλήματος
Ταχύτητα Υλοποίησης	$O(n^2 \log(n))$	$O(n^2 \log(n))$
Τεκμηρίωση	Συνήθως περιγραφικά ονόματα μεταβλητών και επαρκή σχόλια	Παρέχει πλήρη τεκμηρίωση και σχόλια για κάθε βήμα του κώδικα
Σφάλματα	Μέσω ανάλυσης και δοκιμών αναγνωρίζει και διορθώνει σφάλματα	Μπορεί να εντοπίσει σφάλματα αλλά δεν κατανοεί τη λογική τους
Αποτελεσματικότητα	Δίνει βέλτιστες λύσεις έχοντας κατανοήσει το πρόβλημα	Πετυχαίνει τη λειτουργικότητα χρησιμοποιώντας πρότυπα και προσεγγίσεις στα οποία έχει εκπαιδευτεί

ΚΕΦΑΛΑΙΟ 5

ΔΥΝΑΤΟΤΗΤΑ ΕΥΡΕΣΗ ΣΦΑΛΜΑΤΩΝ

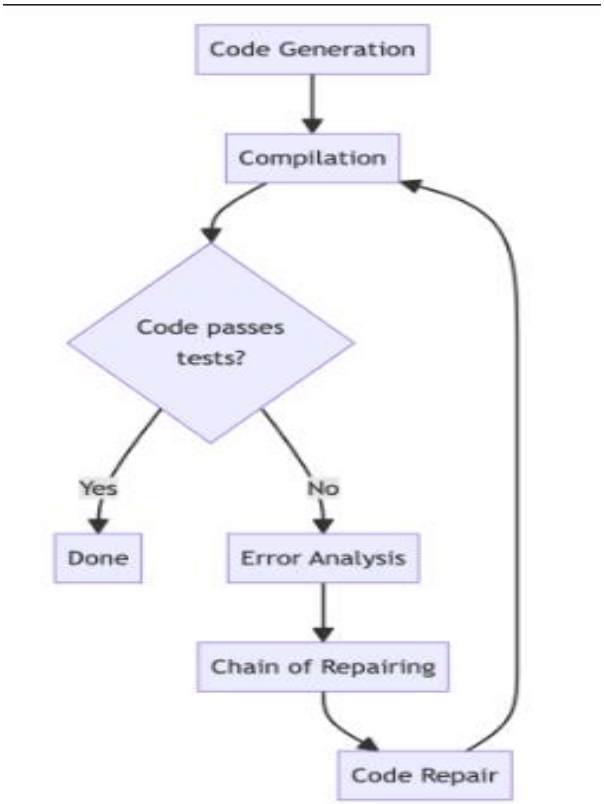
5.1 Αξιολόγηση δυνατοτήτων debugging των LLMs

Η διαδικασία του debugging, ή αλλιώς ο εντοπισμός και διόρθωση λαθών που μπορεί να εμφανιστούν στην υλοποίηση ενός κώδικα είναι απαραίτητη κα διασφαλίζει την αξιοπιστία του τελικού αποτελέσματος. Τα μεγάλα γλωσσικά μοντέλα όπως το GPT-4, το Codex και άλλα, έχουν τη ικανότητα να εντοπίζουν και να διορθώνουν σφάλματα[58, 59, 60] αλλά και να προτείνουν λύσεις που μπορεί να βελτιώσουν τη λειτουργικότητα του κώδικα. Αυτό τα καθιστά πολύ χρήσιμα εργαλεία για κάθε προγραμματιστή αφού μειώνουν σημαντικά το χρόνο που χρειάζεται η διαδικασία debugging.

Αρχικά, αναλύοντας τον κώδικα που τους δίνεται, τα LLMs μπορούν να εντοπίσουν συντακτικά λάθη όπως η έλλειψη παρενθέσεων και αγκυλών ή η λανθασμένη ονομασία μεταβλητών και συναρτήσεων, προτείνοντας παράλληλα ανάλογες βελτιώσεις. Ακόμη, μπορούν να αναγνωρίσουν λογικά λάθη του κώδικα, δηλαδή λάθη που δεν διακόπτουν την εκτέλεση του κώδικα αλλά στο τέλος παράγουν ανεπιθύμητα αποτελέσματα. Για παράδειγμα, η χρήση λανθασμένης συνθήκης μέσα σε μια επανάληψη ή η λανθασμένη χρήση μιας μεταβλητής χωρίς να έχει αρχικοποιηθεί σωστά. Ωστόσο, η απόδοση των μοντέλων στον εντοπισμό λογικών λαθών εξαρτάται από την πολυπλοκότητα του κώδικα και από την εκπαίδευση που έχει δεχτεί το συγκεκριμένο μεγάλο γλωσσικό μοντέλο.

Επιπλέον, τα LLMs προσφέρουν την δυνατότητα ανίχνευσης λαθών που εμφανίζονται κατά την εκτέλεση του προγράμματος, τα runtime errors, που οδηγούν στο πρόωρο τερματισμό του. Αυτά τα λάθη μπορεί να προκύψουν από διάφορους λόγους όπως μία πράξη διαίρεσης ενός αριθμού με το μηδέν, την υπερχείλιση κάποιας μεταβλητής, την κλήση μιας συνάρτησης με δεδομένα στα οποία δεν έχει πρόσβαση, αλλά και από

υλικές δυσλειτουργίες του υπολογιστή. Επιπρόσθετα, στην διαδικασία διόρθωσης του κώδικα τα LLMs μπορούν να προσθέσουν σχόλια και τεκμηρίωση των βημάτων που προτείνουν, κάνοντας έτσι τη λειτουργία του κώδικα πιο κατανοητή.



Εικόνα 5.1: Διαδικασία του code repairing[61].

5.2 Δοκιμή LLM για ανίχνευση λαθών

Θα δοκιμάσουμε το ChatGPT αν μπορεί να εντοπίσει λάθη όπως συντακτικά, λογικά και λάθη εκτέλεσης και να κάνει debug σε ήδη υπάρχοντα παραδείγματα κώδικα που παρήγαγε παραπάνω.

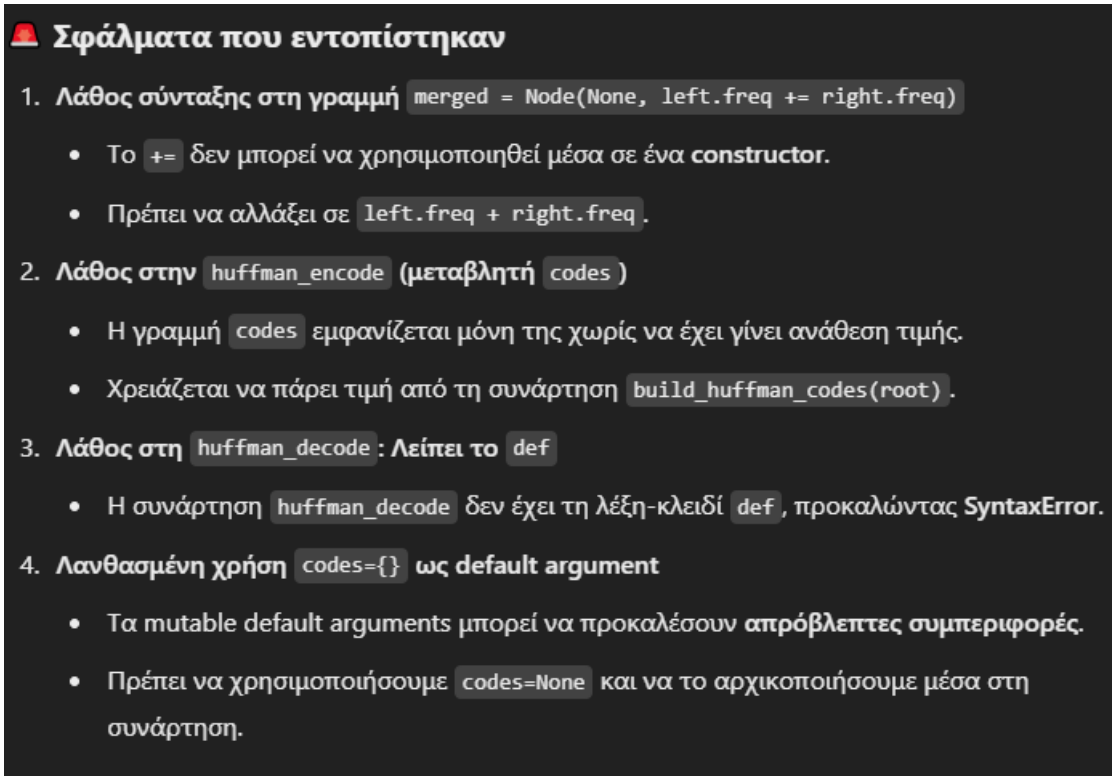
5.2.1 Συμπίεση Huffman με bugs

Πάνω στον κώδικα με την υλοποίηση Huffman που μας είχε δώσει το ChatGPT θα προσθέσουμε bugs παραποιώντας κάποιες γραμμές κώδικα για να δημιουργηθούν τα αντίστοιχα errors.

```
import heapq
from collections import defaultdict

class Node:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq

def build_huffman_tree(text):
    frequency = defaultdict(int)
    for char in text:
        frequency[char] += 1
    priority_queue = [Node(char, freq) for char, freq in frequency.items()]
    heapq.heapify(priority_queue)
    while len(priority_queue) > 1:
        left = heapq.heappop(priority_queue)
        right = heapq.heappop(priority_queue)
        merged = Node(None, left.freq += right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(priority_queue, merged)
    return priority_queue[0]

def build_huffman_codes(root, current_code="", codes={}):
    if root is None:
        return
    if root.char is not None:
        codes[root.char] = current_code
    build_huffman_codes(root.left, current_code + "0", codes)
    build_huffman_codes(root.right, current_code + "1", codes)
    return codes
```

```
def huffman_encode(text):
    root = build_huffman_tree(text)
    codes
    encoded_text = "".join(codes[char] for char in text)
    return encoded_text, codes

huffman_decode(encoded_text, codes):
    reverse_codes = {v: k for k, v in codes.items()}
    decoded_text = ""
    current_code = ""
    for bit in encoded_text:
        current_code += bit
        if current_code in reverse_codes:
            decoded_text += reverse_codes[current_code]
            current_code = ""
    return decoded_text


text = "hello huffman"
encoded_text, codes = huffman_encode(text)
decoded_text = huffman_decode(encoded_text, codes)

print("Αρχικό Κείμενο:", text)
print("Κωδικοποιημένο Κείμενο:", encoded_text)
print("Λεξικό Κωδικοποίησης:", codes)
print("Αποκωδικοποιημένο Κείμενο:", decoded_text)
```

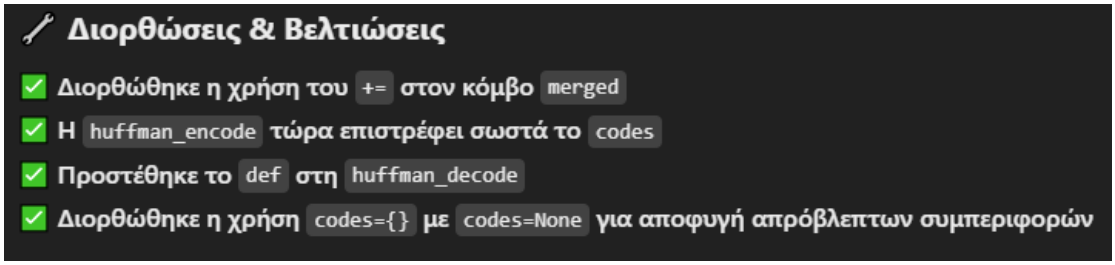
Ζητήσαμε από το ChatGPT να ελέγξει την ορθότητα και τη λειτουργικότητα του κώδικα και αν βρει λάθη να τα διορθώσει και μας έδωσε το παρακάτω αποτέλεσμα.



🚨 **Σφάλματα που εντοπίστηκαν**

1. **Λάθος σύνταξης στη γραμμή** `merged = Node(None, left.freq += right.freq)`
   - Το `+=` δεν μπορεί να χρησιμοποιηθεί μέσα σε ένα **constructor**.
   - Πρέπει να αλλάξει σε `left.freq + right.freq`.
2. **Λάθος στην** `huffman_encode` **(μεταβλητή** `codes`**)**
   - Η γραμμή `codes` εμφανίζεται μόνη της χωρίς να έχει γίνει ανάθεση τιμής.
   - Χρειάζεται να πάρει τιμή από τη συνάρτηση `build_huffman_codes(root)`.
3. **Λάθος στη** `huffman_decode`**: Λείπει το** `def`
   - Η συνάρτηση `huffman_decode` δεν έχει τη λέξη-κλειδί `def`, προκαλώντας **SyntaxError**.
4. **Λανθασμένη χρήση** `codes={}` **ως default argument**
   - Τα mutable default arguments μπορεί να προκαλέσουν **απρόβλεπτες συμπεριφορές**.
   - Πρέπει να χρησιμοποιήσουμε `codes=None` και να το αρχικοποιήσουμε μέσα στη συνάρτηση.

Και έπειτα μας δίνει τον διορθωμένο κώδικα ο οποίος λειτουργεί σωστά κάνοντας τις εξής διορθώσεις:



🛠️ **Διορθώσεις & Βελτιώσεις**

- ✅ **Διορθώθηκε η χρήση του** `+=` **στον κόμβο** `merged`
- ✅ **Η** `huffman_encode` **τώρα επιστρέφει σωστά το** `codes`
- ✅ **Προστέθηκε το** `def` **στη** `huffman_decode`
- ✅ **Διορθώθηκε η χρήση** `codes={}` **με** `codes=None` **για αποφυγή απρόβλεπτων συμπεριφορών**

5.2.2 Max flow με bugs

Παρομοίως, βάζουμε πάλι κάποια bugs στο κώδικα υλοποίησης του προβλήματος μέγιστης ροής.

```
from collections import defaultdict

class Graph
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = defaultdict(lambda: defaultdict(int))

    def add_edge(self, u, v, capacity):
        self.graph[u][v] = capacity

    def dfs(self, source, sink, visited, flow):
        if source == sink:
            return flow
        visited.add(source)
        for neighbor, capacity in self.graph[source].items():
            if neighbor not in visited and capacity >= 0:
                min_flow = min(flow, capacity)
                result = self.dfs(neighbor, sink, visited, min_flow)
                if result > 0:
                    self.graph[source][neighbor] -= result
                    self.graph[neighbor][source] += result
                    return result
        return 0

    def ford_fulkerson(self, source, sink):
        max_flow = 0
        while True:
            flow = self.dfs(source, sink, visited, [float('Inf')])
            if flow == 0:
                break
            max_flow += flow
        return max_flow
```

```python
# Παράδειγμα χρήσης
g = Graph(6)
g.add_edge(0, 1, 16)
g.add_edge(0, 2, 13)
g.add_edge(1, 2, 10)
g.add_edge(1, 3, 12)
g.add_edge(2, 4, 14)
g.add_edge(3, 5, 20)
g.add_edge(4, 3, 7)
g.add_edge(4, 5, 4)

source, sink = 0, 5
max_flow = g.ford_fulkerson(source, sink)
print(f"Μέγιστη ροή: {max_flow}")
```

Ζητήσαμε από το ChatGPT να ελέγξει την ορθότητα και τη λειτουργικότητα του κώδικα και αν βρει λάθη να τα διορθώσει και μας έδωσε το παρακάτω αποτέλεσμα.

🚨 Σφάλματα που εντοπίστηκαν

1. **Συντακτικό λάθος στο** `class Graph`
   - **Λείπει το** `:` **μετά το** `class Graph`
   - Πρέπει να γραφτεί `class Graph:` αντί για `class Graph`
2. **Η μεταβλητή** `visited` **στη** `ford_fulkerson` **δεν έχει δηλωθεί**
   - Στη γραμμή:

     python — Αντιγραφή — Επεξεργασία

     ```python
     flow = self.dfs(source, sink, visited, [float('Inf')])
     ```

     η `visited` δεν έχει αρχικοποιηθεί.
   - Πρέπει να δηλωθεί μέσα στη `ford_fulkerson` ως `visited = set()`.
3. **Λάθος τύπος δεδομένων στο** `flow`
   - Το `flow` αρχικοποιείται ως λίστα `[float('Inf')]` αντί για απλή αριθμητική τιμή `float('Inf')`.
   - Πρέπει να γραφτεί ως:

     python — Αντιγραφή — Επεξεργασία

     ```python
     flow = self.dfs(source, sink, visited, float('Inf'))
     ```
4. **Λάθος στη συνθήκη** `capacity >= 0` **στη** `dfs`
   - Το `capacity` πρέπει να είναι **μεγαλύτερο από 0** (`capacity > 0`), διαφορετικά θα επιτρέπει ροές σε ακμές με **μηδενική** **ανότητα**.

Και έπειτα μας δίνει τον διορθωμένο κώδικα ο οποίος λειτουργεί σωστά κάνοντας τις εξής διορθώσεις:

Διορθώσεις & Βελτιώσεις

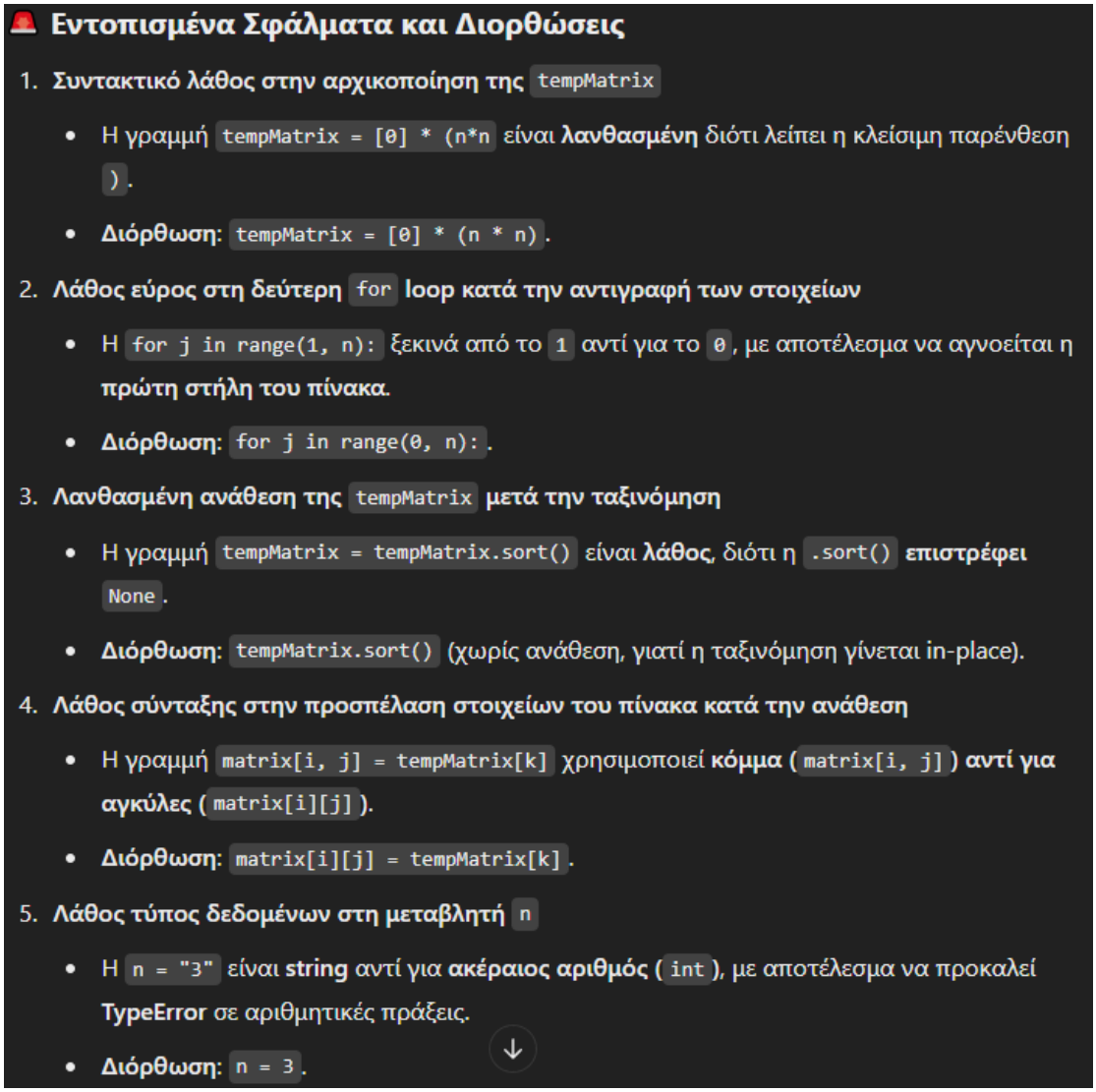
- ✅ Προστέθηκε `:` στο `class Graph:`
- ✅ Αρχικοποιήθηκε σωστά το `visited` μέσα στη `ford_fulkerson`
- ✅ Τροποποιήθηκε το `capacity >= 0` σε `capacity > 0` στη `dfs`
- ✅ Αφαιρέθηκε η περιττή λίστα `[float('Inf')]` και αντικαταστάθηκε με `float('Inf')`

5.2.3. Ταξινόμηση δισδιάστατου πίνακα με bugs

Τώρα θα δοκιμάσουμε να βάλουμε κάποια bugs στον κώδικα που γράψαμε για την ταξινόμηση ενός δισδιάστατου πίνακα ακολουθώντας την ίδια διαδικασία.

```
def sorting(matrix, n):
    k = 0
    tempMatrix = [0] * (n*n

    for i in range(0, n):
        for j in range(1, n):
            tempMatrix[k] = matrix[i][j]
            k = k + 1

    tempMatrix = tempMatrix.sort()
    k = 0

    for i in range(0, n):
        for j in range(0, n):
            matrix[i, j] = tempMatrix[k]
            k = k + 1

def printMatrix(matrix, n):
    for i in range(0, n):
        for j in range(0, n):
            print(matrix[i][j], end=" ")
        print(" ")

matrix = [[3, 8, 7],
          [5, 2, 9],
          [4, 6, 1]]
n = "3"
print("Αρχικός Πίνακας:")
printMatrix(matrix, n)
sorting(matrix, n)
print("Τελικός Πίνακας")
printMatrix(matrix, n)
```

Ζητήσαμε από το ChatGPT να ελέγξει την ορθότητα και τη λειτουργικότητα του κώδικα και αν βρει λάθη να τα διορθώσει και μας έδωσε το παρακάτω αποτέλεσμα.



🚨 Εντοπισμένα Σφάλματα και Διορθώσεις

1. **Συντακτικό λάθος στην αρχικοποίηση της** `tempMatrix`
   - Η γραμμή `tempMatrix = [0] * (n*n` είναι **λανθασμένη** διότι λείπει η κλείσιμη παρένθεση `)`.
   - **Διόρθωση:** `tempMatrix = [0] * (n * n)`.
2. **Λάθος εύρος στη δεύτερη** `for` **loop κατά την αντιγραφή των στοιχείων**
   - Η `for j in range(1, n):` ξεκινά από το `1` αντί για το `0`, με αποτέλεσμα να αγνοείται η **πρώτη στήλη του πίνακα.**
   - **Διόρθωση:** `for j in range(0, n):`.
3. **Λανθασμένη ανάθεση της** `tempMatrix` **μετά την ταξινόμηση**
   - Η γραμμή `tempMatrix = tempMatrix.sort()` είναι **λάθος**, διότι η `.sort()` **επιστρέφει** `None`.
   - **Διόρθωση:** `tempMatrix.sort()` (χωρίς ανάθεση, γιατί η ταξινόμηση γίνεται in-place).
4. **Λάθος σύνταξης στην προσπέλαση στοιχείων του πίνακα κατά την ανάθεση**
   - Η γραμμή `matrix[i, j] = tempMatrix[k]` χρησιμοποιεί **κόμμα (** `matrix[i, j]` **) αντί για αγκύλες (** `matrix[i][j]` **).**
   - **Διόρθωση:** `matrix[i][j] = tempMatrix[k]`.
5. **Λάθος τύπος δεδομένων στη μεταβλητή** `n`
   - Η `n = "3"` είναι **string** αντί για **ακέραιος αριθμός (** `int` **)**, με αποτέλεσμα να προκαλεί **TypeError** σε αριθμητικές πράξεις.
   - **Διόρθωση:** `n = 3`.

5.3 Ανάλυση ακρίβειας και αξιοπιστίας

Παρόλο που τα μεγάλα γλωσσικά μοντέλα φάνηκαν ικανά στο να εντοπίζουν και να διορθώνουν διάφορα σφάλματα στο κώδικα, είναι σημαντικό να ελέγξουμε την ακρίβεια και την αξιοπιστία των αποτελεσμάτων που παράγουν ώστε να είναι πιο ασφαλής η ενσωμάτωσή τους στις καθημερινές εργασίες των προγραμματιστών.

5.3.1 Ακρίβεια

Όπως είδαμε και παραπάνω, όταν κάναμε χρήση του ChatGPT[62, 63] για να ανιχνεύσει και διορθώσει σφάλματα φάνηκε πως τα καταφέρνει με υψηλή ακρίβεια σε λιγοστό χρόνο. Ειδικότερα, ήταν ιδιαίτερα αποτελεσματικό σε περιπτώσεις βασικών λαθών όπως η έλλειψη παρενθέσεων, η λανθασμένη χρήση μεταβλητών και συναρτήσεων, η μη ορθή αρχικοποίηση μεταβλητών, αλλά παρείχε και προτάσεις που επιδιόρθωναν λογικά λάθη που προέκυψαν. Ωστόσο, τα LLMs δεν έχουν την ικανότητα να κατανοήσουν σε βάθος το πρόβλημα που τους δίνεται, συνεπώς όταν έρχονται αντιμέτωπα με πιο σύνθετα λογικά σφάλματα δεν μπορούν να ανταπεξέλθουν. Για αυτό οφείλεται επίσης, η εκπαίδευση που έχουν δεχτεί και η απουσία κριτικής ικανότητας, καθώς για να προτείνουν διορθώσεις, ανατρέχουν σε προϋπάρχοντα παραδείγματα και μοτίβα χωρίς να καταλαβαίνουν πλήρως τις ιδιαιτερότητες ενός πιο σύνθετου προβλήματος.

5.3.2 Αξιοπιστία

Όσον αφορά την αξιοπιστία των LLMs στο κομμάτι του debugging[64], φάνηκε να είναι αρκετά αποτελεσματικά όταν κλήθηκαν να αναλύσουν και να διορθώσουν μικρά κομμάτια κώδικα. Επίσης, όταν του δόθηκε ο ίδιος λανθασμένος κώδικας επανειλημμένα, το ChatGPT παρουσίασε τις ίδιες προτάσεις βελτίωσης που το καθιστά έτσι αξιόπιστο και συνεπές σαν εργαλείο. Αυτό ωστόσο δεν ισχύει σε περιπτώσεις με πιο μεγάλα και σύνθετα προγράμματα που απαιτούν βαθύτερη κατανόηση

περισσότερων παραμέτρων και στοιχείων, καθώς αδυνατούν να συγκρατήσουν μια συνεκτικότητα στην ανάλυσή τους και παραλείπουν κρίσιμα κομμάτια του κώδικα. Επιπρόσθετα, αν τους δοθεί ένα μήνυμα λάθος που προέκυψε από σφάλμα κατά την εκτέλεση του κώδικα, έχουν την δυνατότητα, αφού αναλύσουν το μήνυμα αυτό, να προτείνουν κατάλληλες λύσεις. Όμως, δεν έχουν την δυνατότητα να εκτελέσουν αυτόνομα τον διορθωμένο κώδικα που παράγουν, με αποτέλεσμα κάποιες φορές διαμορφώνοντας τον κώδικα να αλλάζουν την λειτουργικότητά του ή να δημιουργούν νέα προβλήματα.

5.4 Συμπεράσματα

Ύστερα από τις δοκιμές debugging που κάναμε χρησιμοποιώντας το ChatGPT, παρατηρήσαμε ότι έχει τη δυνατότητα να ανιχνεύσει και να στη συνέχεια να διορθώσει λάθη με μεγάλη ακρίβεια και ευκολία, είτε αυτά είναι συντακτικά, είτε λογικά, ιδιαίτερα σε περιπτώσεις μικρής έκτασης κώδικα. Εξίσου αποτελεσματικά είναι και στην ανάλυση λαθών που προκύπτουν κατά την εκτέλεση του κώδικα και στη διόρθωση αυτών προτείνοντας λύσεις που επιτυγχάνουν τη βέλτιστη λειτουργικότητα. Ακόμη, δίνοντας επεξηγήσεις και τεκμηρίωση πάνω στις διορθώσεις που προσφέρει, βοηθά στην κατανόηση των λαθών και στην αποφυγή της επανεμφάνισή τους στο μέλλον.

Από την άλλη, όταν τα μεγάλα γλωσσικά μοντέλα χρειάζεται να αναλύσουν πιο μεγάλα και σύνθετα κομμάτια κώδικα, δυσκολεύονται να κατανοήσουν την συνολική λειτουργικότητά του και κατά συνέπεια μειώνεται η αξιοπιστία τους όσον αφορά τον εντοπισμό λογικών λαθών που μπορεί να προκύψουν. Ακόμη, η αδυναμία των LLMs να εκτελέσουν αυτόνομα και να κάνουν δοκιμές στο κώδικα, περιορίζει τη χρήση τους σε δυναμικά προβλήματα.

Συνεπώς, ενώ είδαμε ότι πως τα μεγάλα γλωσσικά μοντέλα μπορούν να αποδειχτούν χρήσιμα στη διαδικασία διόρθωσης ενός κώδικα, υπάρχουν περιπτώσεις στις οποίες οι δυνατότητές τους είναι περιορισμένες. Για αυτό, είναι απαραίτητη η εποπτεία από κάποιον προγραμματιστή που θα καθοδηγεί και θα ελέγχει τις λειτουργίες του μοντέλου. Έτσι, για να αξιοποιηθούν στο έπακρο οι ικανότητες των LLMs στην

γρήγορη ανάλυση και διόρθωση κώδικα, πρέπει να συνδυαστούν με τις τεχνολογικές γνώσεις και τη κρίση των προγραμματιστών ώστε να φροντίσουν ότι οι λύσεις που προτείνονται είναι αξιόπιστες και κατάλληλες για τη λειτουργικότητα του επικείμενου προβλήματος.

ΚΕΦΑΛΑΙΟ 6

ΕΠΙΔΡΑΣΗ ΤΩΝ ΜΕΓΑΛΩΝ ΓΛΩΣΣΙΚΩΝ ΜΟΝΤΕΛΩΝ

Στις μέρες μας, κανείς δεν μπορεί να αμφισβητήσει τη χρησιμότητα των μεγάλων γλωσσικών μοντέλων και τις αλλαγές που έχουν φέρει στο τομέα ανάπτυξης λογισμικού[65]. Οι δυνατότητες που προσφέρουν με την αυτόματη παραγωγή κώδικα και τη προσθήκη τεκμηρίωσης και επεξήγησης των βημάτων έχουν συμβάλλει σημαντικά στην μείωση του χρόνου της διαδικασίας σχεδίασης λογισμικού αλλά και στην αύξηση της παραγωγικότητας των προγραμματιστών. Ακόμη, αποδείχθηκε πως μπορούν να συνεισφέρουν στον εντοπισμό και στη διόρθωση σφαλμάτων του κώδικα επιταχύνοντας έτσι τη διαδικασία του debugging.

Ωστόσο, παρά όλες τις διευκολύνσεις που προσφέρουν στους προγραμματιστές, τα μεγάλα γλωσσικά μοντέλα δεν αποτελούν “πανάκεια” διότι έχουν και αυτά αστοχίες και οι λύσεις που προτείνουν δεν είναι πάντα αξιόπιστες. Αυτοί και άλλοι περιορισμοί των μοντέλων γεννούν ερωτήματα για το αν μπορούν πραγματικά να αντικαταστήσουν τους προγραμματιστές στη διαδικασία ανάπτυξης λογισμικού. Η απάντηση είναι πως ο στόχος της χρήσης των LLMs δεν θα έπρεπε να είναι η αυτοματοποίηση της διαδικασίας σχεδίασης λογισμικού και κατά συνέπεια η αντικατάσταση των προγραμματιστών σε αυτό το κομμάτι. Αντιθέτως, για να επιτευχθεί η καλύτερη αξιοποίηση των δυνατοτήτων των μεγάλων γλωσσικών μοντέλων, θα έπρεπε να γίνεται χρήση αυτών από τους προγραμματιστές ως βοηθητικά εργαλεία που βελτιώνουν την αποδοτικότητα και την ταχύτητα κάποιων διαδικασιών στη καθημερινή ροή εργασιών τους.

Για να πετύχουμε λοιπόν την βέλτιστη αξιοποίηση των μοντέλων, είναι σημαντικό να γίνουν κάποιες εκπαιδευτικές ενέργειας για τη χρήση τους από τους προγραμματιστές. Αρχικά, δεν θα πρέπει οι προγραμματιστές να στηρίζονται ολοκληρωτικά στις ικανότητες των LLMs να παράγουν κώδικα, καθώς έτσι φθείρονται σταδιακά οι δικές τους δεξιότητες και η κριτική τους ικανότητα. Είναι απαραίτητο να μπορούν να αναλύουν και να ελέγχουν την ορθότητα των αποτελεσμάτων που προτείνει ένα

μοντέλο πριν τα χρησιμοποιήσουν, διότι μπορεί να περιέχουν σφάλματα ή να μην ανταποκρίνονται στην λειτουργικότητα που αποζητούν. Στην συνέχεια, θα μπορούσαμε να εντάξουμε τα μεγάλα γλωσσικά μοντέλα και τις εφαρμογές τους στην εκπαιδευτική διαδικασία των νέων προγραμματιστών ενσωματώνοντάς τα σε προγραμματιστικά εκπαιδευτικά περιβάλλοντα όπως το Scratch. Η δυνατότητα κάποιων μοντέλων να παρέχουν επεξηγήσεις και καθοδήγηση πάνω σε παραδείγματα διαφόρων γλωσσών προγραμματισμού, μπορεί να συμβάλλει στην καλύτερη κατανόηση συναρτήσεων και αλγορίθμων δίνοντας ταυτόχρονα ανατροφοδότηση στους εκπαιδευόμενους.

Εξίσου σημαντικό είναι να υπάρχει πάντα ανθρώπινη εποπτεία και να γίνονται τακτικοί έλεγχοι όταν γίνεται χρήση των LLMs, αφού δεν είναι πάντα αξιόπιστα και ασφαλή τα αποτελέσματα που παράγουν. Μπορούν να δημιουργηθούν μηχανισμοί ελέγχου από τους προγραμματιστές που θα διασφαλίζουν την ποιότητα του κώδικα και την ορθή λειτουργικότητα του τελικού λογισμικού, στηριζόμενοι στη κρίση και στις τεχνολογικές γνώσεις τους. Έτσι το τελικό αποτέλεσμα θα έχει ελεγχθεί και εγκριθεί από τους προγραμματιστές με αποτέλεσμα να μην υπάρχουν λάθη και κενά ασφαλείας στο κώδικα.

ΚΕΦΑΛΑΙΟ 7

ΣΥΜΠΕΡΑΣΜΑΤΑ

Συμπεραίνοντας, στο πλαίσιο της παρούσας διπλωματικής παρατηρήσαμε πως τα μεγάλα γλωσσικά μοντέλα και οι εφαρμογές τους μπορούν να αποτελέσουν σημαντικά βοηθητικά εργαλεία για τους προγραμματιστές, επαναπροσδιορίζοντας την παραδοσιακή διαδικασία ανάπτυξης λογισμικού. Παρά τους περιορισμούς που εμφανίζουν, όταν συνδυαστούν με τις ικανότητες ενός προγραμματιστή μπορούν να πετύχουν τη παραγωγή, τον έλεγχο αλλά και τη διόρθωση κώδικα σε πρωτόγνωρες ταχύτητες, βελτιώνοντας παράλληλα την ποιότητα και την αποδοτικότητα του λογισμικού.

Στο μέλλον, οι προγραμματιστές θα πρέπει να χρησιμοποιούν την κριτική τους σκέψη και να μην εμπιστεύονται τυφλά τις ικανότητες των LLMs, καθώς μόνο μέσω της συνεργασίας των δύο μπορεί να επιτευχθεί η βέλτιστη αξιοποίηση των ευκαιριών που δημιουργούνται. Όσο συνεχίζει να αυξάνεται η εκπαίδευση των μεγάλων γλωσσικών μοντέλων και να δημιουργούνται νέα πιο εξειδικευμένα μοντέλα, τόσο θα αυξάνονται και οι εφαρμογές τους ανοίγοντας νέους δρόμους στο τομέα ανάπτυξης λογισμικού. Αρκεί να μην αντιμετωπίζονται ως αντικαταστάτες των προγραμματιστών αλλά ως ένα ισχυρό εργαλείο που συμβάλλει στην αύξηση της παραγωγικότητας και στη διευκόλυνση στη ροή καθημερινών εργασιών. Με σωστή εκπαίδευση των προγραμματιστών και ελεγχόμενη χρήση των μεγάλων γλωσσικών μοντέλων μπορεί να γίνει εκτενέστερη ενσωμάτωσή τους στη τεχνολογία λογισμικού δίνοντας νέες προοπτικές για το μέλλον

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Chenguang Wang, Mu Li, Alexander J. Smola. "*Language Models with Transformers*", in *arXiv:1904.09408*[cs.CL]

[2] M. A. K. Raiaan *et al*., "A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges," in *IEEE Access*, vol. 12, pp. 26839-26874, 2024

[3] Nayan B. Ruparelia. 2010. Software development lifecycle models. in SIGSOFT Softw. Eng. Notes 35, 3 (May 2010), 8–13

[4] P. L. Li, A. J. Ko and J. Zhu, "What Makes a Great Software Engineer?" *2015 in IEEE/ACM 37th IEEE International Conference on Software Engineering*, Florence, Italy, 2015, pp. 700-710

[5] Derniame, Jean-Claude, Badara A. Kaba, and David Wastell, eds. *Software process: principles, methodology, and technology*. in Springer Science & Business Media, 1999.

[6] The Educative Team, "Top coding languages you should learn in 2025", in Dev Learning Daily, 2024

[7] M. Beller, G. Gousios, A. Panichella, S. Proksch, S. Amann and A. Zaidman, "Developer Testing in the IDE: Patterns, Beliefs, and Behavior," in *IEEE Transactions on Software Engineering*, vol. 45, no. 3, pp. 261-284, 1 March 2019

[8] Fuggetta, Alfonso. "Software process: a roadmap." in *Proceedings of the Conference on the Future of Software Engineering*. 2000

[9] Nabil M. Mohammed, Mahmood Niazi, Mohammad Alshayeb, Sajjad Mahmood, Exploring software security approaches in software development lifecycle: A systematic mapping study, Computer Standards & Interfaces, Volume 50, 2017, Pages 107-115, ISSN 0920-5489

[10] Jeff Shepard. "Secure Software Development Can Be Challenging", for Mouser Electronics, Published December 22, 2020

[11] Fowler, Martin. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018

[12] Source: https://www.techtarget.com/searchapparchitecture/definition/refactoring

[13] Shaw, Mary. "Software engineering education: A roadmap." in *Proceedings of the Conference on the Future of Software Engineering*. 2000

[14] Humphrey, Watts S. *Psp (sm): a self-improvement process for software engineers*. Addison-Wesley Professional, 2005

[15] Ziyin Zhang, Chaoyu Chen, Bingchang Liu, Cong Liao, Zi Gong, Hang Yu, Jianguo Li, Rui Wang. "Unifying the Perspectives of NLP and Software Engineering: A Survey on Language Models for Code", 2024, in *arXiv:2311.07989* [cs.CL]

[16] Muhammad Yasir Saleem, "The Process of Natural Language Processing", LinkedIn, September 11, 2024, from https://www.linkedin.com/in/muhammad-yasir-saleem/

[17] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to Help With Code Understanding. in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24). Association for Computing Machinery*, New York, NY, USA, Article 97, 1–13

[18] Sumit Kumar Dam, Choong Seon Hong, Yu Qiao, Chaoning Zhang." A Complete Survey on LLM-based AI Chatbots", 2024, in *arXiv:2406.16937* [cs.CL]

[19] Naveed, Humza, et al. "A comprehensive overview of large language models." in *arXiv preprint arXiv:2307.06435* (2023)

[20] Ferraris, Andrea Filippo, et al. "The architecture of language: Understanding the mechanics behind LLMs." in *Cambridge Forum on AI: Law and Governance*. Vol. 1. Cambridge University Press, 2025

[21] Achiam, Josh, et al. "GPT-4 technical report." in *arXiv preprint arXiv:2303.08774* (2023)

[22] Poldrack, Russell A., Thomas Lu, and Gašper Beguš. "AI-assisted coding: Experiments with GPT-4." in *arXiv preprint arXiv:2304.13187* (2023)

[23] Image Source: https://www.cursor-ide.com/blog/gpt-41-guide-2025

[24] Finnie-Ansley, James, et al. "The robots are coming: Exploring the implications of OpenAI codex on introductory programming." in *Proceedings of the 24th Australasian computing education conference*. 2022

[25] Image Source: https://medium.com/@singhrajni2210/the-parallel-revolution-how-openai-codex-is-transforming-software-development-2dda3f17f5a3

[26] P. Y. Reddy, N. Sri Satya Aarthi, S. Pooja, B. Veerendra and S. D, "Enhancing Code Intelligence with CodeT5: A Unified Approach to Code Analysis and Generation," *2025 in International Conference on Artificial Intelligence and Data Engineering (AIDE)*, Nitte, India, 2025, pp. 135-140

[27] Wang, Yue, et al. "Codet5+: Open code large language models for code understanding and generation." in *arXiv preprint arXiv:2305.07922* (2023)

[28] Wang, Yue, et al. "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation." in *arXiv preprint arXiv:2109.00859* (2021)

[29] Reini, Niklas. "The Impact of AI powered code completion in the software engineering field." (2022)

[30] *Tabnine AI code assistant: Private, personalized, protected*. Tabnine. https://www.tabnine.com/

[31] Lucas Soares. “Cogram.ai: A Coding Assistant for Data Science and Machine Learning”. in towardsdatascience. Nov 15, 2021

[32] Gozalo-Brizuela, Roberto, and Eduardo C. Garrido-Merchán. "A survey of Generative AI Applications." in *arXiv preprint arXiv:2306.02781* (2023)

[33] Lu, Junyi, et al. "Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning." *2023 in IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023

[34] Roziere, Baptiste, et al. "Code llama: Open foundation models for code." in *arXiv preprint arXiv:2308.12950* (2023)

[35] Guo, Daya, et al. "Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning." in *arXiv preprint arXiv:2501.12948* (2025)

[36] Image Source: https://www.leanware.co/insights/deepseek-r1-vs-gpt-o1

[37] Liu, Jiawei, et al. "Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation." in *Advances in Neural Information Processing Systems* 36 (2023): 21558-21572

[38] Dong, Yihong, et al. "Self-collaboration code generation via ChatGPT." in *ACM Transactions on Software Engineering and Methodology* 33.7 (2024): 1-38

[39] Jiang, Juyong, et al. "A survey on large language models for code generation." in *arXiv preprint arXiv:2406.00515* (2024)

[40] Azaria, Amos, Rina Azoulay, and Shulamit Reches. "ChatGPT is a remarkable tool—for experts." in *Data Intelligence* 6.1 (2024): 240-296

[41] Bareiß, Patrick, et al. "Code generation tools (almost) for free? a study of few-shot, pre-trained language models on code." in *arXiv preprint arXiv:2206.01335* (2022)

[42] Fan, Angela, et al. "Large language models for software engineering: Survey and open problems." *2023 in IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 2023

[43] Guo, Daya, et al. "Longcoder: A long-range pre-trained language model for code completion." in *International Conference on Machine Learning*. PMLR, 2023

[44] Chen, Yizheng, et al. "Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection." in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*. 2023

[45] Wong, Man-Fai, et al. "Natural language generation and understanding of big code for AI-assisted programming: A review." in *Entropy* 25.6 (2023): 888

[46] Kang, Sungmin, et al. "Explainable automated debugging via large language model-driven scientific debugging." in *Empirical Software Engineering* 30.2 (2025): 1-28

[47] Sakib, Fardin Ahsan, Saadat Hasan Khan, and AHM Rezaul Karim. "Extending the frontier of ChatGPT: Code generation and debugging." *2024 in International Conference on Electrical, Computer and Energy Technologies (ICECET*. IEEE, 2024

[48] Tian, Runchu, et al. "Debugbench: Evaluating debugging capability of large language models." in *arXiv preprint arXiv:2401.04621* (2024)

[49] Mastropaolo, Antonio, et al. "An empirical study on code comment completion." *2021 in IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021

[50] Geng, Mingyang, et al. "Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning." in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 2024

[51] Kanade, Aditya, et al. "Learning and evaluating contextual embedding of source code." in *International conference on machine learning*. PMLR, 2020

[52] Ahmed, Toufique, et al. "Automatic semantic augmentation of language model prompts (for code summarization)." in *Proceedings of the IEEE/ACM 46th international conference on software engineering*. 2024

[53] Lin, Luyang, et al. "Investigating bias in LLM-based bias detection: Disparities between LLMs and human perception." in *arXiv preprint arXiv:2403.14896* (2024)

[54] Shen, Da, et al. "Benchmarking language models for code syntax understanding." in *arXiv preprint arXiv:2210.14473* (2022)

[55] Dinh, Tuan, et al. "Large language models of code fail at completing code with potential bugs." in *Advances in Neural Information Processing Systems* 36 (2023): 41386-41412

[56] McAleese, Nat, et al. "LLM critics help catch LLM bugs." in *arXiv preprint arXiv:2407.00215* (2024)

[57] Image Source: https://conspicuous.com/conspicuous-blog/ai-vs-human-coders-comparative-analysis/

[58] Fan, Zhiyu, et al. "Automated repair of programs from large language models." *2023 in IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023

[59] Zhong, Li, Zilong Wang, and Jingbo Shang. "Debug like a human: A large language model debugger via verifying runtime execution step-by-step." in *arXiv preprint arXiv:2402.16906* (2024)

[60] Haque, Md Asraful, and Shuai Li. "The potential use of ChatGPT for debugging and bug fixing." (2023)

[61] Image Source: https://ai.plainenglish.io/enhancing-large-language-models-for-program-debugging-with-interactive-chain-of-repairing-e4d36fe487e6

[62] X. Yu, L. Liu, X. Hu, J. W. Keung, J. Liu and X. Xia, "Fight Fire With Fire: How Much Can We Trust ChatGPT on Source Code-Related Tasks?," in *IEEE Transactions on Software Engineering*, vol. 50, no. 12, pp. 3435-3453, Dec. 2024

[63] Sobania, Dominik, et al. "An analysis of the automatic bug fixing performance of chatgpt." *2023 in IEEE/ACM International Workshop on Automated Program Repair (APR)*. IEEE, 2023

[64] Sain, Zohaib & Serban, Razvan & Agoi, Moses & Hina, Shahzadi. (2024). Leveraging ChatGPT to Enhance Debugging: Evaluating AI-Driven Solutions in Software Development. Asian Journal of Computer Science and Technology. 13. 41-44. 10.70112/ajcst-2024.13.1.4261

[65] Hou, Xinyi, et al. "Large language models for software engineering: A systematic literature review." in *ACM Transactions on Software Engineering and Methodology* 33.8 (2024): 1-79