

ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ

ΤΜΗΜΑ ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΑΥΤΟΜΑΤΗ ΑΞΙΟΛΟΓΗΣΗ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ ΜΕ ΧΡΗΣΗ ΜΕΓΑΛΩΝ ΓΛΩΣΣΙΚΩΝ ΜΟΝΤΕΛΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εισηγητής: Εμμανουήλ Καραμπατζάκης, ΔΕ20119

Επιβλέπων: Γεώργιος Βασιλειάδης, Επίκουρος Καθηγητής

©
2025



HELLENIC MEDITERRANEAN UNIVERSITY

SCHOOL OF MANAGEMENT AND ECONOMICS SCIENCE

DEPARTMENT OF MANAGEMENT SCIENCE AND TECHNOLOGY

AUTOMATIC SOURCE CODE EVALUATION USING LARGE LANGUAGE MODELS

DIPLOMA THESIS

Student: Emmanouil Karampatzakis, DE20119

Supervisor: Giorgos Vasiliadis, Assistant Professor

©
2025

Υπεύθυνη Δήλωση *: Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Διοικητικής Επιστήμης και Τεχνολογίας του ΕΛ.ΜΕ.ΠΑ.*

ΠΕΡΙΛΗΨΗ

Η ταχεία πρόοδος των Μεγάλων Γλωσσικών Μοντέλων (LLMs) έχει δημιουργήσει νέους δρόμους στην κατανόηση και ανάλυση του πηγαίου κώδικα προγραμμάτων επιτρέποντας την ανάπτυξη ευφυών εργαλείων αξιολόγησης και υποστήριξης των προγραμματιστών. Στο πλαίσιο αυτό, η συγκεκριμένη εργασία υλοποίησε ένα ολοκληρωμένο σύστημα αυτόματης αξιολόγησης και ανασκόπησης πηγαίου κώδικα, βασισμένο στο μοντέλο Code Llama και στην τοπική εκτέλεση LLMs μέσω της πλατφόρμας Ollama. Το σύστημα είναι σε θέση να αναλύει αρχεία κώδικα, να εντοπίζει λογικά και δομικά προβλήματα, να προτείνει βελτιώσεις και να αποδίδει βαθμολογία με βάση κριτήρια αναγνωσιμότητας, ποιότητας και ορθότητας. Υποστηρίζει τρεις μεθοδολογίες αξιολόγησης (General, Loose και Strict), προσφέροντας τόσο ποιοτικό σχολιασμό όσο και ποσοτική βαθμολόγηση. Τα πειραματικά αποτελέσματα έδειξαν ότι τα LLMs, και ειδικότερα το Code Llama, μπορούν να λειτουργήσουν ως ένας αποτελεσματικός αυτόματος κριτής κώδικα, παρέχοντας χρήσιμη ανατροφοδότηση τόσο σε εκπαιδευόμενους όσο και σε επαγγελματίες. Επιπλέον, η εργασία αποδεικνύει ότι τα LLMs μπορούν να αποτελέσουν τη βάση για αξιόπιστα και επεκτάσιμα συστήματα αυτόματης αξιολόγησης κώδικα, ανοίγοντας τον δρόμο για περαιτέρω έρευνα σε συνδυαστικές προσεγγίσεις με στατικές και δυναμικές τεχνικές ανάλυσης.

Λέξεις-κλειδιά: Code Review, Code Llama, LLMs, Αυτόματη αξιολόγηση κώδικα, Ποιότητα λογισμικού.

ABSTRACT

The rapid advancement of Large Language Models (LLMs) has opened new pathways in source code understanding and analysis, enabling the development of intelligent tools for evaluation and developer assistance. In this context, the present thesis implements a complete automatic source code evaluation and review system (code reviewer system), based on Code Llama and the local execution of LLMs through the Ollama platform. The system is capable of analyzing programming files, identifying logical and structural issues, proposing improvements, and assigning a score based on readability, quality, and implementation correctness. It supports three evaluation modes (General, Loose, and Strict), offering both qualitative feedback and quantitative scoring. The experimental results showed that LLMs, and in particular Code Llama, can serve as an effective automated code evaluator, providing valuable feedback to both learners and professionals. Moreover, the findings of this thesis show that LLMs can serve as a solid foundation for reliable and extensible automated code evaluation systems, paving the way for further research that combines LLM-based reasoning with static and dynamic analysis techniques.

Keywords: Large Language Models, Code Review, Code Llama, Automated Code Evaluation, Software Quality.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΕΡΙΛΗΨΗ 1
ABSTRACT 2
ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ 3
ΕΥΧΑΡΙΣΤΙΕΣ 4
ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ 1
ΚΕΦΑΛΑΙΟ 2: ΑΝΑΣΚΟΠΗΣΗ ΒΙΒΛΙΟΓΡΑΦΙΑΣ 3
2.1 Μεγάλα Γλωσσικά Μοντέλα στην Εκπαίδευση και στην Αυτόματη Αξιολόγηση Ασκήσεων Προγραμματισμού 3
2.2 Μεγάλα Γλωσσικά Μοντέλα στην Ανασκόπηση Κώδικα (Code Review) 4
2.3 Εξειδικευμένα Μοντέλα για Προγραμματισμό 5
ΚΕΦΑΛΑΙΟ 3: ΜΕΘΟΔΟΛΟΓΙΑ 6
ΚΕΦΑΛΑΙΟ 4: ΥΛΟΠΟΙΗΣΗ 8
4.1 Τεχνολογικό υπόβαθρο και βιβλιοθήκες 8
4.2 Χρήση της πλατφόρμας Ollama 8
4.3 Δομή κώδικα και σχεδιαστικές αποφάσεις 9
4.3.1 Αξιολόγηση κώδικα 9
4.3.2 Διαχείριση αρχείων ZIP 9
4.3.3 Κύρια ροή εκτέλεσης 9
4.4 Λειτουργία του συστήματος 10
4.5 Πλεονεκτήματα και αξία της υλοποίησης 10
ΚΕΦΑΛΑΙΟ 5: ΠΑΡΑΔΕΙΓΜΑΤΑ ΧΡΗΣΗΣ 11
5.1 Υπολογισμος αθροίσματος δύο αριθμών 11
5.2 Συνάρτηση υπολογισμού παραγοντικού 12
5.3 Συνάρτηση διαίρεσης με έλεγχο σφάλματος μηδενικού διαιρέτη 13
5.4 Πρόγραμμα αθροίσματος θετικών αριθμών 14
ΚΕΦΑΛΑΙΟ 6: ΣΥΜΠΕΡΑΣΜΑΤΑ 16
ΒΙΒΛΙΟΓΡΑΦΙΑ 18

ΕΥΧΑΡΙΣΤΙΕΣ

Θέλω να ευχαριστήσω θερμά των επιβλέποντα καθηγητή μου για την καθοδήγηση και την υποστήριξη ώστε να ολοκληρωθεί αυτή η εργασία, το τμήμα για τις γνώσεις, την εμπειρία και τον εξοπλισμό, και την οικογένεια μου για την ψυχολογική στήριξη. Τέλος, θα ήθελα να εκφράσω την ευγνωμοσύνη μου προς τους δημιουργούς των Μεγάλων Γλωσσικών Μοντέλων (LLMs) και των μοντέλων CodeLlama, πάνω στα οποία βασίστηκε η εργασία αυτή.

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ

Η μελέτη και κατανόηση του πηγαίου κώδικα αποτελεί κρίσιμο βήμα για κάθε προγραμματιστή, καθώς επιτρέπει τον εντοπισμό των τμημάτων που χρειάζονται αλλαγές και τη διαμόρφωση τους ώστε να εξασφαλίζεται η συνοχή, η συντηρησιμότητα και η λειτουργική συμβατότητα του συστήματος. Η ανάπτυξη ή αναθεώρηση του κώδικα πρέπει να διέπεται από καθιερωμένες αρχές και βέλτιστες πρακτικές προγραμματισμού, με στόχο τη δημιουργία αξιόπιστου και κατανοητού λογισμικού. Στο πλαίσιο αυτό, οι προγραμματιστές υιοθετούν κοινές τεχνικές για τη συγγραφή καθαρού, ευανάγνωστου και συντηρήσιμου κώδικα, γνωστού και ως *Clean Code*, που διευκολύνει τη συνεργασία, τη συντήρηση και την εξέλιξη του λογισμικού.

Παραδοσιακά, η ποιότητα του κώδικα παρακολουθείται μέσω εργαλείων *στατικής ανάλυσης*, τα οποία εντοπίζουν ενδείξεις υποβάθμισης της ποιότητας, γνωστές ως *Code Smells*. Ο όρος αυτός αναφέρεται σε αποκλίσεις από καθιερωμένες βέλτιστες πρακτικές και λειτουργεί ως προειδοποίηση για πιθανά προβλήματα που απαιτούν την προσοχή του προγραμματιστή. Ορισμένα από αυτά τα φαινόμενα είναι ντετερμινιστικά και μπορούν να ανιχνευθούν αυτόματα, όπως στην περίπτωση του διπλότυπου κώδικα. Επιπλέον, η ανάλυση της ποιότητας του κώδικα μπορεί να υποστηριχθεί και από *δυναμικά εργαλεία ανάλυσης*, τα οποία αξιολογούν την εκτέλεση του λογισμικού, εντοπίζοντας σφάλματα χρόνου εκτέλεσης, προβλήματα απόδοσης και ανεπιθύμητες συμπεριφορές που δεν φαίνονται σε στατικό επίπεδο.

Πρόσφατα, η εμφάνιση των Μεγάλων Γλωσσικών Μοντέλων (Large Language Models, LLMs) έχει μετασχηματίσει σημαντικά το πεδίο της αξιολόγησης κώδικα, αναδεικνύοντάς τα ως ένα νέο, πολλά υποσχόμενο εργαλείο για την ανάλυση και εκτίμηση του πηγαίου κώδικα, χάρη στην ικανότητά τους να κατανοούν, να παράγουν και να ερμηνεύουν προγραμματιστικά μοτίβα μέσω φυσικής γλώσσας. Η ερευνητική δραστηριότητα επικεντρώνεται κυρίως στην εκτίμηση της ικανότητας των μοντέλων να κατανοούν τον κώδικα, καθώς και στην ικανότητά τους να εκτελούν συναφείς εργασίες, όπως η ανίχνευση σφαλμάτων, η αυτόματη συμπλήρωση, η μετάφραση και η περίληψη κώδικα. Ένα χαρακτηριστικό παράδειγμα τέτοιου πλαισίου αξιολόγησης αποτελεί το CodeXGLUE (Lu, 2021), ένα εκτενές σύνολο δεδομένων που χρησιμοποιείται για τη μέτρηση της απόδοσης των LLMs σε πλήθος γλωσσών προγραμματισμού και εργασιών ανάλυσης κώδικα. Παρά την πρόοδο αυτή, το πεδίο παραμένει περιορισμένο όσον αφορά την ικανότητα των LLMs να αξιολογούν την *ποιότητα του κώδικα*, δηλαδή να εκφέρουν κρίσεις σχετικά με την καθαρότητα, τη συντηρησιμότητα και τη συνολική τεχνική αρτιότητα του πηγαίου κώδικα.

Η παρούσα εργασία επικεντρώνεται στη διερεύνηση της δυνατότητας αξιολόγησης κώδικα μέσω του CodeLlama. Το μοντέλο χρησιμοποιήθηκε με κατάλληλα διαμορφωμένες προτροπές

(*prompts*) για την εκχώρηση βαθμολογίας από 0 έως 10, λαμβάνοντας υπόψη παράγοντες όπως η αναγνωσιμότητα, η δομή και η γενική ποιότητα του πηγαίου κώδικα. Η μελέτη στοχεύει στην ποσοτική εκτίμηση της αξιολόγησης που παρέχει το LLM, εξετάζοντας με ρεαλιστικό τρόπο τις δυνατότητες και τους περιορισμούς του CodeLlama ως εργαλείο υποβοήθησης της εκτίμησης της ποιότητας του κώδικα. Η διαδικασία αξιολόγησης πραγματοποιήθηκε με τρεις διαφορετικές προσεγγίσεις, οι οποίες διαφοροποιούνται ως προς το βάθος και τον σκοπό της ανάλυσης. Η πρώτη προσέγγιση αφορά μια γενική επισκόπηση του κώδικα, με στόχο την αναγνώριση προφανών σημείων προς βελτίωση και τη διατύπωση γενικών σχολίων. Η δεύτερη είναι μια πιο χαλαρή αξιολόγηση, η οποία επικεντρώνεται στις βασικές αρχές προγραμματισμού και στις αρχές του *Clean Code*, δίνοντας έμφαση στην αναγνωσιμότητα και τη συνοχή του κώδικα. Η τρίτη προσέγγιση είναι αυστηρότερη, καθώς περιλαμβάνει λεπτομερή ανάλυση για τον εντοπισμό λογικών σφαλμάτων, τον έλεγχο εισόδων, την αποδοτικότητα και τη διαχείριση εξαιρέσεων, παρέχοντας έτσι μια πιο τεχνικά ακριβή εκτίμηση της ποιότητας.

Το σύστημα που αναπτύχθηκε διαθέτει ευρύ φάσμα δυνατοτήτων και μπορεί να αξιοποιηθεί σε διαφορετικά περιβάλλοντα. Σε εκπαιδευτικό πλαίσιο, μπορεί να προσφέρει πολύτιμη ανατροφοδότηση σε φοιτητές/-τριες ή αρχάριους προγραμματιστές σχετικά με την ποιότητα του κώδικά τους. Σε επαγγελματικά περιβάλλοντα ανάπτυξης λογισμικού, μπορεί να υποστηρίξει τη διαδικασία ανασκόπησης κώδικα, βοηθώντας τις ομάδες να εντοπίσουν αντι-μοτίβα, αδυναμίες ασφάλειας και πιθανά σημεία βελτίωσης. Παράλληλα, μπορεί να χρησιμοποιηθεί για την αποτίμηση υπαρχόντων έργων λογισμικού, διευκολύνοντας την κατανόηση και τη συντήρησή τους. Μέσω της ενσωμάτωσης αυτών των δυνατοτήτων, οι προγραμματιστές μπορούν να εξοικονομήσουν σημαντικό χρόνο στη διαδικασία ανασκόπησης και ελέγχου ποιότητας, διασφαλίζοντας παράλληλα ότι οι εφαρμογές τους παραμένουν συνεπείς, ευανάγνωστες και αξιόπιστες. Επιπλέον, η χρήση των LLMs ως βοηθητικών μηχανισμών αξιολόγησης μπορεί να οδηγήσει στη διαμόρφωση νέων υβριδικών προσεγγίσεων που συνδυάζουν την ανθρώπινη κρίση με την υπολογιστική ανάλυση, ανοίγοντας τον δρόμο για πιο αποδοτικά και αντικειμενικά συστήματα αξιολόγησης ποιότητας λογισμικού.

ΚΕΦΑΛΑΙΟ 2: ΑΝΑΣΚΟΠΗΣΗ ΒΙΒΛΙΟΓΡΑΦΙΑΣ

Το παρόν κεφάλαιο παρουσιάζει μια ανασκόπηση της σύγχρονης βιβλιογραφίας που αφορά τη χρήση Μεγάλων Γλωσσικών Μοντέλων (LLMs) στην εκπαίδευση, την αυτόματη αξιολόγηση κώδικα προγραμματισμού και την ανασκόπηση κώδικα. Αρχικά εξετάζονται οι πρόσφατες εξελίξεις και οι τάσεις στη χρήση LLMs για την υποστήριξη της μαθησιακής διαδικασίας και την αυτοματοποίηση της αξιολόγησης κώδικα. Στη συνέχεια, αναλύονται οι ερευνητικές προσεγγίσεις που αξιοποιούν τέτοια μοντέλα για την παραγωγή σχολίων και τη βελτίωση της διαδικασίας ανασκόπησης κώδικα (code review). Τέλος, παρουσιάζονται τα σημαντικότερα εξειδικευμένα μοντέλα που έχουν αναπτυχθεί για κατανόηση, ανάλυση και παραγωγή κώδικα προγραμμάτων, καθώς και τα πλεονεκτήματα και οι περιορισμοί τους.

2.1 Μεγάλα Γλωσσικά Μοντέλα στην Εκπαίδευση και στην Αυτόματη Αξιολόγηση Ασκήσεων Προγραμματισμού

Τα Μεγάλα Γλωσσικά Μοντέλα (Large Language Models, LLMs) έχουν αποδείξει τη χρησιμότητά τους σε ένα ευρύ φάσμα εφαρμογών στην εκπαίδευση, συμπεριλαμβανομένης της εξατομικευμένης μάθησης και των ευφυών συστημάτων διδασκαλίας. Η αξιοποίησή τους επιτρέπει την ανάπτυξη εργαλείων που προσφέρουν προσαρμοσμένες μαθησιακές εμπειρίες, εξατομικευμένο εκπαιδευτικό περιεχόμενο και άμεση ανατροφοδότηση, ενισχύοντας τόσο την αποτελεσματικότητα της μάθησης όσο και την εμπλοκή των μαθητών.

Μερικές πρόσφατες έρευνες εξετάζουν τις προοπτικές των LLMs στην εκπαίδευση, εστιάζοντας στο ρόλο τους στην προσαρμοσμένη μάθηση και στην ανάπτυξη ευφυών εκπαιδευτικών συστημάτων (Gan, Qi, Wu & Lin, 2023). Στον τομέα της αυτόματης αξιολόγησης, οι Piech et al. (2015a) εισήγαγαν την τεχνική Deep Knowledge Tracing (DKT), η οποία βασίζεται σε νευρωνικά δίκτυα για τη μοντελοποίηση της μαθησιακής προόδου των φοιτητών. Αυτή η προσέγγιση αποτελεί θεμέλιο για τη χρήση της τεχνητής νοημοσύνης στη διαχείριση εκπαιδευτικών δεδομένων και στην πρόβλεψη της μελλοντικής απόδοσης των μαθητών. Επιπλέον, οι Piech et al. (2015b) παρουσίασαν μεθόδους εκμάθησης embeddings προγραμμάτων για τη διάδοση ανατροφοδότησης σε κώδικα, προάγοντας τη χρήση αναπαραστάσεων προγράμματος σε αυτόματη αξιολόγηση. Στην κατεύθυνση της αυτόματης βαθμολόγησης ασκήσεων προγραμματισμού, οι Messer, Brown, Kölling & Shi (2023) διεξήγαγαν συστηματική ανασκόπηση 121 ερευνών, εστιάζοντας στις τεχνικές που χρησιμοποιούνται, στην αξιοπιστία των συστημάτων και στα τεχνολογικά κενά, διαπιστώνοντας ότι οι περισσότερες πλατφόρμες επικεντρώνονται στην αξιολόγηση της ορθότητας του κώδικα μέσω τεχνικών unit testing και στατικών εργαλείων. Η ανάλυσή τους καταδεικνύει ότι πολυάριθμες πλατφόρμες αξιολογούν μόνο την ορθότητα του κώδικα, συχνά μέσω τεχνικών που εμπλέκουν νευρωνικά δίκτυα όπως RNNs, LSTMs ή encoder–decoder

μοντέλα (όπου αυτά εφαρμόζονται), καθώς και τεχνικές μηχανικής μάθησης όπως λογιστική παλινδρόμηση και clustering (Bhatia, Kohli & Singh, 2018; Gupta et al., 2017; Gupta, Kanade & Shevade, 2019). Εκτός αυτών, έχουν επίσης αναπτυχθεί αυτόματα συστήματα βαθμολόγησης που αξιολογούν την ορθότητα και την ποιότητα του κώδικα χρησιμοποιώντας μεθόδους black-box testing, με προκαθορισμένες δοκιμές, καθώς και white-box testing, με ανάλυση της εσωτερικής δομής του προγράμματος (Ahmed et al., 2018; Timotej, Martin & Ivan, 2017). Επιπλέον, έχουν προταθεί τεχνικές ανίχνευσης σφαλμάτων μέσω ανάλυσης εκτέλεσης οι οποίες χρησιμοποιούνται για τον εντοπισμό λογικών λαθών που διαφορετικά θα ήταν δύσκολο να εντοπιστούν (π.χ. Brumley et al., 2007; Day, Penumala & González-Sánchez, 2019). Τέλος, η έρευνα των Dunder, Lundborg, Wong & Viberg (2024) εξέτασε την ικανότητα των LLMs να επιλύουν προγραμματιστικές ασκήσεις στην πλατφόρμα Kattis, με τα αποτελέσματα να καταλήγουν στο ότι παρόλο που τα LLMs διαθέτουν σημαντικές δυνατότητες στην αυτοματοποιημένη αξιολόγηση, εξακολουθούν να υπάρχουν προκλήσεις όσον αφορά την ακρίβεια και τη συνέπεια της βαθμολόγησης.

2.2 Μεγάλα Γλωσσικά Μοντέλα στην Ανασκόπηση Κώδικα (Code Review)

Πέρα από την εκπαίδευση, τα LLMs χρησιμοποιούνται ολοένα και περισσότερο και στο πεδίο της ανασκόπησης και αξιολόγησης κώδικα (code review), το οποίο αποτελεί κρίσιμο στάδιο στη διαδικασία ανάπτυξης λογισμικού. Η ανασκόπηση κώδικα στοχεύει στην ανίχνευση σφαλμάτων, τη βελτίωση της συντηρησιμότητας και την ενίσχυση της συνοχής του λογισμικού, ωστόσο παραμένει μια διαδικασία που απαιτεί σημαντικό χρόνο και ανθρώπινη εμπειρία. Τα LLMs μπορούν να προσφέρουν αυτοματοποιημένες προτάσεις σχολίων και βελτιώσεων, μειώνοντας τον φόρτο εργασίας των ομάδων ανάπτυξης. Στην κατεύθυνση αυτή, η μελέτη των Li et al. (2022) παρουσίασε το σύστημα AUGER, το οποίο χρησιμοποιεί μοντέλα T5 για τη δημιουργία σχολίων ανασκόπησης βασισμένων σε δεδομένα έργων ανοικτού λογισμικού. Τα αποτελέσματα δείχνουν ότι η μέθοδος υπερέχει συγκριτικά με υπάρχουσες προσεγγίσεις ως προς την ποιότητα των παραγόμενων σχολίων, βάσει δεικτών όπως ο ROUGE-L. Αντίστοιχα, η έρευνα των Haider, Mostofa, Mosaddek, Iqbal & Ahmed (2024) διερεύνησε την επίδραση της προτροπής (prompting) και της εξειδίκευσης (fine-tuning) σε μεγάλα γλωσσικά μοντέλα για αυτόματη παραγωγή σχολίων code review. Η μελέτη χρησιμοποίησε τεχνικές όπως QLoRA σε ανοικτά μοντέλα και κατέληξε ότι η απόδοση βελτιώνεται σημαντικά όταν παρέχονται συμπληρωματικές πληροφορίες (π.χ. γράφοι κλήσεων, περιλήψεις συναρτήσεων).

Οι παραπάνω προσεγγίσεις υπογραμμίζουν ότι τα LLMs μπορούν να λειτουργήσουν ως ισχυρά υποστηρικτικά εργαλεία στην ανασκόπηση και αξιολόγηση κώδικα, ενισχύοντας τόσο τον εντοπισμό λαθών όσο και τη δημιουργία χρήσιμου σχολίου. Ωστόσο, παραμένουν ανοιχτά ζητήματα που αφορούν τη διαφάνεια, τη συνέπεια και την ερμηνευσιμότητα των προτάσεων που παράγουν τα μοντέλα — θέματα που απαιτούν περαιτέρω έρευνα και κατεύθυνση. Η

παρούσα εργασία αξιοποιεί αυτές τις εξελίξεις για να συνδυάσει τις δυνατότητες των LLMs στην αυτόματη αξιολόγηση, ανασκόπηση και ποιοτική ανάλυση κώδικα σε ένα ενιαίο, επεκτάσιμο πλαίσιο.

2.3 Εξειδικευμένα Μοντέλα για Προγραμματισμό

Τα τελευταία χρόνια έχουν αναπτυχθεί αρκετά μοντέλα τεχνητής νοημοσύνης με εξειδίκευση στην ανάλυση, κατανόηση και παραγωγή πηγαίου κώδικα. Τα μοντέλα αυτά μπορούν να κατανοήσουν τη λογική του κώδικα, να εντοπίσουν σφάλματα, να προτείνουν βελτιώσεις και να προσφέρουν χρήσιμες συμβουλές στους προγραμματιστές. Τα OpenAI GPT-4 και GPT-4o διαθέτουν εξαιρετική ικανότητα κατανόησης σύνθετων δομών κώδικα και χρησιμοποιούνται σε εργαλεία όπως το ChatGPT Code Interpreter, το GitHub Copilot Chat και το Azure AI, προσφέροντας αξιόπιστη ανάλυση και προτάσεις σε πολλές γλώσσες προγραμματισμού. Η οικογένεια CodeLlama της Meta, που χρησιμοποιήθηκε και στην παρούσα εργασία, αποτελεί ένα από τα πιο προηγμένα ανοιχτά μοντέλα, εκπαιδευμένο σε τεράστιο όγκο open-source δεδομένων, με εκδόσεις 7B, 13B και 34B. Το κύριο πλεονέκτημά του είναι ότι μπορεί να εκτελείται τοπικά μέσω πλατφορμών όπως το Ollama, χωρίς σύνδεση στο διαδίκτυο, εξασφαλίζοντας ασφάλεια και ιδιωτικότητα. Το Google Gemini (πρώην AlphaCode) εστιάζει στην κατανόηση και βελτιστοποίηση κώδικα, καθώς και στη δημιουργία αυτόματης τεκμηρίωσης και code synthesis. Παράλληλα, τα StarCoder και SantaCoder του HuggingFace αποτελούν ελαφρύτερα, επίσης ανοικτού κώδικα μοντέλα, τα οποία υποστηρίζουν πολλές γλώσσες και προσφέρουν ταχύτερη, αν και λιγότερο αναλυτική, αξιολόγηση. Από την άλλη πλευρά, το Claude της Anthropic έχει σχεδιαστεί με έμφαση στη φυσική γλώσσα και τη λεπτομερή επεξήγηση του κώδικα, παρέχοντας πολύ καθαρά σχόλια και ανάλυση λογικών δομών. Το Grok της xAI, που αναπτύχθηκε από τον Elon Musk, στοχεύει στην κατανόηση και αυτόματη διόρθωση κώδικα με υψηλή ταχύτητα, ενώ ενσωματώνεται ήδη στο περιβάλλον του X (Twitter). Τέλος, το DeepSeek αποτελεί μια νεότερη προσέγγιση που εστιάζει στη βελτιστοποίηση απόδοσης και την ανίχνευση σφαλμάτων σε μεγάλα συστήματα λογισμικού, χρησιμοποιώντας τεχνικές ενισχυμένης εκπαίδευσης (Reinforcement Learning). Ωστόσο, τα περισσότερα από αυτά τα συστήματα, όπως τα GPT, Claude, Grok και DeepSeek, δεν είναι ανοιχτού κώδικα και δεν μπορούν να εκτελεστούν τοπικά, σε αντίθεση με το CodeLlama, το οποίο αποτελεί μια ελεύθερη και προσαρμόσιμη λύση για ανάλυση και αξιολόγηση κώδικα χωρίς ανάγκη σύνδεσης στο cloud.

ΚΕΦΑΛΑΙΟ 3: ΜΕΘΟΔΟΛΟΓΙΑ

Το σύστημα που αναπτύξαμε λειτουργεί ως ένα εργαλείο ανάλυσης και αξιολόγησης κώδικα, το οποίο χρησιμοποιεί μεγάλα γλωσσικά μοντέλα (LLMs) για να εξετάσει την ποιότητα του κώδικα, να εντοπίσει πιθανά προβλήματα και να προτείνει βελτιώσεις. Η βασική του λειτουργία είναι να παίρνει τον κώδικα ενός προγράμματος, να το διαβάζει και να αναλύει πόσο καλά είναι γραμμένα, δίνοντας σχόλια και βαθμολογίες. Η αξιολόγηση γίνεται με χρήση Μεγάλων Γλωσσικών Μοντέλων (LLMs), που χρησιμοποιεί κανόνες και βέλτιστες πρακτικές προγραμματισμού για να κρίνει την ποιότητα του κώδικα. Το σύστημα μπορεί να χρησιμοποιηθεί σε πολλαπλά επίπεδα, από αρχάριους προγραμματιστές που θέλουν να βελτιώσουν τον κώδικά τους, μέχρι επαγγελματικές ομάδες ανάπτυξης λογισμικού που θέλουν να εξασφαλίσουν υψηλή ποιότητα στο λογισμικό τους. Η διαδικασία αξιολόγησης βασίζεται στη χρήση διαφορετικών προτροπών (prompts) που δίνονται στο LLM για να κάνει την ανάλυση του κώδικα. Ανάλογα με το είδος της αξιολόγησης που επιλέγεται, η προτροπή διαμορφώνεται διαφορετικά ώστε να κατευθύνει το μοντέλο να επικεντρωθεί σε συγκεκριμένες πτυχές της ποιότητας του κώδικα. Υπάρχουν τρεις κύριοι τρόποι αξιολόγησης, καθένας από τους οποίους χρησιμοποιεί διαφορετική προτροπή προς το μοντέλο:

1. **Αξιολόγηση ”General” (Ανασκόπηση κώδικα):** Σε αυτή την περίπτωση, το σύστημα ζητά από το μοντέλο να εξετάσει τον κώδικα και να δώσει γενικές συμβουλές βελτίωσης. Το ερώτημα που του δίνεται διατυπώνεται ως εξής: “Δες αυτόν τον κώδικα και δώσε προτάσεις βελτίωσης. Πρότεινε καλές πρακτικές προγραμματισμού, βελτίωσε την αναγνωσιμότητα και τη συντηρησιμότητα. Εντόπισε τυχόν λάθη ή κακές πρακτικές (code smells, anti-patterns) και πρότεινε αλλαγές. Αν δεν χρειάζεται βελτίωση, πες απλά ‘Δεν απαιτούνται αλλαγές’.“ Το μοντέλο, λοιπόν, διαβάζει τον κώδικα και επιστρέφει αναλυτικές παρατηρήσεις. Αν εντοπίσει αδυναμίες, προτείνει τροποποιήσεις και παραθέτει παραδείγματα κώδικα που δείχνουν πώς μπορεί να βελτιωθεί. Αν δεν υπάρχουν προβλήματα, απαντά ότι ο κώδικας είναι ήδη καλά γραμμένος.

2. **Αξιολόγηση “Loose” (Γενική βαθμολόγηση):** Αυτή η αξιολόγηση είναι πιο απλή και έχει σκοπό να δώσει μια αριθμητική ένδειξη για την ποιότητα του κώδικα, χωρίς πολλή ανάλυση. Η προτροπή που χρησιμοποιείται είναι η εξής: “Εξέτασε τις βασικές αρχές προγραμματισμού στον παρακάτω κώδικα και δώσε μια συνολική βαθμολογία στην κλίμακα 0-10.“ Σε αυτή την περίπτωση, το μοντέλο επιστρέφει μόνο έναν αριθμό, που εκφράζει τη γενική ποιότητα του κώδικα. Αν και δεν παρέχει αναλυτικά σχόλια, μπορεί να χρησιμοποιηθεί για μια γρήγορη σύγκριση μεταξύ διαφορετικών αρχείων ή εκδόσεων του κώδικα.

3. **Αξιολόγηση "Strict" (Αυστηρή ανάλυση):** Αυτή η επιλογή είναι η πιο λεπτομερής και απαιτητική. Το σύστημα ζητά από το μοντέλο να ελέγξει πολλές διαφορετικές παραμέτρους του κώδικα, όπως:
    - Λογικά λάθη: Αν υπάρχουν προβλήματα που θα προκαλέσουν σφάλματα κατά την εκτέλεση.
    - Αποδοτικότητα: Αν ο κώδικας είναι γρήγορος ή αν υπάρχει τρόπος να γίνει καλύτερος.
    - Ασφάλεια: Αν υπάρχουν κενά που θα μπορούσαν να οδηγήσουν σε παραβίαση.
    - Διαχείριση σφαλμάτων: Αν ο κώδικας περιέχει μηχανισμούς για την αποφυγή προβλημάτων κατά την εκτέλεση.

Η διαδικασία που ακολουθείται για τον συνδυασμό των παραπάνω βημάτων περιλαμβάνει μια αλληλουχία ενεργειών που ξεκινούν από την επιλογή της μεθόδου αξιολόγησης από τον χρήστη. Αρχικά, ο χρήστης επιλέγει μία από τις τρεις διαθέσιμες μεθόδους αξιολόγησης, ανάλογα με το επίπεδο λεπτομέρειας και αυστηρότητας που επιθυμεί. Στη συνέχεια, το σύστημα αποστέλλει τον κώδικα προς ανάλυση στο μοντέλο τεχνητής νοημοσύνης, συνοδευόμενο από την κατάλληλη προτροπή (prompt) που καθορίζει το πλαίσιο της αξιολόγησης. Το μοντέλο αναλύει τον παρεχόμενο κώδικα και παράγει μια απάντηση, η οποία μπορεί να περιλαμβάνει είτε ποσοτική βαθμολογία είτε αναλυτικά σχόλια και προτάσεις βελτίωσης. Τέλος, το αποτέλεσμα παρουσιάζεται στον χρήστη, ο οποίος μπορεί να το αξιοποιήσει προκειμένου να εντοπίσει αδυναμίες και να βελτιώσει την ποιότητα και τη δομή του κώδικα του.

Το σύστημα λειτουργεί σαν ένας έξυπνος βοηθός προγραμματισμού, ο οποίος όχι μόνο εντοπίζει προβλήματα στον κώδικα, αλλά και παρέχει χρήσιμες συμβουλές για το πώς μπορεί να γίνει καλύτερος. Η χρήση διαφορετικών προτροπών (prompts) δίνει στο μοντέλο τη δυνατότητα να κάνει ανάλυση του κώδικα με διαφορετικούς τρόπους, ανάλογα με τις ανάγκες του χρήστη. Είτε χρειάζεται απλώς μια αριθμητική εκτίμηση, είτε μια αναλυτική επεξήγηση των λαθών και προτάσεων βελτίωσης, το σύστημα προσαρμόζεται και παρέχει τις κατάλληλες πληροφορίες. Αυτή η μέθοδος επιτρέπει την αυτοματοποιημένη και αντικειμενική αξιολόγηση κώδικα, ενώ παράλληλα προσφέρει ένα χρήσιμο εργαλείο για την εκμάθηση και βελτίωση της ποιότητας του προγραμματισμού. Το σύστημα μπορεί να χρησιμοποιηθεί τόσο για εκπαιδευτικούς σκοπούς (για να καταλάβει κάποιος πού κάνει λάθος), όσο και για αξιολόγηση έτοιμου κώδικα σε επαγγελματικά έργα. Οι τρεις διαφορετικοί τρόποι δίνουν ευελιξία ανάλογα με το πόσο λεπτομερή αξιολόγηση χρειάζεται ο τελικός χρήστης. Στο πλαίσιο της παρούσας μελέτης, το σύστημα σχεδιάστηκε ώστε να παρέχει αυτοματοποιημένη διαχείριση, ανάλυση και αξιολόγηση αρχείων κώδικα, προσφέροντας μια ολοκληρωμένη ροή εργασίας από την είσοδο μέχρι την τελική αξιολόγηση.

ΚΕΦΑΛΑΙΟ 4: ΥΛΟΠΟΙΗΣΗ

Η παρούσα ενότητα περιγράφει την υλοποίηση του συστήματος αξιολόγησης κώδικα που αναπτύχθηκε στο πλαίσιο της εργασίας. Το σύστημα βασίζεται στο Ollama, μια πλατφόρμα ανοιχτού κώδικα η οποία επιτρέπει την εκτέλεση Μεγάλων Γλωσσικών Μοντέλων (Large Language Models – LLMs) τοπικά στον υπολογιστή του χρήστη. Για την υλοποίηση μας χρησιμοποιήσαμε το μοντέλο CodeLlama, το οποίο προσφέρει εξειδικευμένες δυνατότητες κατανόησης και ανάλυσης πηγαίου κώδικα.

4.1 Τεχνολογικό υπόβαθρο και βιβλιοθήκες

Η υλοποίηση του συστήματος πραγματοποιήθηκε εξολοκλήρου στη γλώσσα Python, αξιοποιώντας τη βιβλιοθήκη `langchain_ollama` για την επικοινωνία με το μοντέλο CodeLlama μέσω του Ollama. Η αρχιτεκτονική του συστήματος ακολουθεί μια αυτοματοποιημένη ροή εργασίας, η οποία επιτρέπει στον χρήστη να εισάγει έναν φάκελο που περιέχει αρχεία ZIP. Το σύστημα αποσυμπιέζει τα αρχεία, εντοπίζει τα αρχεία Python και τα υποβάλλει αυτόματα για αξιολόγηση από το μοντέλο.

4.2 Χρήση της πλατφόρμας Ollama

Το Ollama επιλέχθηκε ως τεχνολογική βάση του συστήματος λόγω των χαρακτηριστικών του που το καθιστούν ιδανικό για τοπική, αποδοτική και ασφαλή εκτέλεση LLMs. Παρέχει τη δυνατότητα λειτουργίας χωρίς σύνδεση στο Διαδίκτυο, εξασφαλίζοντας ότι τα δεδομένα και οι αλληλεπιδράσεις με το μοντέλο παραμένουν στο τοπικό περιβάλλον του χρήστη. Η επικοινωνία με το μοντέλο πραγματοποιείται μέσω απλών εντολών, γεγονός που διευκολύνει την εναλλαγή μεταξύ διαφορετικών μοντέλων (π.χ. Llama2, Mistral) ανάλογα με τις ανάγκες της ανάλυσης.

Επιπλέον, η πλατφόρμα υποστηρίζει επιτάχυνση μέσω GPU, επιτυγχάνοντας υψηλή απόδοση κατά την εκτέλεση. Στην παρούσα εργασία, το Ollama χρησιμοποιείται ως backend μηχανισμός επεξεργασίας, στον οποίο αποστέλλεται ο πηγαίος κώδικας προς αξιολόγηση. Ο χρήστης μπορεί να επιλέξει τον τύπο αξιολόγησης που επιθυμεί – *Review*, *Loose* ή *Strict* – και η αντίστοιχη προτροπή (prompt) ενσωματώνεται αυτόματα στο μήνυμα που αποστέλλεται στο LLM μέσω του API του Ollama. Η επιλογή αυτής της πλατφόρμας προσφέρει μια αποδοτική και αυτοδύναμη λύση, εξασφαλίζοντας τη δυνατότητα επέκτασης και προσαρμογής για μελλοντικές εκδόσεις ή άλλα μοντέλα.

4.3 Δομή κώδικα και σχεδιαστικές αποφάσεις

Ο κεντρικός κορμός του συστήματος βρίσκεται στο αρχείο `main.py`, το οποίο οργανώνεται σε τρεις κύριες ενότητες: (α) την αξιολόγηση κώδικα, (β) τη διαχείριση αρχείων ZIP και (γ) τη ροή εκτέλεσης του προγράμματος.

4.3.1 Αξιολόγηση κώδικα

Η συνάρτηση `evaluate_code(option, code)` είναι υπεύθυνη για την αποστολή του πηγαίου κώδικα στο μοντέλο **CodeLlama** μέσω του Ollama. Ανάλογα με την επιλογή του χρήστη, ενεργοποιείται μία από τις τρεις προδιαμορφωμένες προτροπές:

- **General:** πραγματοποιεί μια γενική ανασκόπηση του κώδικα, εντοπίζοντας προβλήματα αναγνωσιμότητας και πιθανά "code smells", και προτείνει βελτιώσεις.
- **Loose:** εκτελεί βασική αξιολόγηση, αποδίδοντας αριθμητική βαθμολογία από 0 έως 10 χωρίς εκτενείς σχολιασμούς.
- **Strict:** προβαίνει σε λεπτομερή τεχνική ανάλυση, ελέγχοντας για λογικά λάθη, ζητήματα ασφάλειας, αποδοτικότητας και διαχείρισης εξαιρέσεων, και παρέχει πλήρη αναφορά με τελική βαθμολογία.

Η επικοινωνία με το LLM επιτυγχάνεται μέσω του API του Ollama, ενώ το αποτέλεσμα επιστρέφεται και εμφανίζεται άμεσα στο τερματικό του χρήστη. Ο σχεδιασμός αυτός επιτρέπει την εύκολη ενσωμάτωση νέων τύπων αξιολόγησης ή διαφορετικών μοντέλων χωρίς σημαντικές τροποποιήσεις στον κώδικα.

4.3.2 Διαχείριση αρχείων ZIP

Η συνάρτηση `extract_and_return_all_zip_files_data_from_path()` εντοπίζει όλα τα αρχεία ZIP μέσα στον φάκελο εισόδου, τα αποσυμπιέζει προσωρινά και αναζητά τα αρχεία με κατάληξη `.py`. Για κάθε αρχείο Python, διαβάζει το περιεχόμενο και το επιστρέφει σε μορφή λεξικού (dictionary), όπου κάθε κλειδί αντιστοιχεί στο όνομα του αρχείου και η τιμή στο περιεχόμενό του. Η λειτουργία αυτή επιτρέπει την επεξεργασία ολόκληρων έργων ή πολλαπλών scripts μέσα σε μία μόνο εκτέλεση, εξασφαλίζοντας αυτοματοποίηση και ευελιξία στη χρήση.

4.3.3 Κύρια ροή εκτέλεσης

Το κύριο τμήμα του προγράμματος διαχειρίζεται τις παραμέτρους που δίνονται από τον χρήστη κατά την εκτέλεση. Το πρώτο όρισμα καθορίζει το είδος αξιολόγησης (*strict*, *loose* ή *review*), ενώ το δεύτερο προσδιορίζει το μονοπάτι προς τον φάκελο που περιέχει τα αρχεία

ZIP. Το πρόγραμμα ανακτά τα δεδομένα μέσω της συνάρτησης εξαγωγής, τα αναλύει και τα προωθεί στο μοντέλο για αξιολόγηση. Η έξοδος που παράγεται περιλαμβάνει, ανάλογα με τον τύπο της προτροπής, είτε αναλυτικές παρατηρήσεις είτε αριθμητικές βαθμολογίες.

4.4 Λειτουργία του συστήματος

Η λειτουργία του συστήματος είναι απλή και αυτοματοποιημένη. Ο χρήστης τοποθετεί τα αρχεία ZIP σε έναν φάκελο και εκτελεί το πρόγραμμα δίνοντας ως παραμέτρους τον τύπο αξιολόγησης και το μονοπάτι του φακέλου, για παράδειγμα:

```
python main.py strict ./test_zips_folder
```

Το σύστημα εντοπίζει αυτόματα τα αρχεία, τα αποσυμπιέζει, εντοπίζει τα αρχεία Python και τα προωθεί στο μοντέλο CodeLlama μέσω του Ollama. Το μοντέλο επιστρέφει είτε αναλυτικές παρατηρήσεις (Review), είτε γενική βαθμολογία (Loose), είτε αυστηρή τεχνική αξιολόγηση (Strict). Η διαδικασία ολοκληρώνεται σε λίγα δευτερόλεπτα, χωρίς να απαιτείται σύνδεση στο διαδίκτυο.

4.5 Πλεονεκτήματα και αξία της υλοποίησης

Η χρήση του Ollama προσφέρει σημαντικά πλεονεκτήματα, όπως τοπική εκτέλεση χωρίς αποστολή δεδομένων στο cloud, ενισχύοντας την ιδιωτικότητα και την ασφάλεια των δεδομένων. Επιπλέον, επιτρέπει τη χρήση διαφορετικών LLMs με ελάχιστες αλλαγές, καθώς και την κλιμάκωση της λύσης για μεγαλύτερα ή πιο εξειδικευμένα μοντέλα στο μέλλον.

Η αρχιτεκτονική του συστήματος συμβάλλει στην αυτοματοποίηση της διαδικασίας συλλογής και ανάλυσης κώδικα, προσφέρει ευελιξία στην επιλογή τύπου αξιολόγησης και διαθέτει σημαντική εκπαιδευτική αξία, παρέχοντας ανατροφοδότηση σε φοιτητές ή αρχάριους προγραμματιστές. Παράλληλα, μπορεί να αξιοποιηθεί και σε επαγγελματικά περιβάλλοντα ανάπτυξης, υποστηρίζοντας τη διαδικασία ανασκόπησης κώδικα και τον ποιοτικό έλεγχο μεγάλων έργων. Η συνδυασμένη χρήση τοπικής εκτέλεσης, αυτοματοποίησης και αξιοποίησης LLMs καθιστά το σύστημα ένα ευέλικτο και αξιόπιστο σύστημα αξιολόγησης κώδικα προγραμμάτων.

ΚΕΦΑΛΑΙΟ 5: ΠΑΡΑΔΕΙΓΜΑΤΑ ΧΡΗΣΗΣ

Για την πληρέστερη κατανόηση της λειτουργίας και των δυνατοτήτων του συστήματος, το παρόν κεφάλαιο παρουσιάζει ενδεικτικά παραδείγματα χρήσης του για την αξιολόγηση κώδικα. Σκοπός είναι να αναδειχθεί ο τρόπος με τον οποίο το σύστημα μας επεξεργάζεται και αξιολογεί διαφορετικά παραδείγματα κώδικα.

Κάθε παράδειγμα αξιολογείται με έναν από τους τρεις διαθέσιμους τρόπους λειτουργίας του συστήματος, ανάλογα με το επιθυμητό βάθος ανάλυσης και το επίπεδο λεπτομέρειας που ζητείται. Η λειτουργία **General** εστιάζει στη γενική ανασκόπηση και τη διατύπωση προτάσεων βελτίωσης, η λειτουργία **Loose** παρέχει μια συνοπτική ποσοτική εκτίμηση της ποιότητας του κώδικα, ενώ η λειτουργία **Strict** πραγματοποιεί ενδελεχή τεχνική αξιολόγηση, ελέγχοντας λογικά σφάλματα, ζητήματα αποδοτικότητας και ορθότητα διαχείρισης εξαιρέσεων.

5.1 Υπολογισμος αθροίσματος δύο αριθμών

Το πρώτο παράδειγμα αποτελεί ένα απλό πρόγραμμα σε γλώσσα Python, το οποίο ζητά από τον χρήστη την εισαγωγή δύο αριθμών και στη συνέχεια εμφανίζει το άθροισμά τους. Παρότι το πρόγραμμα είναι ιδιαίτερα απλό, χρησιμεύει για να επιδειχθεί η βασική λειτουργία του συστήματος αξιολόγησης, καθώς επιτρέπει την παρατήρηση του τρόπου με τον οποίο το μοντέλο CodeLlama εντοπίζει στοιχειώδη προγραμματιστικά σφάλματα και παρέχει σχόλια σχετικά με τη σωστή χρήση τύπων δεδομένων και τη βασική λογική του προγράμματος. Μέσα από αυτό το παράδειγμα καθίσταται εμφανής η ικανότητα του συστήματος να αξιολογεί ακόμη και απλές περιπτώσεις κώδικα, εντοπίζοντας προβλήματα που σχετίζονται με την αναγνωσιμότητα ή τη λειτουργική ακρίβεια του κώδικα.

```
a = input("Δώσε τον πρώτο αριθμό: ")
b = input("Δώσε τον δεύτερο αριθμό: ")

print("Το άθροισμα είναι:", a + b)
```

Αξιολόγηση με εντολή “general”:

- Το σύστημα επισημαίνει ότι το πρόγραμμα δεν μετατρέπει τις εισόδους σε αριθμούς (λειτουργούν ως κείμενο).
- Προτείνει τη χρήση int() για μετατροπή και τον έλεγχο σφαλμάτων με try-except. Παράδειγμα που προτείνει:

```
try:
    a = int(input(“Δώσε τον πρώτο αριθμό: “))
```

```
        b = int(input("Δώσε τον δεύτερο αριθμό: "))
        print("Το άθροισμα είναι:", a + b)
    except ValueError:
        print("Παρακαλώ δώσε έγκυρους αριθμούς.")
```

Αξιολόγηση με εντολή "loose":

- Δίνει μια βαθμολογία, π.χ. 3/10.
- Σχόλιο: "Ο κώδικας λειτουργεί αλλά έχει βασικά λάθη – δεν γίνεται έλεγχος εισόδου και το αποτέλεσμα μπορεί να είναι λάθος."

Αξιολόγηση με εντολή "strict" (Παρέχει λεπτομερή έλεγχο):

- Δεν υπάρχει μετατροπή σε αριθμούς.
- Δεν γίνεται διαχείριση εξαιρέσεων.
- Δεν υπάρχει σαφής δομή ή επεξήγηση.
- Τελική βαθμολογία: 2/10

5.2 Συνάρτηση υπολογισμού παραγοντικού

Το δεύτερο παράδειγμα παρουσιάζει μια κλασική υλοποίηση της συνάρτησης παραγοντικού σε γλώσσα Python. Η συνάρτηση `factorial(n)` υπολογίζει το γινόμενο όλων των θετικών αριθμών μέχρι το `n`, χρησιμοποιώντας αναδρομή. Το συγκεκριμένο παράδειγμα επιλέχθηκε επειδή επιτρέπει στο σύστημα αξιολόγησης να εξετάσει πτυχές που σχετίζονται με την ορθότητα της λογικής, τη δομή του κώδικα και την αποδοτικότητα της αναδρομικής κλήσης. Επιπλέον, δίνει τη δυνατότητα στο μοντέλο CodeLlama να αναγνωρίσει ζητήματα που αφορούν πιθανές βελτιώσεις, όπως η απουσία σχολίων τεκμηρίωσης ή η ενσωμάτωση ελέγχου εγκυρότητας των εισόδων. Μέσω αυτού του παραδείγματος αξιολογείται η ικανότητα του συστήματος να κατανοεί πιο σύνθετες ροές ελέγχου και να αποδίδει εύστοχη ανατροφοδότηση σε προγράμματα με αναδρομική λογική.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Αξιολόγηση με εντολή "general":

- Αναγνωρίζει ότι η λογική είναι σωστή, αλλά λείπει ο έλεγχος για αρνητικούς αριθμούς.
- Προτείνει βελτίωση με προσθήκη docstring και έλεγχο εισόδου.

Αξιολόγηση με εντολή "loose":

- Βαθμολογία: 7/10
- Σχόλιο: "Η λογική είναι σωστή, αλλά δεν είναι πλήρης – χρειάζεται έλεγχος τιμών και σχόλια."

Αξιολόγηση με εντολή "strict" (Παρέχει λεπτομερή έλεγχο):

- Η συνάρτηση μπορεί να προκαλέσει αναδρομικό σφάλμα για μεγάλες τιμές.
- Δεν καλύπτονται όλες οι περιπτώσεις εισόδου.
- Τελική βαθμολογία: 5/10

5.3 Συνάρτηση διαίρεσης με έλεγχο σφάλματος μηδενικού διαιρέτη

Το τρίτο παράδειγμα αφορά μια απλή αλλά ιδιαίτερα σημαντική περίπτωση προγραμματισμού, καθώς επικεντρώνεται στη *διαχείριση εξαιρέσεων* κατά τη διάρκεια εκτέλεσης. Η συνάρτηση `divide(a, b)` επιχειρεί να διαιρέσει τον αριθμό `a` με τον `b`, επιστρέφοντας το αποτέλεσμα της πράξης. Σε περίπτωση που ο δεύτερος αριθμός είναι μηδέν, η συνάρτηση ενεργοποιεί έναν μηχανισμό χειρισμού εξαίρεσης (**`try-except`**) και επιστρέφει κατάλληλο μήνυμα σφάλματος. Το παράδειγμα αυτό αναδεικνύει τη δυνατότητα του συστήματος αξιολόγησης να εντοπίζει ζητήματα που σχετίζονται με την *αξιοπιστία* και τη *σταθερότητα* του κώδικα, όπως η ορθή χρήση μηχανισμών χειρισμού λαθών και η διασφάλιση της ομαλής εκτέλεσης προγραμμάτων υπό μη αναμενόμενες συνθήκες. Παράλληλα, προσφέρει στο μοντέλο CodeLlama τη δυνατότητα να αξιολογήσει την τεκμηρίωση, τη σαφήνεια της συνάρτησης και την επάρκεια των ελέγχων ασφαλείας, παρέχοντας μια πιο τεχνικά εστιασμένη ανατροφοδότηση σε σχέση με τα προηγούμενα παραδείγματα.

```
def divide(a, b):
    """Διαιρεί το a με το b, επιστρέφει λάθος αν b είναι 0"""
    try:
        return a / b
    except ZeroDivisionError:
        return "Δεν μπορείς να διαιρέσεις με το μηδέν."
```

Αξιολόγηση με εντολή "general":

- Το σύστημα απαντάει: "No changes needed."
- Αναγνωρίζει ότι ο κώδικας είναι καθαρός, σωστά δομημένος και περιέχει χειρισμό λαθών.

Αξιολόγηση με εντολή "loose":

- Βαθμολογία: 9/10
- Σχόλιο: "Ο κώδικας είναι σωστά γραμμένος, με καλή διαχείριση σφαλμάτων και τεκμηρίωση."

Αξιολόγηση με εντολή "strict" (Παρέχει λεπτομερή έλεγχο):

- Εξετάζει τη σαφήνεια, την τεκμηρίωση και τον έλεγχο εξαιρέσεων.
- Δεν βρίσκει σοβαρά μειονεκτήματα.
- Βαθμολογία: 9.5/10

5.4 Πρόγραμμα αθροίσματος θετικών αριθμών

Το τελευταίο παράδειγμα παρουσιάζει ένα απλό πρόγραμμα σε γλώσσα Python που υπολογίζει το άθροισμα θετικών αριθμών που εισάγει ο χρήστης. Το πρόγραμμα διαβάζει επαναλαμβανόμενα τιμές μέσω εντολής εισόδου και, όσο οι τιμές είναι διαφορετικές από το μηδέν, ελέγχει αν είναι θετικές και τις προσθέτει στο τρέχον άθροισμα. Η επαναληπτική διαδικασία τερματίζεται όταν ο χρήστης εισαγάγει την τιμή 0, οπότε και εμφανίζεται το συνολικό άθροισμα. Το συγκεκριμένο παράδειγμα επιτρέπει την αξιολόγηση της *λογικής ροής* και της *δομής ελέγχου* ενός βασικού προγράμματος που περιλαμβάνει βρόχο επανάληψης και συνθήκες επιλογής. Παράλληλα, δίνει τη δυνατότητα στο σύστημα να εντοπίσει πιθανές ελλείψεις, όπως η απουσία σχολίων τεκμηρίωσης ή η χρήση μη περιγραφικών ονομάτων μεταβλητών, και να διατυπώσει προτάσεις βελτίωσης σχετικά με την αναγνωσιμότητα και τη σαφήνεια του κώδικα. Μέσα από αυτό το παράδειγμα αξιολογείται η ικανότητα του μοντέλου CodeLlama να κατανοεί τη ροή προγραμμάτων που συνδυάζουν δομές επανάληψης και ελέγχου, παρέχοντας εποικοδομητική και προσαρμοσμένη ανατροφοδότηση.

```
timh = int(input("δωσε μια τιμη: "))
sum = 0
while timh != 0:
    if timh > 0:
        sum = sum + timh
    timh = int(input("ξανα δωσε μια τιμη: "))
print(sum)
```

Αξιολόγηση με εντολή "general":

- Αναγνωρίζει ότι η λογική είναι σωστή, αλλά λείπει ο έλεγχος σε μη έγκυρα δεδομένα (δηλαδή, μη ακέραιες τιμές)
- Αναγνωρίζει ότι ο κώδικας είναι καθαρός, αλλά προτείνει βελτίωση με προσθήκη στον έλεγχο εισόδου.

Αξιολόγηση με εντολή "loose":

- Βαθμολογία: 9/10
- Σχόλιο: "Ο κώδικας υπολογίζει σωστά το άθροισμα των εισαγόμενων αριθμών, αλλά δεν χειρίζεται σωστά τα μη έγκυρα δεδομένα (δηλαδή, μη ακέραιες τιμές)."

Αξιολόγηση με εντολή "strict" (Παρέχει λεπτομερή έλεγχο):

- Βαθμολογία: 8/10
- Σχόλιο: “Συνολικά, ο κώδικας έχει ορισμένα ζητήματα ορθότητας και μπορεί να βελτιωθεί όσον αφορά την επικύρωση εισόδου, τη δυνατότητα συντήρησης και τη διαχείριση σφαλμάτων. Ωστόσο, εξακολουθεί να είναι μια έγκυρη λύση στη δήλωση του προβλήματος.”
- Σχόλιο: “Λογικά λάθη: 5/10 - Ο κώδικας έχει κάποια λογικά σφάλματα που πρέπει να διορθωθούν. Για παράδειγμα, η συνθήκη “if timh > 0:” δεν είναι απαραίτητη, επειδή ο βρόχος θα εκτελείται μόνο όταν το “timh” δεν είναι μηδενικό. Επιπλέον, η μεταβλητή “sum” θα πρέπει να αρχικοποιηθεί στο μηδέν πριν από την έναρξη του βρόχου για να αποφευχθεί η συσσώρευση προηγούμενων αθροισμάτων.”

ΚΕΦΑΛΑΙΟ 6: ΣΥΜΠΕΡΑΣΜΑΤΑ

Η παρούσα εργασία επικεντρώθηκε στη διερεύνηση της δυνατότητας αξιολόγησης πηγαίου κώδικα με τη χρήση Μεγάλων Γλωσσικών Μοντέλων (LLMs), και ειδικότερα του CodeLlama. Η ανάπτυξη του συστήματος σε Python, με την ενσωμάτωση της βιβλιοθήκης langchain_ollama, επέτρεψε τη δημιουργία ενός πλήρως αυτοματοποιημένου συστήματος που μπορεί να διαχειρίζεται φακέλους έργων, να αναγνωρίζει αρχεία κώδικα και να εκτελεί διαφορετικού τύπου αξιολογήσεις, ανάλογα με τις ανάγκες του χρήστη. Η αρχιτεκτονική σχεδιάστηκε ώστε να υποστηρίζει τρεις βαθμίδες ανάλυσης – *Review*, *Loose* και *Strict* – επιτρέποντας τόσο επιφανειακή όσο και εις βάθος τεχνική αξιολόγηση. Η επιλογή της πλατφόρμας Ollama αποδείχθηκε στρατηγικά σημαντική, καθώς εξασφαλίζει τοπική εκτέλεση των μοντέλων, χαμηλό χρόνο απόκρισης, υψηλό επίπεδο ιδιωτικότητας και δυνατότητα επέκτασης με εναλλακτικά LLMs. Σε *εκπαιδευτικό επίπεδο*, το προτεινόμενο σύστημα μπορεί να αξιοποιηθεί ως σύστημα ανατροφοδότησης για φοιτητές ή νέους προγραμματιστές, βοηθώντας τους να κατανοήσουν τα δυνατά και αδύνατα σημεία του κώδικά τους. Σε *επαγγελματικό περιβάλλον*, μπορεί να υποστηρίξει τη διαδικασία ανασκόπησης και ελέγχου ποιότητας, εντοπίζοντας γρήγορα προβλήματα ή αποκλίσεις από τις βέλτιστες πρακτικές προγραμματισμού.

Τα *παραδείγματα χρήσης* που παρουσιάστηκαν ανέδειξαν τη λειτουργικότητα και την προσαρμοστικότητα του συστήματος. Στα απλά προγράμματα, όπως ο υπολογισμός αθροίσματος ή η αναδρομική συνάρτηση παραγοντικού, το μοντέλο παρείχε ακριβή και χρήσιμη ανατροφοδότηση σχετικά με τη δομή και τη λογική του κώδικα. Σε πιο σύνθετες περιπτώσεις, όπως ο χειρισμός εξαιρέσεων ή οι δομές επανάληψης, το σύστημα επέδειξε ικανότητα εντοπισμού σφαλμάτων και παροχής τεκμηριωμένων προτάσεων βελτίωσης. Τα αποτελέσματα επιβεβαιώνουν ότι τα LLMs μπορούν να κατανοήσουν ουσιαστικά τον πηγαίο κώδικα, να αξιολογήσουν την ποιότητά του και να εκφέρουν κρίσεις που είναι συγκρίσιμες με αυτές ενός έμπειρου προγραμματιστή σε βασικό επίπεδο. Ωστόσο, η μελέτη ανέδειξε και ορισμένους *περιορισμούς*. Η αξιολόγηση από τα LLMs παραμένει σε σημαντικό βαθμό *υποκειμενική*, καθώς εξαρτάται από τη διατύπωση των προτροπών (prompts) και τη συνεκτικότητα των απαντήσεων του μοντέλου. Επιπλέον, η βαθμολόγηση δεν βασίζεται πάντοτε σε απόλυτα ποσοτικά κριτήρια, γεγονός που μπορεί να οδηγήσει σε μικρές αποκλίσεις ή ασυνέπειες μεταξύ εκτελέσεων. Παρά ταύτα, η συνολική απόδοση του συστήματος είναι ενθαρρυντική, ιδίως αν ληφθεί υπόψη η απλότητα και η αυτοδυναμία της προσέγγισης.

Συνοψίζοντας, η εργασία απέδειξε ότι η αξιοποίηση των LLMs για την εκτίμηση της ποιότητας πηγαίου κώδικα αποτελεί μια *καινοτόμο, πρακτική και επεκτάσιμη προσέγγιση*. Παρότι υπάρχουν ακόμη προκλήσεις που αφορούν τη συνέπεια και την ερμηνευσιμότητα των αποτελεσμάτων, τα πειραματικά ευρήματα καταδεικνύουν ότι τα μοντέλα αυτού του τύπου

μπορούν να λειτουργήσουν ως πολύτιμα εργαλεία υποστήριξης στην εκπαίδευση, την έρευνα και την ανάπτυξη λογισμικού, ανοίγοντας τον δρόμο για περαιτέρω εξελίξεις στην αυτόματη αξιολόγηση κώδικα με τη βοήθεια τεχνητής νοημοσύνης.

Μελλοντικά, η εργασία αυτή μπορεί να επεκταθεί προς διάφορες κατευθύνσεις. Ενδεικτικά, η ενσωμάτωση *ποσοτικών μετρικών ποιότητας λογισμικού* (όπως μέτρα πολυπλοκότητας, coupling και cohesion) θα μπορούσε να ενισχύσει τη συνέπεια της βαθμολόγησης. Παράλληλα, η χρήση *fine-tuning* σε εξειδικευμένα σύνολα δεδομένων εκπαιδευτικών ή επαγγελματικών εργασιών θα επέτρεπε στο μοντέλο να προσαρμόζεται καλύτερα στο εκάστοτε πεδίο χρήσης. Τέλος, η ανάπτυξη *υβριδικών προσεγγίσεων*, που θα συνδυάζουν την κρίση ενός LLM με στατικές και δυναμικές αναλύσεις κώδικα, θα μπορούσε να οδηγήσει σε ένα πιο αντικειμενικό και αξιόπιστο πλαίσιο αξιολόγησης.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ahmed, U.Z., Kumar, P., Karkare, A., Kar, P. & Gulwani, S. (2018) 'Compilation error repair: for the student programs, from the student programs', *Proceedings of the International Conference on Software Engineering*, pp. 78–87.

Bengtsson, D. & Kaliff, A. (2023) *Assessment accuracy of a large language model on programming assignments*, Degree project in Computer Science and Engineering, KTH Royal Institute of Technology.

Bhatia, S., Kohli, P. & Singh, R. (2018) 'Neuro-symbolic program corrector for introductory programming assignments', *Proceedings of the International Conference on Software Engineering*, pp. 60–70.

Brumley, D., Caballero, J., Liang, Z., Newsome, J. & Song, D. (2007) 'Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation', *USENIX Security Symposium*, vol. 15, pp. 1–16.

Day, M., Penumala, M.R. & Gonzalez-Sanchez, J. (2019) 'Annete: An Intelligent Tutoring Companion Embedded into the Eclipse IDE', *IEEE First International Conference on Cognitive Machine Intelligence*, pp. 71–80 (or appropriate page numbers).

Dunder, N., Lundborg, S., Wong, J. & Viberg, O. (2024) 'Kattis versus ChatGPT: Assessment and Evaluation of Programming Tasks in the Age of Artificial Intelligence', *Proceedings of the 14th Learning Analytics and Knowledge Conference (LAK '24)*, ACM, pp. 821–827.

Gan, W., Qi, Z., Wu, J. & Lin, J.C.W. (2023) 'Large Language Models in Education: Vision and Opportunities', *2023 IEEE International Conference on Big Data (BigData)*, Los Alamitos, CA: IEEE, pp. 4776–4785.

Gupta, R., Kanade, A. & Shevade, S. (2019) 'Deep reinforcement learning for syntactic error repair in student programs', *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 930– (page number to verify).

Gupta, R., Pal, S., Kanade, A. & Shevade, S. (2017) 'DeepFix: Fixing Common C Language Errors by Deep Learning', *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1345– (page number to verify).

Li, L., Yang, L., Jiang, H., Yan, J., Luo, T., Hua, Z. & Zuo, C. (2022) 'AUGER: Automatically Generating Review Comments with Pre-training Models', *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1009–1021.

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., ... & Liu, S. (2021). Codexglue: A machine learning benchmark dataset for code understanding and generation. arXiv preprint arXiv:2102.04664.

Haider, M.A., Mostofa, A.B., Mosaddek, S.S.B., Iqbal, A. & Ahmed, T. (2024) 'Prompting and fine-tuning large language models for automated code review comment generation', *arXiv preprint arXiv:2411.10129*.

Messer, M., Brown, N.C.C., Kölling, M. & Shi, M. (2023) 'Automated grading and feedback tools for programming education: a systematic review', *ACM Transactions on Computing Education*, vol. 1, no. 1 (or verify actual volume/issue) [also available as arXiv preprint].

Paiva, J.C., Leal, P. & Figueira, Á. (2022) 'Automated assessment in computer science education: a state-of-the-art review', *ACM Transactions on Computing Education*, vol. 22, no. 3, pp. 1–40.

Piech, C., Bassen, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L. & Sohl-Dickstein, J. (2015a) 'Deep Knowledge Tracing', *Advances in Neural Information Processing Systems*, pp. 505–513.

Piech, C., Huang, J., Nguyen, A., Phulsuksombati, M., Sahami, M. & Guibas, L. (2015b) 'Learning program embeddings to propagate feedback on student code', *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1093–1101.

Timotej, L., Martin, M. & Ivan, B. (2017) 'Automatic extraction of AST patterns for debugging student programs', *Lecture Notes in Computer Science*, vol. 10331.

ΠΑΡΑΡΤΗΜΑ Α

Παρακάτω παρουσιάζεται η υλοποίηση του συστήματός μας, το οποίο πραγματοποιεί αυτόματη αξιολόγηση και ανάλυση αρχείων Python που περιέχονται σε συμπιεσμένα αρχεία (.zip). Το σύστημα αξιοποιεί το μοντέλο CodeLlama μέσω της βιβλιοθήκης LangChain Ollama, προκειμένου να εκτελεί έλεγχο ποιότητας, ανάλυση βέλτιστων πρακτικών και απονομή βαθμολογίας βάσει διαφορετικών προφίλ αξιολόγησης (strict, loose, general).

```
from langchain_ollama import OllamaLLM
from sys import argv
import os, zipfile

llm = OllamaLLM(model="codellama")


def evaluate_code(option, code):
    messages = [
        # messages[0]
        (
            code
            + "\n Review this code, provide suggestions for improvement, "
            "coding best practices, improve readability, and maintainability. "
            "Remove any code smells and anti-patterns. Provide code examples "
            "for your suggestions. Respond in markdown format. If the file "
            "does not have any code or does not need any changes, say "
            "'No changes needed'."
        ),
        # messages[1]
        (
            "Review the basic programming principles of the following code and "
            "output an overall score in the range 0–10:\n" + code
        ),
        # messages[2]
        (
            code
            + "\n Review this code, assess the correctness, check for logical "
            "errors. Evaluate input validation, maintainability, readability, "
            "organization, performance, efficiency, and safety. Check if error "
            "handling is used for unexpected inputs or exceptions. Respond with "
            "an overall score in the range 0–10."
        ),
    ]

    if option == "general":
        response = llm.invoke(str(messages[0]))
    elif option == "loose":
        response = llm.invoke(str(messages[1]))
    elif option == "strict":
        response = llm.invoke(str(messages[2]))
    else:
        response = "Invalid evaluation profile."
    print(response)
```

```
def extract_and_return_all_zip_files_data_from_path(folder_path):
    all_data = {}

    if not os.path.isdir(folder_path):
        raise FileNotFoundError(
            f"The folder '{folder_path}' does not exist."
        )

    for filename in os.listdir(folder_path):
        if filename.endswith(".zip"):
            zip_path = os.path.join(folder_path, filename)
            zip_data = {}

            with zipfile.ZipFile(zip_path, "r") as zip_ref:
                extract_folder = os.path.join(
                    folder_path, "temp_extracted_files"
                )
                os.makedirs(extract_folder, exist_ok=True)
                zip_ref.extractall(extract_folder)

                for root, _, files in os.walk(extract_folder):
                    for file in files:
                        if file.endswith(".py"):
                            file_path = os.path.join(root, file)
                            with open(file_path, "r", encoding="utf-8") as f:
                                zip_data[file] = f.read()

                # Clean up extracted files
                for root, _, files in os.walk(
                    extract_folder, topdown=False
                ):
                    for name in files:
                        os.remove(os.path.join(root, name))
                    for name in os.listdir(root):
                        os.rmdir(os.path.join(root, name))
                os.rmdir(extract_folder)

            all_data[filename] = zip_data

    return all_data


if __name__ == "__main__":
    try:
        try:
            if len(argv) < 3 or not argv[1] or not argv[2]:
                raise IndexError

            data = extract_and_return_all_zip_files_data_from_path(argv[2])
            for zip_file, files in data.items():
                for file_name, content in files.items():
                    print(
                        f"ZIP File: {zip_file}\n"
                        f"File: {file_name}\nContent:\n\n{content}"
                    )
```

```
                evaluate_code(argv[1], content)
                print("\n" + "-" * 50 + "\n")

    except KeyboardInterrupt:
        exit("\nExiting...")

except IndexError:
    print(
        "\t\tProfile\t\tDirectory\nUsage: "
        + argv[0]
        + "  [strict|loose|general]  [path]"
    )
```