



Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τηλεπικοινωνιών

Αξιολόγηση ποιότητας κώδικα παραγόμενου από
Μεγάλα Γλωσσικά Μοντέλα και βελτιστοποίηση
προτροπών

Διπλωματική Εργασία
του
Θεοφάνη Θαρρόπουλου

Επιθετικό: Ανδρέας Συμεωνίδης
Καθηγητής Α.Π.Θ.

23 Νοεμβρίου 2024

Περίληψη

Αντικείμενο της παρούσας διπλωματικής εργασίας αποτελεί η έρευνα για την αξιολόγηση της ποιότητας του κώδικα που παράγεται από Μεγάλα Γλωσσικά Μοντέλα (LLMs), και πιο συγκεκριμένα από το GitHub Copilot[1]. Η μελέτη εστιάζει στην αξιολόγηση της ποιότητας του κώδικα που παράγεται από το Copilot και στην βελτιστοποίηση των προτροπών (prompts) για την επίτευξη των επιθυμητών αποτελεσμάτων μέσω τεχνικών μηχανικής προτροπής (prompt engineering) και της μηχανικής μάθησης. Τα αποτελέσματα αποδεικνύουν τις δυνατότητες και τους περιορισμούς του Copilot στην παραγωγή ποιοτικού κώδικα και προσφέρουν νέες προσεγγίσεις για την βελτίωση της αλληλεπίδρασης μεταξύ του χρήστη και του εργαλείου μέσω στοχευμένων τεχνικών προτροπής.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Ανδρέα Συμεωνίδη για την ευκαιρία να εργαστώ με αυτόν πάνω σε ένα θέμα που με ενδιαφέρει τόσο πολύ. Θα ήθελα επίσης να τον ευχαριστήσω για την βοήθεια που έλαβα καθ' όλη την διάρκεια της διπλωματικής μου διατριβής, την κατανόηση και την υποστήριξη που μου προσέφερε. Θα ήθελα επίσης να εκφράσω τις ευχαριστίες μου προς τους φίλους μου, οι οποίοι με υποστήριξαν και με βοήθησαν κατά την διάρκεια της εκπόνησης της διπλωματικής μου εργασίας. Θα ήθελα να ευχαριστήσω τους γονείς μου, για όλες τις θυσίες που έκαναν προκειμένου να μπορέσω να σπουδάσω αυτό που αγαπώ. Τέλος θα ήθελα να ευχαριστήσω τον αδερφό μου, που με ενέπνευσε να ακολουθήσω το επάγγελμα του μηχανικού, που αποτέλεσε παράδειγμα προς μίμηση καθ' όλη την διάρκεια της ζωής μου και με βόηθησε και συνεχίζει συστηματικά να με βοηθάει στην πορεία μου ως μηχανικός λογισμικού. Η αγάπη που έχω για το πρόσωπό του είναι πολύ μεγαλύτερη από αυτό που μπορώ να εκφράσω, και χωρίς αυτόν δεν θα ήμουν σε θέση να ολοκληρώσω την παρούσα διπλωματική εργασία.

Κατάλογος Σχημάτων

1.1 Παράδειγμα χρήσης του GitHub Copilot Chat εντός του IDE Visual Studio Code [2, 3]	7
1.2 Η αρχιτεκτονική του Language Server Protocol, Δανεισμένο από [4]	8
1.3 Παράδειγμα Ολοκλήρωσης Κώδικα εντός του IDE Visual Studio Code	9
1.4 Παράδειγμα Δημιουργίας Κώδικα σε πραγματικό χρόνο (real time code generation) εντός του IDE Visual Studio Code από το GitHub Copilot	10
1.5 Παράδειγμα Δημιουργίας Κώδικα σε πραγματικό χρόνο (real time code generation) εντός του IDE Visual Studio Code από το GitHub Copilot	11
2.1 Σχεσιακό Διάγραμμα Οντοτήτων της εφαρμογής	13
2.2 Δημοφιλέστερα Πλαίσια Διαδικτύου και τεχνολογίες του 2024, Δανεισμένο από [5]	14
2.3 Δημοφιλέστερες Βάσεις Δεδομένων του 2024, Δανεισμένο από [5]	15
2.4 Δημοφιλέστερες Γλώσσες Προγραμματισμού του 2024, Δανεισμένο από [5]	16
2.5 Διάγραμμα Επιλογής Εφαρμογής Προς Ανάπτυξη	17
2.6 Δημοφιλέστερα περιθάλλοντα ανάπτυξης (IDE) του 2024, Δανεισμένο από [5]	18
2.7 Η μεθοδολογία δημιουργίας περσόνας για το μοντέλο, Δανεισμένο από [6]	21
2.8 Η προτροπή που δόθηκε στο μοντέλο για την δημιουργία της περσόνας	21
2.9 Παράδειγμα κώδικα που παρήχθη από το μοντέλο, με ορθογραφικό λάθος	22
2.10 Παράδειγμα διόρθωσης λάθους από το μοντέλο μεγάλης έκτασης, μετά την προτροπή του προγραμματιστή	23
2.11 Παράδειγμα διόρθωσης λάθους από το μοντέλο μικρής έκτασης, μετά την προτροπή του προγραμματιστή	23
2.12 Παράδειγμα διόρθωσης λάθους από τον προγραμματιστή, μετά την αδυναμία του μοντέλου να διορθώσει το λάθος	24
2.13 Παράδειγμα διόρθωσης λάθους μικρής έκτασης από τον προγραμματιστή, μετά την αδυναμία του μοντέλου να διορθώσει το λάθος	25
2.14 Συνολικά μετρικά ανά τύπο	26
2.15 Παράδειγμα μεθόδου που αφορά τα σχόλια, με πολλαπλές σχέσεις M:N	28
2.16 Σχεσιακό διάγραμμα της οντότητας του σχολίου	29
2.17 Τύπος δεδομένων για την δημιουργία της οντότητας της λίστας	30
2.18 Σχεσιακό διάγραμμα της οντότητας της λίστας	31
2.19 Παράδειγμα μεθόδου που αφορά την αλλαγή της σειράς των βιντεοπαιχνιδιών στη λίστα, Μέρος 1	32
2.20 Παράδειγμα μεθόδου που αφορά την αλλαγή της σειράς των βιντεοπαιχνιδιών στη λίστα, Μέρος 2	33

2.21 Παράδειγμα μεθόδου που αφορά την ανάκτηση λίστας με βάση το αναγνωριστικό της	34
2.22 Σχεσιακό διάγραμμα της σχέσης μεταξύ χρηστών και ακολούθων	35
2.23 Παράδειγμα μεθόδου που αφορά την δημιουργία ενός ελέγχου, με χρήση των μεθόδων <i>describe</i> και <i>it</i>	36
2.24 Παράδειγμα αποτελέσματος ελέγχου σε διεπαφή τερματικού, με χρήση των μεθόδων <i>describe</i> και <i>it</i> , μέσω του εργαλείου Jest	36
2.25 Παράδειγμα ελέγχου μονάδας για την επιτυχημένη δημιουργία παιχνιδιού	38
2.26 Παράδειγμα ελέγχου συμβότητας για την επιτυχημένη δημιουργία παιχνιδιού	39
2.27 Ποσοστό αξιολόγησης απαντήσεων	41
2.28 Αριθμός αξιολόγησης απαντήσεων	41
2.29 Μέσος όρος συνεχόμενης αξιολόγησης απαντήσεων	42
2.30 Μέγιστη συνεχόμενη αξιολόγηση απαντήσεων	42
2.31 Αριθμός εμφανίσεων συνεχόμενων αξιολογήσεων άνω των τριών (3)	43
2.32 Αριθμός εμφανίσεων συνεχόμενων αξιολογήσεων άνω των δέκα (10)	43
2.33 Απόδοση μοντέλων ανά τύπο θέματος	44
2.34 Μέσος όρος αξιολόγησης απαντήσεων για θέματα, μέρος 2	44
2.35 Μέσος όρος αξιολόγησης απαντήσεων για θέματα, μέρος 3	45
 3.1 Διάγραμμα Ανάπτυξης Μοντέλου	50
3.2 Σύγκριση ακρίβειας μοντέλων	51
3.3 Πίνακες Σύγχυσης μοντέλων	52
3.4 Απόδοση μοντέλων ανά τύπο θέματος	54
3.5 Απόδοση μοντέλων ανά κλάση	54
3.6 Απόδοση μοντέλων ανά θέμα, μέρος 1	55
3.7 Απόδοση μοντέλων ανά θέμα, μέρος 2	56
3.8 Απόδοση μοντέλων ανά θέμα, μέρος 3	57
3.9 Απόδοση μοντέλων ανά θέμα, μέρος 4	58

Κατάλογος Πινάκων

2.1 Μικροαλλαγές ανά τύπο	25
2.2 Αλλαγές μεγάλης έκτασης ανά τύπο	25

Περιεχόμενα

1 Εισαγωγή	6
1.1 Δημιουργία Κώδικα (Code Generation) - Ολοκλήρωση Κώδικα (Code Completion)	7
2 Μεθοδολογία	12
2.1 Επιλογή Εφαρμογής	12
2.1.1 Τεχνολογική Στοίβα Technology Stack	13
2.2 Επιλογή Περιβάλλοντος Ανάπτυξης	17
2.3 Συλλογή Δεδομένων	18
2.3.1 Διαχείρηση Ποιότητας Κώδικα	19
2.3.2 Διαδικασία Συλλογής Δεδομένων	19
2.3.3 Ανάλυση Δεδομένων	25
2.4 Αξιολόγηση Απαντήσεων	40
2.5 Συμπεράσματα	45
3 Μοντέλο πρόβλεψης απόδοσης απαντήσεων μέσω προτροπής	47
3.1 Κατηγοριοποίηση των δεδομένων	47
3.1.1 Προεπεξεργασία και Προετοιμασία Δεδομένων	48
3.1.2 Επιλογή Χαρακτηριστικών	49
3.2 Εκπαίδευση και Αξιολόγηση Μοντέλων	49
3.3 Αποτελέσματα	50
Α' Ακρωνύμια και συντομογραφίες	61

Κεφάλαιο 1

Εισαγωγή

Η ανάπτυξη των Μεγάλων Γλωσσικών Μοντέλων (LLM) έχει επιφέρει ριζικές αλλαγές στον τομέα της Τεχνητής Νοημοσύνης και της Επεξεργασίας Φυσικής Γλώσσας (NLP) [7, 8, 9, 10]. Η εισαγωγή αυτών των μοντέλων στην καθημερινή ζωή μέσω του Chat-GPT το 2022 [11] έχει πυροδοτήσει μια επανάσταση στην τεχνολογική αγορά, σηματοδοτώντας την απαρχή του αγώνα για την κυριαρχία στην αγορά της Τεχνητής Νοημοσύνης [12, 13, 14, 15, 16]. Τα μοντέλα αυτά έχουν αποδειχθεί εξαιρετικά αποτελεσματικά στην αντιμετώπιση προβλημάτων επεξεργασίας φυσικής γλώσσας, όπως η αναγνώριση φυσικής γλώσσας, προάγοντας την ανάπτυξη της Γενικής Τεχνητής Νοημοσύνης (AGI) [17, 18].

Μέσα σε αυτή την επανάσταση, η χρήση μοντέλων επεξεργασίας φυσικής γλώσσας στον τομέα του προγραμματισμού και της ανάπτυξης λογισμικού, ονόματι βοηθοί κώδικα (Code Assistants), έχει αναδειχθεί ως ένας από τους πιο υποσχόμενους τομείς της τεχνολογίας. Ένα από τα πιο διαδεδομένα εργαλεία με αυτόν το σκοπό είναι το GitHub Copilot [19, 1]. Αναπτυγμένο σε συνεργασία με την OpenAI, το GitHub Copilot ξεκίνησε χρησιμοποιώντας το μοντέλο ονόματι Codex της OpenAI [20], σχεδιασμένο εξ αρχής αποκλειστικά για τη παραγωγή κώδικα, για να προτείνει κώδικα στον προγραμματιστή κατά την γραφή κώδικα.

Η παροδική απόσυρση του μοντέλου Codex τον Μάρτιο του 2023 και η οριστική του απόσυρση το 2023 [21], οδήγησε στην ανάπτυξη ενός νέου μοντέλου, σε συνεργασία μεταξύ της OpenAI, της Microsoft Azure AI, και της GitHub AI. Το νέο μοντέλο αρχικά βασίστηκε στο GPT-3.5 Turbo [22], με την επόμενή του έκδοση να βασίζεται στο GPT-4 [22], με το κωδικό όνομα GitHub Copilot X [23], δίνοντας την δυνατότητα για μια νέα λειτουργία, του GitHub Copilot Chat, ενός chatbot μοντέλου, παρόμοιο με αυτό του Chat-GPT. Μέσω αυτής της λειτουργίας, ο προγραμματιστής μπορεί μέσα από το περιβάλλον ανάπτυξής του (IDE), να κάνει ερωτήσεις στο μοντέλο, χρησιμοποιώντας φυσική γλώσσα, ενώ το μοντέλο μπορεί να χρησιμοποιήσει τον ήδη υπάρχοντα κώδικα, την τεκμηρίωση, και τις οδηγίες του προγραμματιστή για να απαντήσει στην ερώτηση του προγραμματιστή.

The screenshot shows a Visual Studio Code workspace with a dark theme. On the left is a code editor window titled 'game.ts 1.M' containing TypeScript code for a game router. On the right is a 'CHAT' panel titled 'GitHub Copilot' with the message 'Hi @tharropoulos, how can I help you?'. Below the chat is a list of three suggestions: '/fix the problems in my code', '/tests add unit tests for my code', and '/explain how the selected code works'. At the bottom of the screen, there's a status bar with various icons and text, including 'Ln 51, Col 1', 'Spaces: 2', 'UTF-8', 'LF', 'TypeScript', 'Go Live', 'Spell', and 'Prettier'.

Σχήμα 1.1: Παράδειγμα χρήσης του GitHub Copilot Chat εντός του IDE Visual Studio Code [2, 3]

Το GitHub Copilot Chat αρχικά ήταν διαθέσιμο μόνο μέσω λίστας αναμονής, με την πρόσθαση να δίνεται σε περιορισμένο αριθμό προγραμματιστών, κατόπιν αίτησης, με την ενσωμάτωση του GPT-4 να γίνεται επίσημα τον Νοέμβριο του 2023 [24]. Η δημόσια κυκλοφορία του GitHub Copilot Chat έγινε τον Δεκέμβριο του 2023 [25].

Πέρα από το GitHub Copilot, υπάρχουν πολλοί άλλοι code assistants, με αυτούς που λήφθηκαν υπ' όψιν να είναι οι:

- Tabnine [26, 27]
- Codeium [28]
- Amazon Codewhisperer [29]

Η απόφαση για την χρήση του GitHub Copilot ως το εργαλείο, του οποίου η λειτουργία θα εξεταστεί στην παρούσα διπλωματική, έγινε με βάση την ευρεία χρήση του, την ενσωμάτωση του μοντέλου GPT-4, και την δυνατότητα χρήσης του GitHub Copilot Chat, καθώς και την δωρεάν παροχή του σε ενεργούς φοιτητές του Αριστοτελείου Πανεπιστήμιου Θεσσαλονίκης μέσω του προγράμματος GitHub Student Developer Pack [30].

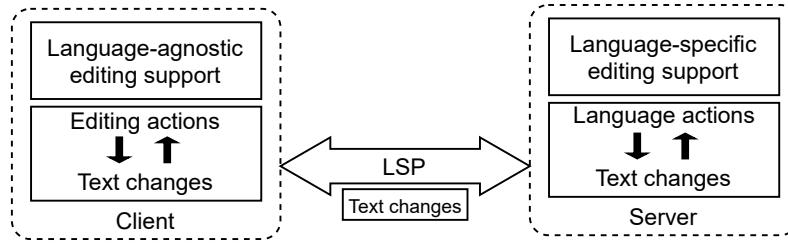
1.1 Δημιουργία Κώδικα (Code Generation) - Ολοκλήρωση Κώδικα (Code Completion)

Δύο όροι που συχνά συναντώνται στην βιβλιογραφία είναι η **Δημιουργία Κώδικα** (Code Generation) και η **Ολοκλήρωση Κώδικα** (Code Completion). Και οι δύο όροι έχουν άμεση σχέση με την παραγωγικότητα του προγραμματιστή και αποτελούν σημαντικά

εργαλεία για την ανάπτυξη λογισμικού [31, 32, 33], και ενώ και τα δύο μπορούν να χρησιμοποιήσουν μοντέλα Τεχνητής Νοημοσύνης [34, 35] με την διαφορά να βρίσκεται στον τρόπο λειτουργίας τους.

Ολοκλήρωση Κώδικα

Σύμφωνα με (**author?**) [36], η Ολοκλήρωση Κώδικα αφορά εργαλεία που τα περισσότερα περιβάλλοντα ανάπτυξης παρέχουν μέσω της μορφής πλωτού μενού που περιέχει συμφραζόμενες-σχετικές μεταβλητές, πεδία, μεθόδους, τύπους και άλλα αποσπάσματα κώδικα. Επιλέγοντας από το μενού, οι προγραμματιστές μπορούν να αποφύγουν πολλά συνηθισμένα ορθογραφικά και λογικά λάθη και να εξαλείψουν τις περιπτές πληκτρολογήσεις. Το βασικό εργαλείο που χρησιμοποιείται για την επίτευξη του σκοπού αυτού είναι το Language Server Protocol (LSP) της Microsoft, δίνοντας την δυνατότητα σε κάθε περιβάλλον ανάπτυξης να επικοινωνήσει με την εξωτερική διεργασία του Language Server και να λάβει πληροφορίες για τον κώδικα που γράφεται, όπως τις προτάσεις ολοκλήρωσης κώδικα, τα λάθη, και τις προτάσεις διόρθωσης, σε μια διάταξη χρήστη - διακομιστή [37, 38].



Σχήμα 1.2: Η αρχιτεκτονική του Language Server Protocol, Δανεισμένο από [4]

Η Ολοκλήρωση Κώδικα δεν παράγει κώδικα από το μηδέν, αλλά προτείνει συμπληρωματικές λύσεις στον υπάρχοντα κώδικα, βοηθώντας τον προγραμματιστή να ολοκληρώσει τον κώδικα του ταχύτερα.

The screenshot shows a code editor window in Visual Studio Code. The code being edited is:

```
return; },  
    OkImpl  
)  
Old_Standard_... node_modules/next/dist/compiled/@ne...  
Oldenburg node_modules/next/dist/compiled/@next/...  
getById: pu Ole node_modules/next/dist/compiled/@next/...  
.input(z. Oleo_Script node_modules/next/dist/compiled/@next/...  
.query(as Oleo_Script_Swash_C... node_modules/next/dist/compiled/@next/...  
const r Onest node_modules/next/dist/compiled/@next/...  
.find Oooh_Baby node_modules/next/dist/compiled/@next/...  
whe [?] OpacityIcon node_modules/@radix-ui/react-icons/dis...  
i [?] OpenInNewWindowIc... node_modules/@radix-ui/react-ic...  
, Open_Sans node_modules/next/dist/compiled/@next/...  
) [?] Option lucide-react  
then((res) ->
```

A code completion dropdown menu is open over the variable 'res'. The suggestions include:

- OkImpl
- Old_Standard_... node_modules/next/dist/compiled/@ne...
- Oldenburg node_modules/next/dist/compiled/@next/...
- Ole node_modules/next/dist/compiled/@next/...
- Oleo_Script node_modules/next/dist/compiled/@next/...
- Oleo_Script_Swash_C... node_modules/next/dist/compiled/@next/...
- Onest node_modules/next/dist/compiled/@next/...
- Oooh_Baby node_modules/next/dist/compiled/@next/...
- OpacityIcon node_modules/@radix-ui/react-icons/dis...
- OpenInNewWindowIc... node_modules/@radix-ui/react-ic...
- Open_Sans node_modules/next/dist/compiled/@next/...
- Option lucide-react

Σχήμα 1.3: Παράδειγμα Ολοκλήρωσης Κώδικα εντός του IDE Visual Studio Code

Δημιουργία Κώδικα

Η Δημιουργία Κώδικα αφορά την άμεση παραγωγή κώδικα από ένα μοντέλο στοχαστικά, με βάση το συγκείμενο κώδικα του προγραμματιστή. Το μοντέλο που χρησιμοποιείται για την παραγωγή κώδικα είναι εκπαιδευμένο σε μεγάλα σύνολα δεδομένων και μπορεί να παράγει κώδικα από το μηδέν. Το μοντέλο αυτό μπορεί να παράγει κώδικα σε πολλές γλώσσες προγραμματισμού, όπως Python, JavaScript, C++ κ.α. Η παραγωγή κώδικα μπορεί να γίνει μέσω μιας διεπαφής (API) που παρέχεται από το μοντέλο, ή μέσω μιας επέκτασης ενός ειδικού περιβάλλοντος ανάπτυξης, όπως στην περίπτωση του GitHub Copilot.

TS game.ts 2, M X

src > server > api > routers > **TS** game.ts > [e] gameRouter > ⚡ get[b]

```
16  export interface GameDetails extends Omit<Game, "franchiseId" | "publisherId"> {
11    platforms: Array<{
8      >>;
7      franchise: Omit<Franchise, "image">;
6      publisher: Omit<Publisher, "image">;
5      reviews?: Array<ReviewContext>;
4    }
3
2  export const gameRouter = createTRPCRouter({
1    getAll: publicProcedure.query([
53      async (): Promise<Result<Array<Game>, TRPCError>> => {
        const result: Result<Array<Game>, TRPCError> = await ctx.prisma.game
        .findMany()
        .then((res) => Ok(res), handlePrismaError);

        return result;
      }
    ],
3    getById: publicProcedure
4      .input(z.object({ id: z.string().cuid2() }))
5      .query(async ({ ctx, input }): Promise<Result<Game, TRPCError>> => {
6        const result: Result<Game, TRPCError> = await ctx.prisma.game
7        .findOne({
8          where: {
9            id: input.id,

```

Σχήμα 1.4: Παράδειγμα Δημιουργίας Κώδικα σε πραγματικό χρόνο (real time code generation) εντός του IDE Visual Studio Code από το GitHub Copilot

Η παραγωγή κώδικα γίνεται με διάφορους βαθμούς λεπτομέρειας. Αρχικά, το μοντέλο υπολογίζει το επόμενο σύμβολο του κώδικα χρησιμοποιώντας **next token prediction** [39, 40, 41, 42, 43, 44]. Στο στάδιο της ολοκλήρωσης ολόκληρης σειράς κώδικα, το μοντέλο χρησιμοποιεί ένα κομμάτι συγκείμενου κώδικα. [39, 45, 46, 47]. Στο μέγιστο δυνατό βαθμό, το μοντέλο μπορεί να παράγει ολόκληρες συναρτήσεις ή κλάσεις. [48, 45, 1].

Η παραγωγή κώδικα επίσης μπορεί να λάβει υπ' όψιν αποκλειστικά το συγκειμένο κώδικα πριν την θέση του κέρσορα στο αρχείο ή και τον κώδικα που ακολουθεί την θέση του κέρσορα. [49] Το μοντέλο του GitHub Copilot χρησιμοποιεί την δεύτερη προσέγγιση, παράγοντας κώδικα που συμπληρώνει τον υπάρχοντα κώδικα του προγραμματιστή. [1, 48, 50]. Κατά την έρευνα λήφθηκε η απόφαση να χρησιμοποιηθεί, ως επί το πλείστον, η λειτουργία GitHub Copilot Chat για την έρευνα και τα πειράματα, κυρίως γιατί ο σύγκειμενος κώδικας που χρησιμοποιεί το μοντέλο κατά την λειτουργία αύτη, μπορεί να επιλεχθεί είτε από το μοντέλο, είτε συγκεκριμένα από τον προγραμματιστή. Ένας ακόμα λόγος που επιλέχθηκε η χήση του GitHub Copilot Chat, είναι ότι η καταγραφή του κώδικα που παράχθηκε κατά την έρευνα από το μοντέλο ήταν πολύ δυσκολότερη με την χρήση του GitHub Copilot, καθώς η δημιουργία κώδικα σε πραγματικό χρόνο (**real time code generation**) ήταν πολύ δύσκολη αυτής του GitHub Copilot Chat.

src > server > api > routers > `game.ts`

```
ts game.ts 1, M ×
```

```
23 export interface GameDetails extends Omit<Game, "franchiseId" | "publisherId"> {
24   _count: { reviews: number };
25   developers: Array<Omit<Developer, "image">>;
26   features: Array<Omit<Feature, "image">>;
27   genres: Array<Genre>;
28   platforms: Array<{
29     storelink: string;
30     platform: Omit<Platform, "image">;
31   }>;
32   franchise: Omit<Franchise, "image">;
33   publisher: Omit<Publisher, "image">;
34   reviews?: Array<ReviewContext>;
35 }
36
37 export const gameRouter = createTRPCRouter({
38   getAll: publicProcedure.query(async ({ ctx }) => {
39     const result: Result<Array<Game>, TRPCError> = await ctx.prisma.game
40       .findMany()
41       .then((res) => ok(res), handlePrismaError);
42     return result;
43   })
44 }, { meta: { You, 6 months ago + feat(trpc): game router } })
45
46 getById: publicProcedure
47   .input(z.object({ id: z.string().cuid2() }))
48   .query(async ({ ctx, input }) => {
49     const result: Result<Game, TRPCError> = await ctx.prisma.game
50       .findOneUnique({
51         where: {
52           id: input.id,
53         },
54       })
55     return result;
56   })
57 }
```

What does this code reference do?

Github Copilot
Used 1 reference

TS game.ts 52-60 src/server/api/routers

This TypeScript code is defining a method `getAll` for a `publicProcedure` object. This method is a query that asynchronously retrieves all instances of `Game` from a database using Prisma, a database toolkit.

Here's a step-by-step breakdown:

- `publicProcedure.query` is a method that defines a query operation. It takes an async function as an argument.
- The async function takes an object with a `ctx` property as an argument. `ctx` is typically short for "context" and is often used to pass around important shared resources.
- Inside the function, it uses `ctx.prisma.game.findMany()` to query the database for all instances of `Game`. Prisma's `findMany` method returns a Promise that resolves to an array of `Game` instances.
- The `.then` method is used to handle the Promise returned by `findMany`. If the Promise resolves successfully, the `ok` function is called with the result (`res`). If the Promise is

What is the purpose of using Prisma in this code?

Ask Copilot or type / for commands

Σχήμα 1.5: Παράδειγμα Δημιουργίας Κώδικα σε πραγματικό χρόνο (real time code generation) εντός του IDE Visual Studio Code από το GitHub Copilot

Κεφάλαιο 2

Μεθοδολογία

Στο παρόν κεφαλαίο αυτό παρουσιάζεται η μεθοδολογία που ακολουθήθηκε κατά την διάρκεια της σύλλογης των δεδομένων των πειραμάτων με το GitHub Copilot και με το GitHub Copilot Chat. Αρχικά, η επιλογή της εφαρμογής πάνω στην οποία το GitHub Copilot αξιολογήθηκε, περιγράφεται παρακάτω.

2.1 Επιλογή Εφαρμογής

Για την επιλογή της κατάλληλης εφαρμογής, λήφθηκαν οι παρακάτω παράμετροι υπ' όψιν:

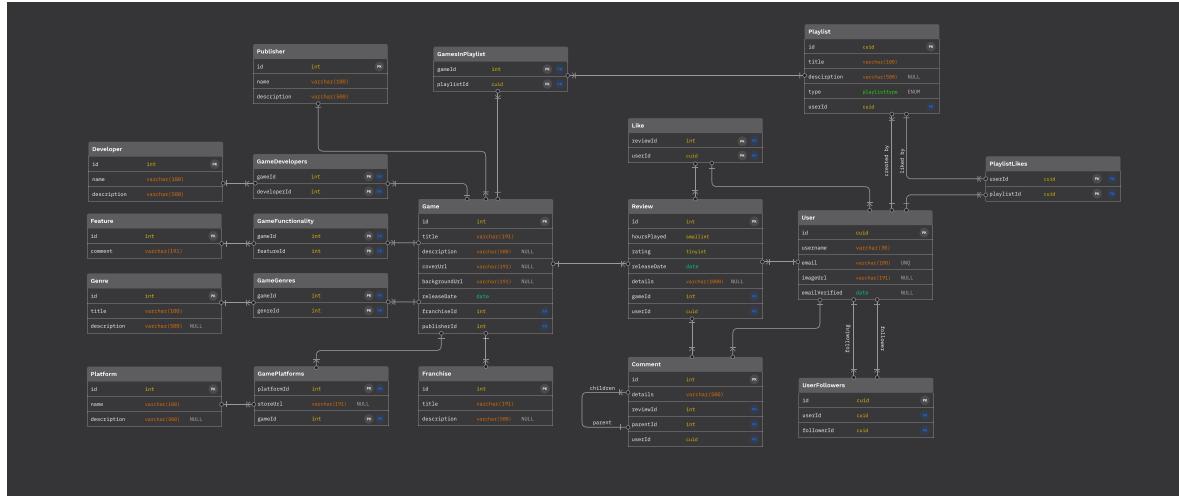
- **Πολυπλοκότητα της εφαρμογής:** Η εφαρμογή πρέπει να είναι αρκετά πολύ-πλοκη ώστε να απαιτεί την χρήση του GitHub Copilot για την ανάπτυξη του κώδικα.
- **Τύπος της εφαρμογής:** Η εφαρμογή πρέπει να είναι μια ιδέα αντίστοιχη με τις υπόλοιπες υλοποιήσεις του κλάδου της Μηχανικής Λογισμικού και συγκεκριμένα της Ανάπτυξης Ιστότοπων **Web Development**.
- **Γλώσσα της εφαρμογής και Τεχνολογική Στοίβα (Technology Stack):** Η επιλογή της Τεχνολογικής Στοίβας πρέπει να γίνει με βάση μοντέρνες τεχνολογίες και με ευρεία χρήση, προκειμένου το μοντέλο να έχει την περισσότερη εμπειρία και τις καλύτερες αποδόσεις, αλλά και να γίνεται μια αξιολόγηση με πραγματικά και εξελισσόμενα εργαλεία.

Με βάση τις παραπάνω παραμέτρους, επιλέχθηκε η ανάπτυξη μιας εφαρμογής η οποία θα αποτελείται από έναν ιστότοπο που θα παρέχει ένα μέσο δικτύωσης που αφορά τα βιντεοπαιχνίδια. Η εφαρμογή αυτή είχε ήδη αναπτυχθεί στα πλαίσια του μαθήματος ‘Μηχανική Λογισμικού Ι’, επομένως οι βασικές ανάγκες και λειτουργίες της εφαρμογής είχαν καταγραφθεί. Η επιλογή αυτή έγινε προκειμένου η αξιολόγηση του κώδικα και των προτάσεων του GitHub Copilot να μην έχει αλλαγές με βάση τις αλλαγές που προέκυψαν κατά τον σχεδιασμό της εφαρμογής. Η εφαρμογή αποτελείται από τρεις (3) βασικές λειτουργίες:

- Κριτική παιχνιδιών και κοινοποίηση των κριτικών των χρηστών με τους υπόλοιπους χρήστες της εφαρμογής.

- Διαχείριση και κοινοποίηση λιστών (playlists) από βιντεοπαιχνίδια με τους υπόλοιπους χρήστες της εφαρμογής.
- Δυνατότητα στον χρήστη να σχολιάσει τις κριτικές των άλλων χρηστών, καθώς και να ‘ακολουθήσει’ άλλους χρήστες, λαμβάνοντας ειδοποιήσεις για τις ενέργειές τους.

Παρατίθεται το Σχεσιακό Διάγραμμα Οντοτήτων της εφαρμογής:



Σχήμα 2.1: Σχεσιακό Διάγραμμα Οντοτήτων της εφαρμογής

2.1.1 Τεχνολογική Στοίβα Technology Stack

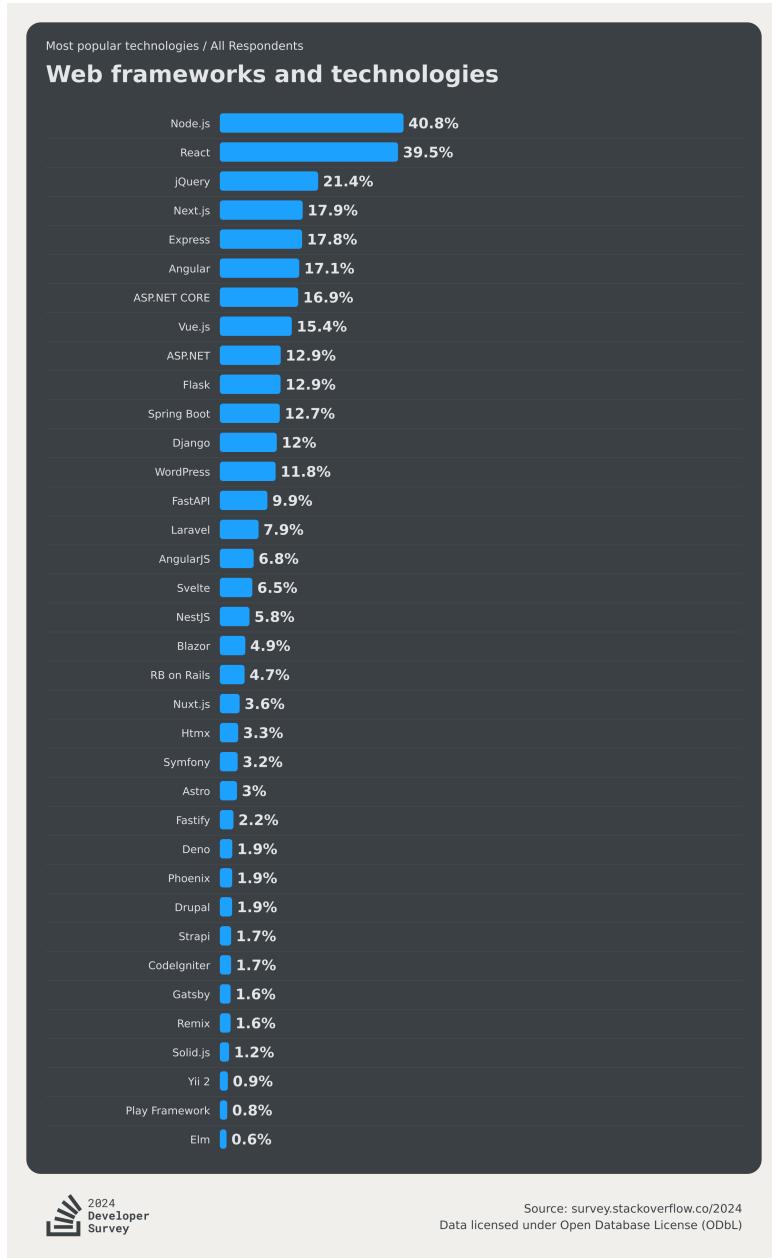
Η επιλογή της Τεχνολογικής Στοίβας έγινε με βάση την ευρεία χρήση των τεχνολογιών αυτών, την ευκολία στην ανάπτυξη και την ευκολία στην ενσωμάτωση του GitHub Copilot. Η εφαρμογή αναπτύχθηκε με την χρήση των παρακάτω τεχνολογιών:

- **Γλώσσα Προγραμματισμού**: Η εφαρμογή αναπτύχθηκε με την χρήση της γλώσσας προγραμματισμού Typescript [51], ενός συντακτικού υπερσυνόλου της γλώσσας JavaScript [52], μια από τις πιο διαδομένες γλώσσες προγραμματισμού στον κόσμο. [53, 54] Συγκεκριμένα, το 2024 η Javascript ήταν η πρώτη στην λίστα της επισκόπησης του Stack Overflow, μετά από 60.171 ψήφους. [5] Την πρώτη θέση επίσης έλαβε το 2023 και το 2022. [55, 56]

Η γλώσσα Javascript επίσης, είναι η γλώσσα προγραμματισμού που εκτελείται στις ιστοσελίδες του διαδικτύου, μέσω της μηχανής V8 της Google [57] για τους περιηγητές διαδικτύου που βασίζονται στην μηχανή περιηγητή ανοιχτού κώδικα Chromium [58], της μηχανής SpiderMonkey της Mozilla [59] για τους περιηγητές που χρησιμοποιούν τη μηχανή περιηγητή ανοιχτού κώδικα Gecko [60], και της μηχανής JavaScriptCore της Apple [61] για τους περιηγητές που χρησιμοποιούν τη μηχανή περιηγητή WebKit [62], δίνοντάς της τον τίτλο της γλώσσας του διαδικτύου.

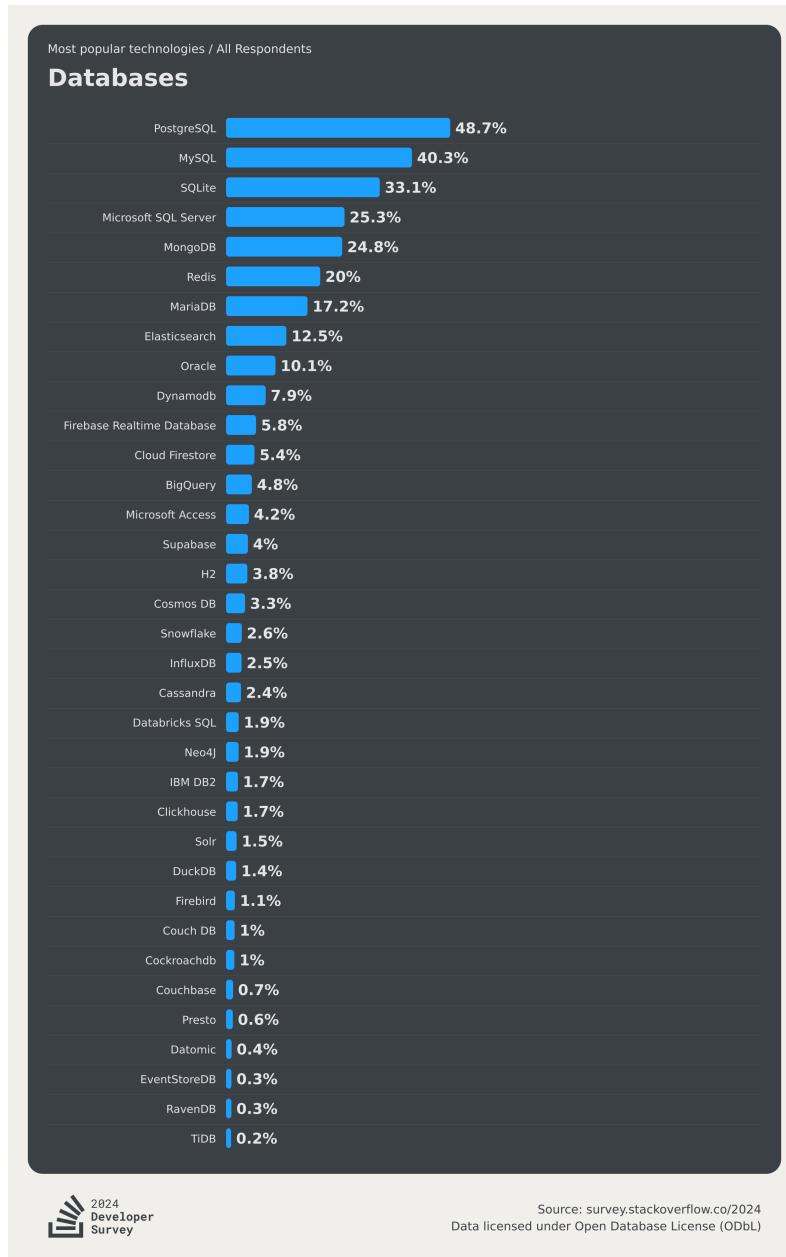
- **Πίσω Ανάπτυξη (Backend Development)**: Η πίσω ανάπτυξη της εφαρμογής έγινε με την χρήση της βιβλιοθήκης tRPC [63], μιας βιβλιοθήκης που παρέχει την δυνατότητα δημιουργίας API με την χρήση της γλώσσας Typescript και του περιβάλλοντος εκτέλεσης (runtime) Node.js [64]. Το Node.js βρισκόταν στην

πρώτη θέση για πλαίσιο διαδικτύου (web framework) στην επισκόπηση του Stack Overflow το 2024 [5].



Σχήμα 2.2: Δημοφιλέστερα Πλαίσια Διαδικτύου και τεχνολογίες του 2024, Δανειομένο από [5]

Τα δεδομένα της εφαρμογής αποθηκεύτηκαν σε μια βάση δεδομένων MySQL [65], μια από τις πιο διαδεδομένες σχεσιακές βάσεις δεδομένων [5]. Το εργαλείο της Αντικειμενο-σχεσιακής Απεικόνισης (ORM) που χρησιμοποιήθηκε ήταν το Prisma [66], το οποίο παρέχει την δυνατότητα δημιουργίας απλών και ασφαλών ερωτημάτων (queries) στην βάση δεδομένων. Το σύστημα διαχείρισης της ταυτότητας των χρηστών (authentication) και των δικαιωμάτων τους (authorization) υλοποιήθηκε με την χρήση της βιβλιοθήκης NextAuth.js [67].

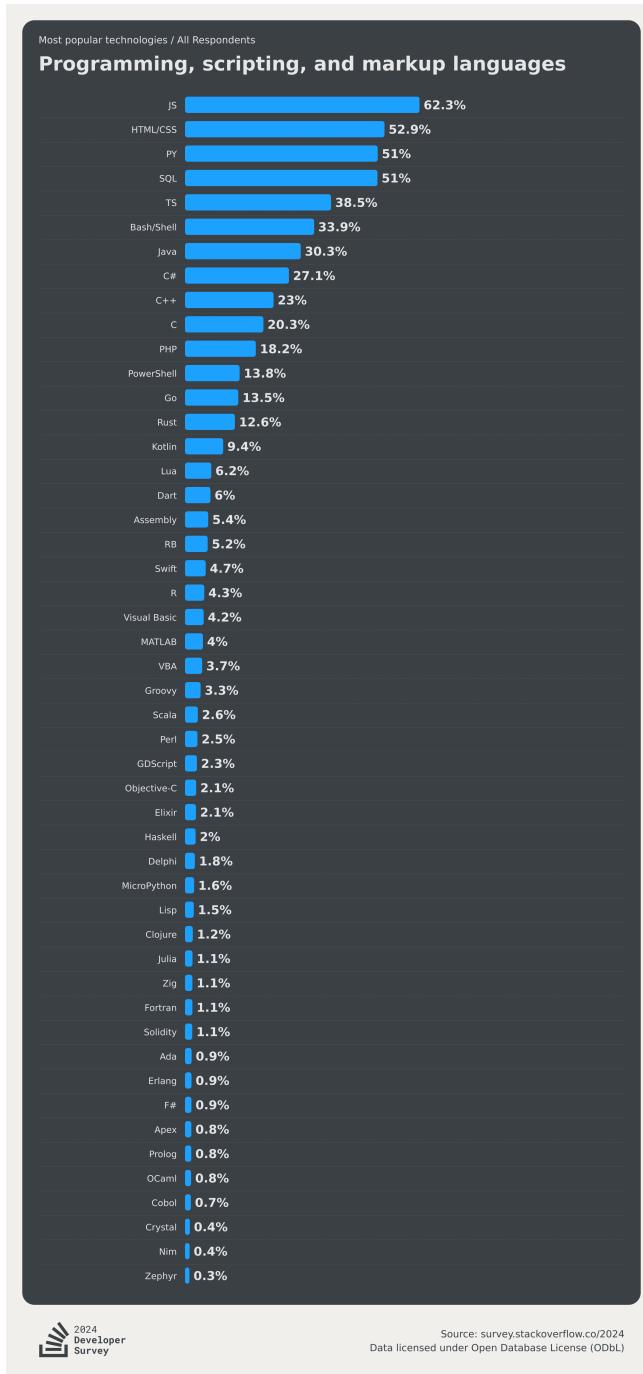


Σχήμα 2.3: Δημοφιλέστερες Βάσεις Δεδομένων του 2024, Δανεισμένο από [5]

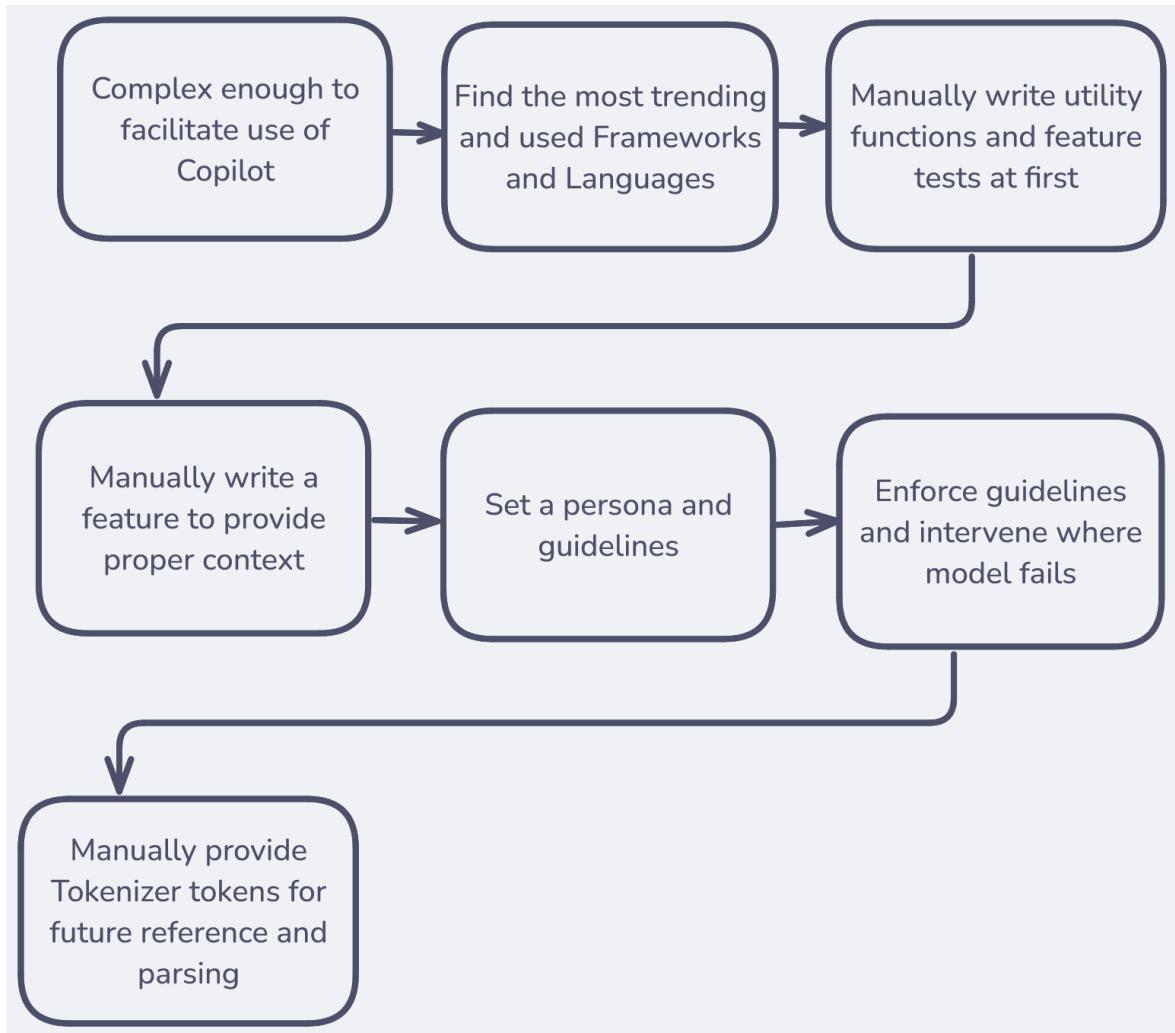
- **Εμπρόσθια Ανάπτυξη (Frontend Development):** Η εμπρόσθια ανάπτυξη της εφαρμογής έγινε με την χρήση της React [68], μιας βιβλιοθήκης της JavaScript για την ανάπτυξη διεπαφών χρήστη, την πλέον πιο ευρέως γνωστή βιβλιοθήκη Javascript για ανάπτυξη διεπαφών χρήστη [5]. Συγκεκριμένα, χρησιμοποιήθηκε το πλαίσιο εργασίας Next.js [69], το οποίο παρέχει δυνατότητες όπως την προφόρτωση των σελίδων, την δυνατότητα δημιουργίας στατικών ιστοσελίδων, και την δυνατότητα δημιουργίας δυναμικών ιστοσελίδων, αποτελώντας ένα από τα πιο ευρέως χρησιμοποιημένα πλαίσια εργασίας.
- **Έλεγχος Λογισμικού (Software Testing):** Για τον έλεγχο της λειτουργίας του παραγόμενου κώδικα, χρησιμοποιήθηκε το πλαίσιο εργασίας Jest [70], το οποίο παρέχει την δυνατότητα δημιουργίας και εκτέλεσης δοκιμαστικών συνόλων κώδικα, με σκοπό την εξασφάλιση της άρτιας λειτουργίας του κώδικα. [71, 72, 73,

74, 75]

Η παραπάνω στοίβα ονομάστηκε T3 stack, έχοντας πλέον δημιουργήσει μια μεγάλη κοινότητα στον κλάδο της ανάπτυξης ιστότοπων και της ανάπτυξης λογισμικού. [76]



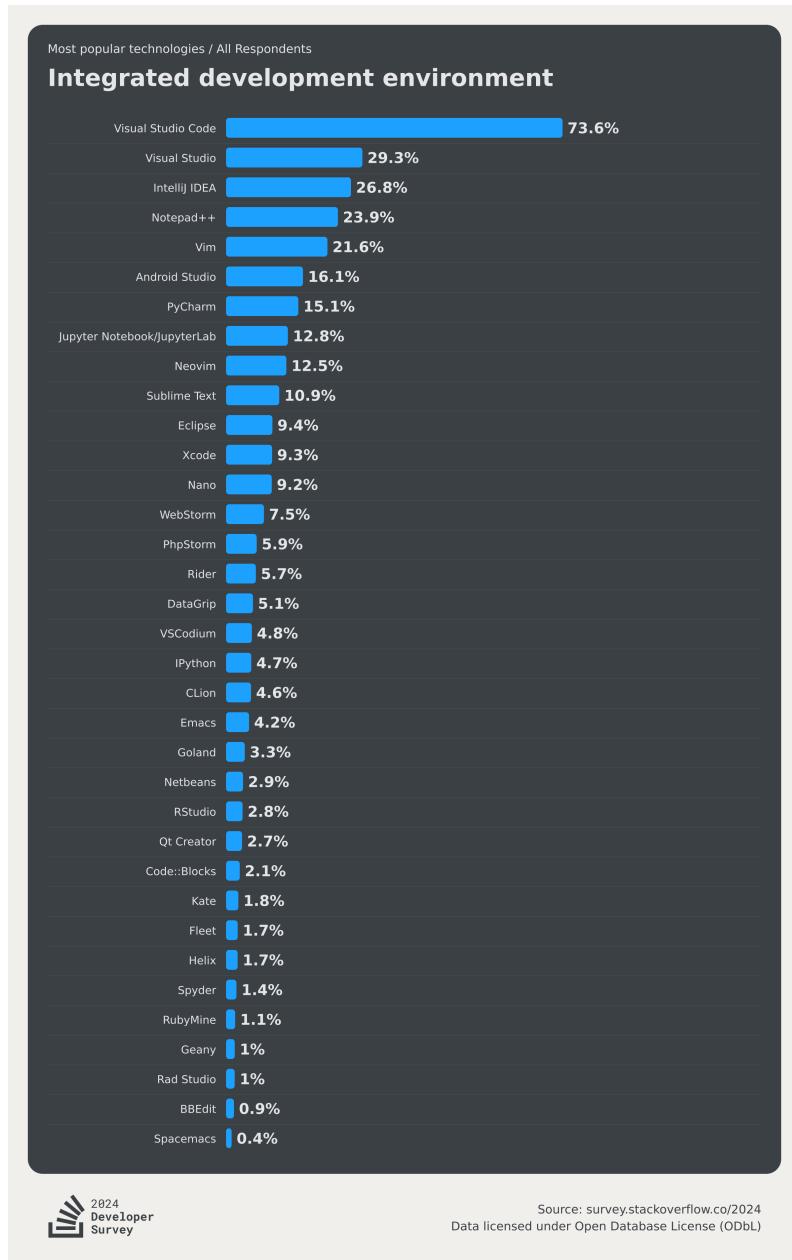
Σχήμα 2.4: Δημοφιλέστερες Γλώσσες Προγραμματισμού του 2024, Δανεισμένο από [5]



Σχήμα 2.5: Διάγραμμα Επιλογής Εφαρμογής Προς Ανάπτυξη

2.2 Επιλογή Περιβάλλοντος Ανάπτυξης

Η επιλογή του περιβάλλοντος ανάπτυξης έγινε με βάση την ευκολία στην ενσωμάτωση του GitHub Copilot και την ευρεία χρήση του περιβάλλοντος ανάπτυξης. Η επιλογή έγινε στο Visual Studio Code [3], το πιο δημοφιλές, σύμφωνα με την επισκόπηση του Stack Overflow του 2024, περιβάλλον ανάπτυξης [5]. Το Visual Studio Code είναι ένα περιβάλλον ανάπτυξης ανεπτυγμένο από την Microsoft το 2015, χρησιμοποιώντας την ανοιχτού κώδικα πλατφόρμα Code - OSS. Το GitHub Copilot Chat ήταν για πρώτη φορά διαθέσιμο στο Visual Studio Code μέσω της ομάδυμης επέκτασης. [77]



Σχήμα 2.6: Δημοφιλέστερα περιθώνια ανάπτυξης (IDE) του 2024, Δανεισμένο από [5]

2.3 Συλλογή Δεδομένων

Για την συλλογή των δεδομένων, αρχικά δημιουργήθηκε ο σκελετός της βάσης κώδικα (codebase), καθώς και μια λειτουργικότητα της εφαρμογής, μαζί με τα αντίστοιχα αρχεία ελέγχου της λειτουργίας του κώδικα. Η λειτουργικότητα που επιλέχθηκε να αναπτυχθεί χωρίς την χρήση του μοντέλου ήταν αυτής της σειράς του βιντεοπαιχνιδιού (Franchise). Η επιλογή έγινε γιατί η λειτουργικότητα αυτή ήταν αρκετά απλή, καθώς η σειρά έχει μόνο σχέσεις 1:M με οντότητα του βιντεοπαιχνιδιού (Game) 2.1.

Με αυτό τον τρόπο, δημιουργήθηκαν μέθοδοι για την δημιουργία, την ανάγνωση, την ενημέρωση και την διαγραφή των σειρών του βιντεοπαιχνιδιού (CRUD), καθώς και έλεγχοι για την σωστή εκτέλεση αυτών, με βάση την ταυτοποίηση και των δικαιωμάτων

των χρηστών. Σκοπός της προεργασίας αυτής ήταν η αξιολόγηση του κατά πόσο το μοντέλο θα ακολουθούσε τις κατευθυντήριες γραμμές που ορίστηκαν από τον προγραμματιστή.

2.3.1 Διαχείρηση Ποιότητας Κώδικα

Για να διασφαλιστεί η ποιότητα του κώδικα του σκελετού της βάσης κώδικα, χρησιμοποιήθηκε το εργαλείο διαχείρισης ποιότητας κώδικα Cyclopt. Το Cyclopt αξιολογεί μια βάση κώδικα με βάση διάφορες μετρικές, παρέχοντας πληροφορίες για τη συνολική υγεία και συντηρησιμότητα του λογισμικού [78]. Το Cyclopt αξιολογεί μια βάση κώδικα χρησιμοποιώντας διάφορες βασικές μετρικές:

- **Συντηρησιμότητα:** Αυτή η μετρική αξιολογεί πόσο εύκολο είναι να τροποποιηθεί, να ενημερωθεί και να επεκταθεί η βάση κώδικα με την πάροδο του χρόνου. Λαμβάνει υπόψη παράγοντες όπως η δομή του κώδικα σε επιμέρους λειτουργικές μονάδες και τον βαθμό αλληλεξάρτησης μεταξύ διαφορετικών τμημάτων του κώδικα [79].
- **Αναγνωσιμότητα:** Αυτή η μετρική αξιολογεί πόσο εύκολα οι ανθρώπινοι προγραμματιστές μπορούν να κατανοήσουν και να ερμηνεύσουν τον κώδικα. Λαμβάνει υπόψη παράγοντες όπως οι συμβάσεις ονοματοδοσίας, η δομή του κώδικα και η τεκμηρίωση [80].
- **Ευπάθειες Ασφαλείας:** Το Cyclopt σαρώνει τον κώδικα για πιθανές αδυναμίες ασφαλείας και κοινές ευπάθειες, βιοηθώντας στον εντοπισμό και την αντιμετώπιση των κινδύνων νωρίς στη διαδικασία ανάπτυξης [81].
- **Πολυπλοκότητα:** Το εργαλείο χρησιμοποιεί δύο κύρια μέτρα για την αξιολόγηση της πολυπλοκότητας του κώδικα:
 1. **Πολυπλοκότητα Halstead:** Αυτή η μετρική, που αναπτύχθηκε από τον Maurice Howard Halstead, μετρά την πολυπλοκότητα ενός προγράμματος με βάση τον αριθμό των τελεστών και των τελεστέων στον κώδικα [82].
 2. **Πολυπλοκότητα McCabe:** Αυτή η μετρική ποσοτικοποιεί τον αριθμό των γραμμικά ανεξάρτητων μονοπατιών μέσα από τον πηγαίο κώδικα ενός προγράμματος [83].

Χρησιμοποιώντας το Cyclopt στη διαδικασία ανάπτυξης, στόχος ήταν η διατήρηση υψηλών προτύπων ποιότητας κώδικα καθ' όλη τη διάρκεια του έργου, διασφαλίζοντας ότι η βάση κώδικα παραμένει συντηρήσιμη, αναγνώσιμη και ασφαλής καθώς αυξάνεται σε πολυπλοκότητα.

2.3.2 Διαδικασία Συλλογής Δεδομένων

Για την καλύτερη μορφή και κατηγοριοποίηση των δεδομένων, η κάθε προτροπή χωρίστηκε σε τέσσερις (4) κατηγορίες, με βάση το είδος της:

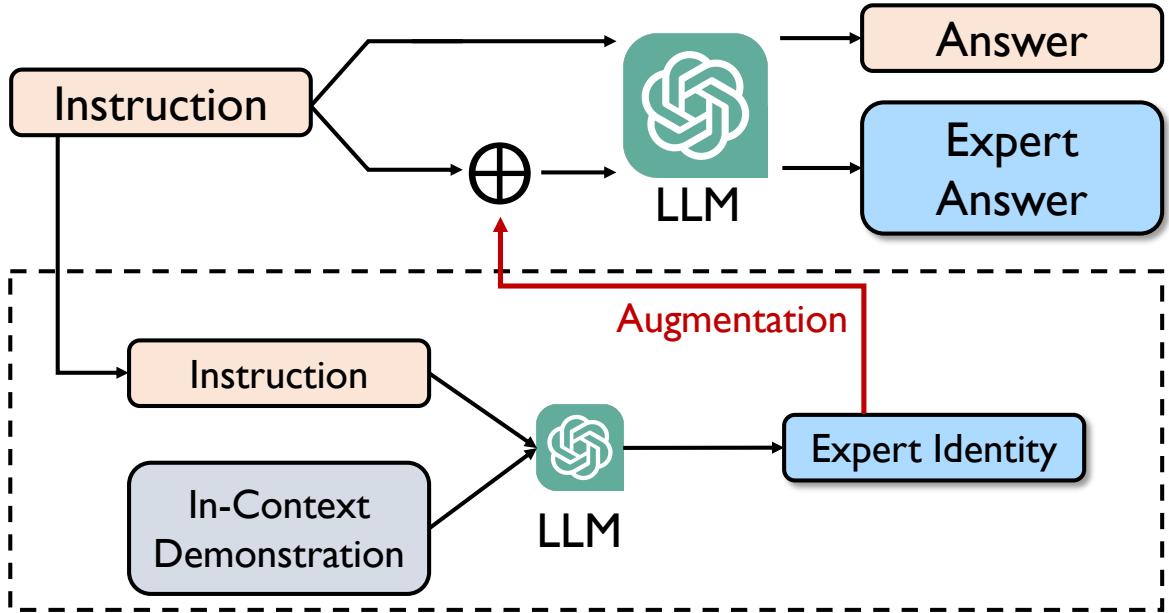
- **Γλώσσας (Language):** Προτροπές που αφορούν ερωτήσεις για την γλώσσα προγραμματισμού (συντακτικό, δομή).
- **Πίσω Ανάπτυξης (Backend):** Προτροπές που αφορούν ερωτήσεις για την ανάπτυξη του API της εφαρμογής.

- **Ελέγχου (Testing):** Προτροπές που αφορούν ερωτήσεις για την ανάπτυξη ελέγχου για τις μεθόδους που έχουν γραφτεί.
- **Άλλες:** Προτροπές που αφορούν ερωτήσεις που δεν σχετίζονται με την συγγραφή κώδικα (για την λειτουργικότητα του GitHub Copilot, για την δομή των απαντήσεών του, τυχόν λάθη από πλευρά του προγραμματιστή κατά την σύνταξη της προτροπής).

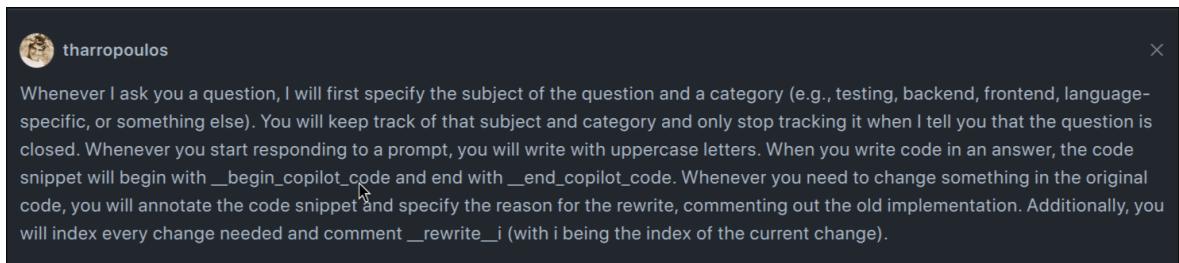
Πέρα από την αρχική κατηγοριοποίηση των προτροπών, η κάθε προτροπή επίσης κατηγοριοποίηθηκε περαιτέρω, με το κάθε θέμα της ερώτησης, όπως για παράδειγμα ανάπτυξη των μεθόδων που αφορούσαν μια οντότητα της εφαρμογής και ανάπτυξη ελέγχων αυτών των μεθόδων, εκ των οποίων ελέγχων χωρίστηκαν σε Integration Tests (έλεγχοι συμβατότητας) και σε Unit Tests (έλεγχοι μονάδας) [84, 85]. Οι διαφορές μεταξύ των δύο είναι οι εξής:

- **Unit Testing (Έλεγχος Μονάδας):** Το Unit Testing επικεντρώνεται στον έλεγχο μεμονωμένων κομματιών κώδικα, συνήθως μεθόδων ή συναρτήσεων, σε απομόνωση. Στοχεύει στην επαλήθευση ότι κάθε μονάδα του λογισμικού λειτουργεί σωστά.
- **Integration Testing (Έλεγχος Συμβατότητας):** Το Integration Testing εξετάζει πώς διαφορετικά μέρη του συστήματος λειτουργούν μαζί. Ελέγχει την αλληλεπίδραση μεταξύ διαφορετικών μονάδων ή συστατικών για να διασφαλίσει ότι συνεργάζονται σωστά ως σύνολο. Η ειδοποιός διαφορά είναι ότι το Unit Testing εστιάζει σε μεμονωμένα κομμάτια κώδικα, ενώ το Integration Testing εξετάζει πώς αυτά τα κομμάτια συνεργάζονται. [86]

Για την συλλογή των δεδομένων, δημιουργήθηκε μια περσόνα ενός προγραμματιστή για το μοντέλο [87, 88, 6], με τους κανόνες να ακολουθεί κάθε του απάντηση μια συγκεκριμένη δομή. Ζητήθηκε από το μοντέλο να αρχίζει κάθε μήνυμα με το θέμα της ερώτησης, τον τύπο της ερώτησης και έναν άριθμο που αναπαριστά την σειρά της ερώτησης για το συγκεκριμένο θέμα. Στη συνέχεια, ζητήθηκε από το μοντέλο να βάζει σχόλια στον κώδικα που πάραγε, υποδεικνύοντας την αρχή και το τέλος του παραγόμενου κώδικα, με τη μορφή *BEGIN_COPILOT_CODE / END_COPILOT_CODE*. Τέλος, ζητήθηκε από το μοντέλο να παράγει κώδικα που να ακολουθεί τις κατευθυντήριες γραμμές που ορίστηκαν από τον προγραμματιστή.



Σχήμα 2.7: Η μεθοδολογία δημιουργίας περσόνας για το μοντέλο, Δανεισμένο από [6]



Σχήμα 2.8: Η προτροπή που δόθηκε στο μοντέλο για την δημιουργία της περσόνας

Η επιλογή να ζητηθεί από το μοντέλο να υποδεικνύει τον κώδικα που παράγει έγινε προκειμένου να συλλεχθούν δεδομένα για την ποιότητα του κώδικα που παρήχθη, αλλά και για να καταγραφτούν οι φορές που ο προγραμματιστής επενέβη, με σκοπό να διορθώσει λάθη που το μοντέλο αδυνατούσε. Η τακτική αυτή προϋπόθετε ότι ο προγραμματιστής και χειριστής του μοντέλου θα είχε την δυνατότητα να αντιληφθεί τα λάθη που παρήχθησαν από το μοντέλο, και να τα διορθώσει, καθώς και να γνωρίζει πώς να προτρέψει το μοντέλο να απαντήσει την επιθυμητή απάντηση. Χωρίς αυτήν την απόφαση, το μοντέλο θα παρήγαγε κώδικα ο οποίος δεν θα ήταν λειτουργικός, καθώς υπήρχαν συντακτικά λάθη που δεν μπορούσε να παρατηρήσει.



```
1 // BEGIN_COPILOT_CODE
2 describe("and the user is following the creator", () => {
3   it("should return the playlist", async () => {
4     // Arrange
5     const playlist = await prisma.playlist.create({
6       data: {
7         name: faker.lorem.words(3),
8         description: faker.lorem.words(),
9         type: "CUSTOM",
10        visibility: "FOLLOWERS_ONLY",
11        user: {
12          connect: { id: mockAdmin.id },
13        },
14      },
15    });
16    ...
17 // END_COPILOT_CODE
```

Σχήμα 2.9: Παράδειγμα κώδικα που παρήχθη από το μοντέλο, με ορθογραφικό λάθος

Η διόρθωση λαθών από τον προγραμματιστή αποτέλεσε έσχατη λύση, καθώς πρώτη επιλογή ήταν η αποφυγή συγγραφής κώδικα από τον προγραμματιστή, μέσω επισύμανσης των λαθών στο μοντέλο, με την προσδοκία ότι το μοντέλο θα διόρθωνε το λάθος χωρίς την ανάγκη επέμβασης του προγραμματιστή. Τα λάθη που διορθώθηκαν από το μοντέλο μετά την προτροπή του προγραμματιστή σημειώθηκαν με σχόλια με πρόθημα *REWRITE*, για αλλαγές σε μία σειρά κώδικα, ενώ με σχόλια με πρόθημα *REVISION* για αλλαγές σε μικρά κομμάτια κώδικα.

```
1  data: {
2    // REVISION_1: Correctly connect the user and game to the review
3    // content: input.content,
4    // rating: input.rating,
5    // userId: ctx.user.id,
6    // gameId: input.gameId,
7    ...rest,
8    createdAt: new Date(),
9    updatedAt: new Date(),
10   user: {
11     connect: {
12       id: ctx.session.user.id,
13     },
14   },
15   game: {
16     connect: {
17       id: gameId,
18     },
19   },
20 },
21 ...
```

Σχήμα 2.10: Παράδειγμα διόρθωσης λάθους από το μοντέλο μεγάλης έκτασης, μετά την προτροπή του προγραμματιστή

● ● ●

```
1 // REWRITE_1: remove deprecated image field
2 //   image: faker.image.imageUrl(),
3 image: faker.image.url(),
```

Σχήμα 2.11: Παράδειγμα διόρθωσης λάθους από το μοντέλο μικρής έκτασης, μετά την προτροπή του προγραμματιστή

Κατά την διόρθωση λαθών, ο προγραμματιστής άφηνε σχόλια, εξηγώντας το λόγο που παρενέβη, με σκοπό την καλύτερη δυνατή τεκμηρίωση, αλλά και την εκμάθηση του μοντέλου, με σκοπό την αποφυγή μελλοντικών λαθών, σημειώνοντας τα κομμάτια κώδικα με τα σχόλια *BEGIN_NON_COPILOT_CODE / END_NON_COPILOT_CODE*.

```

1 // BEGIN_NON_COPILOT_CODE
2 // Explicit prisma relations are hard to work with, copilot struggles with them
3 getGames: publicProcedure
4   .input(z.object({ id: z.string().cuid2() }))
5   .query(
6     async ({
7       ctx,
8       input,
9     }): Promise<
10    Result<Array<GameToPlatform & { game: Game }>, TRPCError>
11  > => {
12    const result: Result<
13      Array<GameToPlatform & { game: Game }>,
14      TRPCError
15    > = await ctx.prisma.platform
16      .findUnique({
17        where: {
18          id: input.id,
19        },
20        select: {
21          games: {
22            include: {
23              game: true,
24            },
25          },
26        },
27      })
28    ...
29 // END_NON_COPILOT_CODE

```

Σχήμα 2.12: Παράδειγμα διόρθωσης λάθους από τον προγραμματιστή, μετά την αδυναμία του μοντέλου να διορθώσει το λάθος

Λάθη που διορθώθηκαν μέσω μιας γραμμής (όπως ονομασίες μεταβλητών ή συμβόλων), σημειώθηκαν με σχόλια με πρόθημα *MANUAL_REWRITE*. Επίσης, κατά την χειροκίνητη συγγραφή κώδικα, το μοντέλο του GitHub Copilot, έδινε στον προγραμματιστή προτάσεις διόρθωσης του κώδικα. Ορισμένες προτάσεις που έλυναν το πρόβλημα, ενσωματώθηκαν στον τελικό κώδικα, σημειώνοντας την αρχή και το τέλος του κώδικα που παρήχθη σε πραγματικό χρόνο, μέσω σχολίων *BEGIN_COPILOT_SUGGESTION* / *END_COPILOT_SUGGESTION*, σημειώνοντας την αρχή και το τέλος του προτεινόμενου κομματιού κώδικα.



```
1 // MANUAL_REWRITE: correct rating type
2 // rating: faker.datatype.number({ min: 1, max: 5 }),
3 rating: faker.number.int({ min: 1, max: 5 }),
```

Σχήμα 2.13: Παράδειγμα διόρθωσης λάθους μικρής έκτασης από τον προγραμματιστή, μετά την αδυναμία του μοντέλου να διορθώσει το λάθος

2.3.3 Ανάλυση Δεδομένων

Μετά την συλλογή των δεδομένων, ακολούθησε η ανάλυσή των ευρημάτων, μέσω ενός αναλυτή (lexer [89]), προκειμένου να μετρηθούν τα εξής:

- Ο αριθμός των περιστατικών επέμβασης του προγραμματιστή σε μεγάλες εκτάσεις κώδικα
- Ο αριθμός των περιστατικών επέμβασης του προγραμματιστή σε μικρές εκτάσεις κώδικα
- Ο αριθμός των περιστατικών που το μοντέλο διόρθωσε το λάθος σε μικρές εκτάσεις κώδικα
- Ο αριθμός των περιστατικών που το μοντέλο διόρθωσε μεγαλύτερα λάθη, έπειτα από προτροπή του προγραμματιστή

Τύπος	Μικροαλλαγές Copilot	Μικροαλλαγές προγραμματιστή
Backend	20	3
Integration Tests	26	7
Unit Tests	32	3

Πίνακας 2.1: Μικροαλλαγές ανά τύπο

Τύπος	Αλλαγές Copilot	Αλλαγές προγραμματιστή
Backend	3	10
Integration Tests	30	14
Unit Tests	4	3

Πίνακας 2.2: Αλλαγές μεγάλης έκτασης ανά τύπο

Τα δεδομένα λήφθηκαν μέσα από εικοσιοκτώ (28) αρχεία και πεντακόσιες εικοσιπέντε (525) προτροπές και συγκεκριμένα:

- Δεκαεπτά (17) αρχεία που αφορούν τον έλεγχο του κώδικα
- Έντεκα (11) αρχεία που αφορούν την ανάπτυξη του API

Παρατίθεται ένα διάγραμμα που αναπαριστά τα αποτελέσματα της ανάλυσης:



Σχήμα 2.14: Συνολικά μετρικά ανά τύπο

Όπως φαίνεται από το διάγραμμα 2.14, οι περισσότερες αλλαγές που πραγματοποιήθηκαν από την πλευρά του μοντέλου έγιναν σε μικρές εκτάσεις κώδικα, ενώ οι περισσότερες αλλαγές που πραγματοποιήθηκαν από την πλευρά του προγραμματιστή έγιναν σε μεγάλες εκτάσεις κώδικα. Σε κάθε τύπο αρχείου παρατηρείται αυτή η τάση, με στο προγραμματιστή να χρειάζεται να επεμβεί πολύ σπανιότερα σε σχέση με το μοντέλο και συγκεκριμένα, σε κάθε τύπο αρχείου, οι επεμβάσεις του προγραμματιστή για μικρές εκτάσεις κώδικα είναι οι λιγότερες.

Αξιοσημείωτη επίσης η παρατήρηση ότι στην περίπτωση των αρχειών που αφορούσαν το Unit Testing, οι αλλαγές σε μικρή έκταση κώδικα από το μοντέλο ήταν με διαφορά οι περισσότερες, ενώ παράλληλα οι υπόλοιπες αλλαγές ήταν ελάχιστες. Αυτό μπορεί να οφείλεται στη χρήση βιβλιοθηκών, με βάση τον έλεγχο του κώδικα και πιο συγκεκριμένα αυτή της faker.js[90]. Η συγκεκριμένη βιβλιοθήκη αφορά την δημιουργία ψεύτικων δεδομένων, στο ίδιο πνεύμα με το Lorem Ipsum [91], αλλά για δεδομένα που αφορούν ένα λογισμικό, όπως ημερομηνίες, ονόματα, διευθύνσεις ηλεκτρονικού ταχυδρομίου κ.α.. Η βιβλιοθήκη έχει λάβει πολλές εκδόσεις τα τελευταία χρόνια, με αλλαγές στον τρόπο που καλούνται οι μέθοδοι του API της, με αποτέλεσμα το μοντέλο να μην μπορεί να ακολουθήσει πάντοτε τις αλλαγές αυτές. Σημαντικό να σημειωθεί επίσης ότι το μοντέλο αδυνατούσε να παρατηρήσει τις αλλαγές που ζητήθηκαν από τον προγραμματιστή, με αποτέλεσμα να ζητηθεί μέσω διαφορετικής προτροπής η αλλαγή της ίδιας κλήσης μιας μεθόδου πάνω από μία φορά στο ίδιο αρχείο.

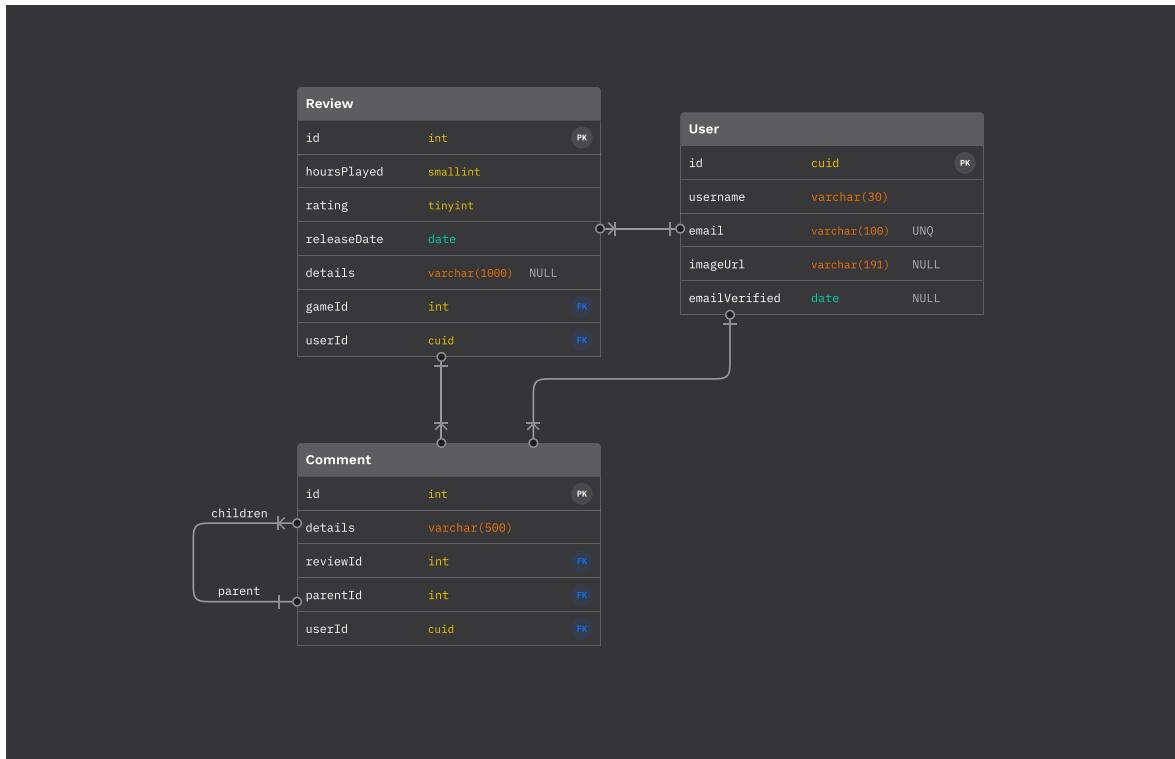
Στην περίπτωση των αρχείων που αφορούσαν το Backend, οι απαιτούμενες αλλαγές ήταν οι λιγότερες σε πλήθος, με τη διαφορά ότι οι περισσότερες επεμβάσεις του προγραμματιστή αφορούσαν την γραφή ολόκληρων μεθόδων. Η απουσία των απαραίτητων αλλαγών, οφείλεται στο γεγονός ότι η βασική λογική πίσω από ένα μεγάλο ποσοστό των μεθόδων αφορά την γραφή βασικών CRUD μεθόδων, οι οποίες είναι προκαθορισμένες και δεν απαιτούν αλλαγές. Αυτό έχει ως αποτέλεσμα το μοντέλο να μην χρειάζεται να προτρέπεται να αλλάξει την λογική των μεθόδων, αλλά μόνο την υλοποίηση τους. Στις περιπτώσεις μεθόδων που επηρέαζαν οντότητες με σχέσεις M:N [92], το μοντέλο του GitHub Copilot αδυνατούσε να ακολουθήσει την λογική, με αποτέλεσμα να χρειαστεί

να υλοποιηθούν από τον προγραμματιστή.

Παράδειγμα αποτελεί η περίπτωση της οντότητας του σχολίου. Τα σχόλια πραγματοποιούνται σε κριτικές των χρηστών, παρομοίως όπως σε άλλες εφαρμογές κοινωνικών δικτύων. Τα σχόλια πραγματοποιούνται σε κριτικές των χρηστών, παρομοίως όπως σε άλλες εφαρμογές κοινωνικών δικτύων. Κάθε σχόλιο έχει μια σχέση 1:M με την κριτική, μια σχέση 1:M με τον χρήστη, μια σχέση M:1 με τις αντιδράσεις τύπου Mou αρέσει (Likes), αλλά μπορεί να έχει και άλλα σχόλια που συνδέονται με αυτό, δημιουργώντας ένα "νήμα" (thread) σχολίων, παρόμοιο με έναδιατεταγμένο Κ-οστο δέντρο [93]. Η λογική πίσω από την υλοποίηση των μεθόδων που αφορούν τα σχόλια, είναι περίπλοκη, με τα ερωτήματα που απαιτούνται να σταλθούν προς στη βάση μέσω του Prisma να είναι πολλά και περίπλοκα. Το μοντέλο του GitHub Copilot αδυνατούσε να ακολουθήσει την λογική, με αποτέλεσμα να χρειαστεί να υλοποιηθούν από τον προγραμματιστή.

```
1 const baseSelectClause = Prisma.validator<Prisma.CommentSelect>()({  
2   _count: {  
3     select: {  
4       likes: true,  
5       children: false,  
6     },  
7   },  
8   id: true,  
9   content: true,  
10  user: {  
11    select: {  
12      image: true,  
13      id: true,  
14      name: true,  
15    },  
16  },  
17  reviewId: true,  
18  createdAt: true,  
19  updatedAt: true,  
20});  
21  
22 const parentSelectClause = {  
23   ...baseSelectClause,  
24   // Don't take the children (this is where the current comment is)  
25   children: false,  
26   // Don't take the parent (this is two depths above)  
27   parent: false,  
28 };  
29  
30 const childrenSelectClause = {  
31   // Don't take the parent (current comment)  
32   ...baseSelectClause,  
33   parent: false,  
34   // Take the first depth of children  
35   children: {  
36     // Take the second depth of children, and no more than that  
37     select: parentSelectClause,  
38   },  
39};
```

Σχήμα 2.15: Παράδειγμα μεθόδου που αφορά τα σχόλια, με πολλαπλές σχέσεις M:N



Σχήμα 2.16: Σχεσιακό διάγραμμα της οντότητας του σχολίου

Στη περίπτωση της οντότητας της λίστας (Playlist), η λογική της λειτουργικότητας αυτής είναι να δίνει στο χρήστη τη δυνατότητα να δημιουργεί λίστες με τα αγαπημένα του βιντεοπαιχνίδια. Ο κάθε χρήστης έχει κατά την εγγραφή του στην εφαρμογή πέντε (5) λίστες:

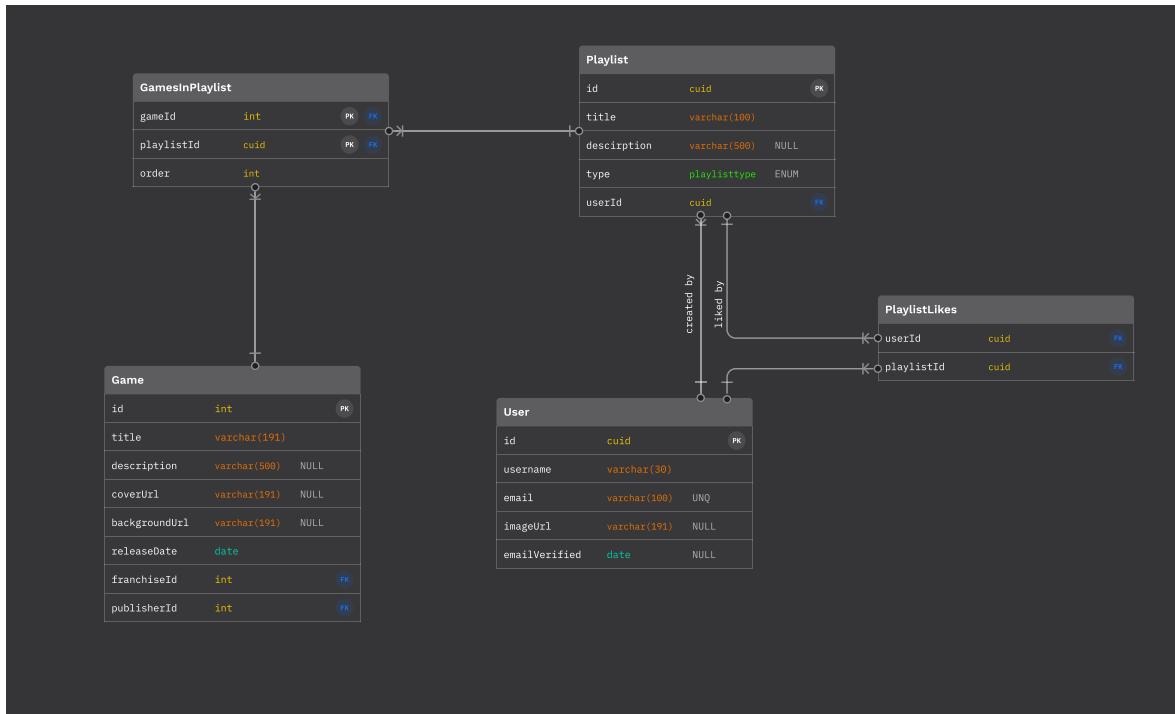
- Αγαπημένα (Liked): Τα βιντεοπαιχνίδια που άρεσαν στον χρήστη, η πιο εύχρηστη λίστα για τον χρήστη
- Λίστα αναμονής (Backlog): Τα βιντεοπαιχνίδια που θέλει να παίξει στο μέλλον
- Ολοκληρωμένα (Completed): Τα βιντεοπαιχνίδια που έχει παίξει και φτάσει στο τέλος τους στο παρελθόν παίξει στο μέλλον
- Ενεργά (Playing): Τα βιντεοπαιχνίδια που παίζει αυτή τη χρονική περίοδο
- Εγκαταλειμμένα (Dropped): Τα βιντεοπαιχνίδια που δεν του άρεσαν και τα έχει εγκαταλείψει

Ο χρήστης έχει το δικαίωμα να δημιουργήσει όσες ακόμα προσαρμοσμένες (Custom) λίστες θέλει, και να τις μοιραστεί είτε με τους ακολούθους του, είτε να τις κάνει δημόσιες, είτε να τις κρατήσει ως ιδιωτικές.

```
● ● ●  
1 export const createPlaylistSchema = z.object({  
2   name: z.string(),  
3   description: z.string().optional(),  
4   type: z.enum([  
5     "BACKLOG",  
6     "LIKED",  
7     "COMPLETED",  
8     "PLAYING",  
9     "DROPPED",  
10    "CUSTOM",  
11  ]),  
12  visibility: z.enum(["PUBLIC", "PRIVATE", "FOLLOWERS_ONLY"]),  
13});
```

Σχήμα 2.17: Τύπος δεδομένων για την δημιουργία της οντότητας της λίστας

Κάθε λίστα έχει μια σχέση 1:M με τον χρήστη, μια σχέση 1:M με τις αντιδράσεις, και μία σχέση N:M με βιντεοπαιχνίδια. Η σχέση με τα βιντεοπαιχνίδια υλοποιείται μέσω ενός πίνακα σύνδεσης [94], ο οποίος, πέρα από τα ξένα κλειδιά του βιντεοπαιχνιδιού και της λίστας, περιέχει και το πεδίο order, το οποίο καθορίζει τη σειρά με την οποία τα βιντεοπαιχνίδια εμφανίζονται στη λίστα. Η λογική πίσω από την υλοποίηση προϋπόθεται τη χρήση μιας διπλά συνδεμένης λίστας, προκειμένου να διατηρείται η επιθυμητή σειρά, αν ο χρήστης αποφασίσει να αλλάξει τη σειρά με την οποία τα βιντεοπαιχνίδια θα εμφανίζονται στη λίστα.



Σχήμα 2.18: Σχεσιακό διάγραμμα της οντότητας της λίστας

Το μοντέλο του GitHub Copilot αδυνατούσε να ακολουθήσει την λογική, με αποτέλεσμα να χρειαστεί να υλοποιηθούν εξ ολοκλήρου από τον προγραμματιστή.

```

1 updateOrder: protectedProcedure
2   .input(
3     z.object({
4       id: z.string().cuid2(),
5       gameId: z.string().cuid2(),
6       order: z.number().int(),
7     })
8   )
9   .mutation(
10    async ({
11      ctx,
12      input,
13    }): Promise<Result<Playlist & { games: Array<GameToPlaylist & { game: Game }> }, TRPCError>> => {
14      const whereClause = Prisma.validator<Prisma.PlaylistWhereInput>()({
15        id: input.id,
16        deleted: null,
17        user: { id: ctx.session.user.id },
18      });
19
20      const oldOrder = await ctx.prisma.gameToPlaylist
21        .findUnique({
22          where: {
23            gameId_playlistId: {
24              gameId: input.gameId,
25              playlistId: input.id,
26            },
27          },
28          select: { order: true },
29        })
30        .then((res) => Ok(res), handlePrismaError);
31
32      if (!oldOrder.ok) return oldOrder;
33
34      const increment = ctx.prisma.playlist.update({
35        where: whereClause,
36        data: {
37          games: {
38            updateMany: {
39              where: {
40                order: {
41                  gte: input.order,
42                },
43              },
44              data: {
45                order: {
46                  increment: 1,
47                },
48              },
49            },
50          },
51        },
52      });
53

```

Σχήμα 2.19: Παράδειγμα μεθόδου που αφορά την αλλαγή της σειράς των βιντεοπαιχνιδιών στη λίστα, *Μέρος 1*

```

1  const decrement = ctx.prisma.playlist.update({
2    where: whereClause,
3    data: {
4      games: {
5        updateMany: {
6          where: {
7            order: {
8              lte: input.order,
9            },
10           },
11           data: {
12             order: {
13               decrement: 1,
14             },
15           },
16           },
17         },
18       },
19     });
20
21 const update = ctx.prisma.playlist.update({
22   where: whereClause,
23   data: {
24     games: {
25       update: {
26         where: {
27           gameId_playlistId: {
28             gameId: input.gameId,
29             playlistId: input.id,
30           },
31         },
32         data: {
33           order: input.order,
34         },
35       },
36     },
37   },
38   include: {
39     games: {
40       orderBy: {
41         order: "asc",
42       },
43       include: {
44         game: true,
45       },
46     },
47   },
48 });
49
50 switch (true) {
51   case oldOrder instanceof Ok && oldOrder.val instanceof Object && typeof oldOrder.val.order === "number":
52     const result: Result<Playlist & { games: Array<GameToPlaylist & { game: Game }> }, TRPCError> =
53       oldOrder.val.order > input.order
54         ? await ctx.prisma.$transaction([increment, update]).then(([res] => Ok(res[1]), handlePrismaError))
55         : await ctx.prisma.$transaction([decrement, update]).then(([res] => Ok(res[1]), handlePrismaError));
56     return result;
57   default:
58     return new Err(new TRPCError({ code: "INTERNAL_SERVER_ERROR" }));
59   }
60 }
61 ),

```

Σχήμα 2.20: Παράδειγμα μεθόδου που αφορά την αλλαγή της σειράς των βιντεοπαιχνιδιών στη λίστα, *Μέρος 2*

Πέρα από τη σειρά των βιντεοπαιχνιδιών, σημαντική ήταν και η εξασφάλιση της ορθής ορατότητας της λίστας σε άλλους χρήστες, χωρίς να επιτρέπεται η πρόσθιαση σε λίστες που δεν είναι είτε δημόσιες, είτε ανήκουν σε χρήστες που ο χρήστης ακολουθεί, είτε είναι προσωπικές λίστες του ίδιου. Στην περίπτωση αυτή, αν και το μοντέλο ακολούθησε τη σωστή λογική, αδυνάτισε να βρει το σωστό ερώτημα προς τη βάση όσο αφορά την ορατότητα της λίστας προς τους ακολούθους του χρήστη.

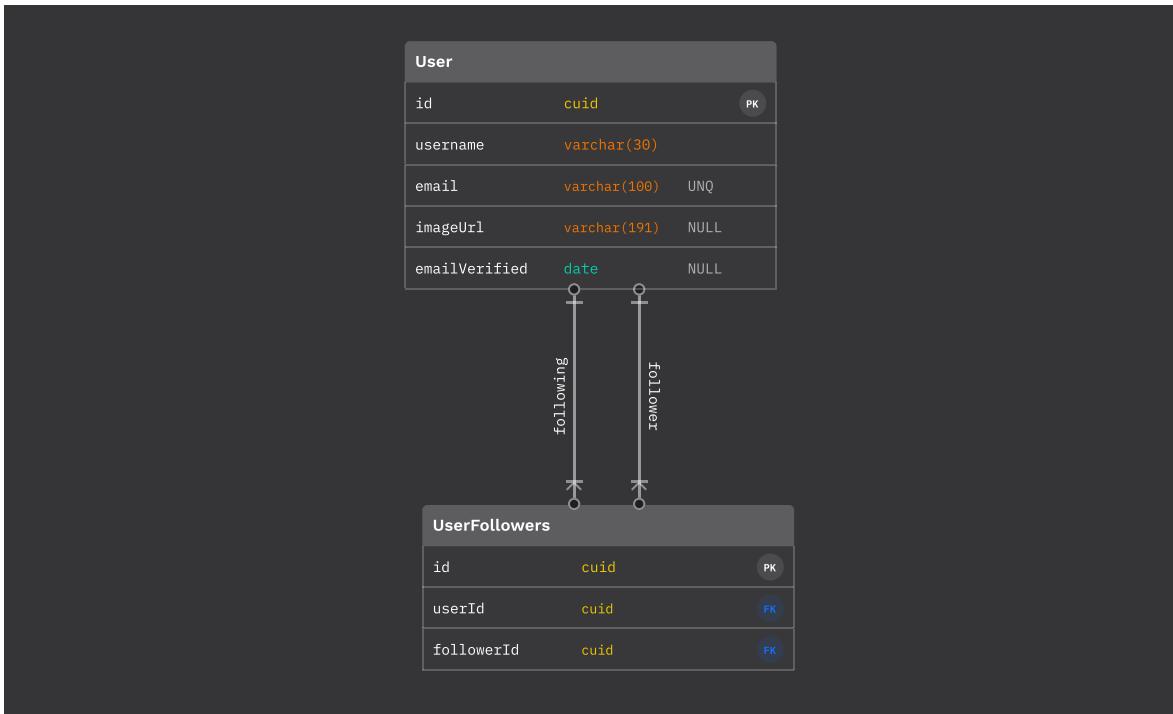
```

1  getById: publicProcedure
2    .input(z.object({ id: z.string().cuid2() }))
3    .query(async ({ ctx, input }): Promise<Result<Playlist, TRPCError>> => {
4      const whereClause: Prisma.PlaylistWhereInput = {
5        id: input.id,
6        deleted: null,
7        OR: [{ visibility: "PUBLIC" }],
8      };
9
10     if (ctx.session?.user?.id) {
11       (whereClause.OR as Prisma.PlaylistWhereInput[]).push(
12         // REWRITE_2: use user relation instead of userId
13         // {userId: ctx.session.user.id},
14         { user: { id: ctx.session.user.id } },
15         {
16           visibility: "FOLLOWERS_ONLY",
17           // END_COPILOT_CODE
18           // BEGIN_NON_COPILOT_CODE
19           // Copilot didn't find the bug
20           // user: { followers: { some: { id: ctx.session.user.id } } },
21           user: {
22             followers: { some: { follower: { id: ctx.session.user.id } } },
23           },
24           // END_NON_COPILOT_CODE
25         }
26       );
27     }
28
29     const result: Result<Playlist, TRPCError> = await ctx.prisma.playlist
30       // REWRITE_1: use findfirst instead of findUnique
31       // .findUnique({ where: whereClause })
32       .findFirst({ where: whereClause })
33       .then((res) => {
34         return res
35         ? Ok(res)
36         : new Err(
37           new TRPCError({
38             code: "NOT_FOUND",
39             message: "Playlist not found",
40           })
41         );
42       }, handlePrismaError);
43
44     return result;
45   },

```

Σχήμα 2.21: Παράδειγμα μεθόδου που αφορά την ανάκτηση λίστας με βάση το αναγνωριστικό της

Όπως παρατηρείται στο διάγραμμα ;;, το μοντέλο ορθά αναγνώρισε ότι σε περίπτωση που μια λίστα δημοσιεύονταν μόνο στους ακολούθους του δημιουργού της, θα έπρεπε ο χρήστης που την ανακτά θα έπρεπε να ανήκει στην λίστα των ακολούθων του δημιουργού, η υλοποίηση του κώδικα δεν ήταν σωστή, καθώς αγνόησε την M:N σχέση μεταξύ των χρηστών και των ακολούθων τους.



Σχήμα 2.22: Σχεσιακό διάγραμμα της σχέσης μεταξύ χρηστών και ακολούθων

Στις περιπτώσεις των αρχείων που αφορούσαν το Integration Testing, οι αλλαγές ήταν οι περίσσοτερες συγκριτικά με τις περιπτώσεις ανάπτυξης του API και του Unit Testing και οι αλλαγές από την πλευρά του μοντέλου ήταν αρκετά πιο ισορροπημένες, με τις αλλαγές σε μεγάλη έκταση κώδικα να υπερβαίνουν αυτών της μικρής έκτασης κώδικα.

Στην περίπτωση αυτή, ρόλο έπαιξε το γεγονός ότι κατά την γραφή των ελέγχων, το εργαλείο Jest, παρέχει τη δυνατότητα στον προγραμματιστή να συντάξει μια περιγραφή για τον έλεγχο, μέσω των μεθόδων *describe* και *it*. Οι μέθοδοι αυτές, παίρνουν ως πρώτο όρισμα μια συμβολοσειρά που περιγράφει τον έλεγχο, και ως δεύτερο όρισμα μια συνάρτηση *callback* [95] που περιέχει τον κώδικα του ελέγχου. Η περιγραφή του ελέγχου, ακολουθεί την ανάπτυξη **Λογισμικού Καθοδηγούμενη Από Τη Συμπεριφορά** (Behavior Driven Development, BDD).

Η BDD είναι μια προσέγγιση στην ανάπτυξη λογισμικού που επικεντρώνεται στη συμπεριφορά του συστήματος από την οπτική γωνία του χρήστη. Ενθαρρύνει τη συνεργασία μεταξύ προγραμματιστών, ελεγκτών και μη τεχνικών ενδιαφερομένων, χρησιμοποιώντας μια κοινή γλώσσα για να περιγράψει την επιθυμητή συμπεριφορά του λογισμικού. Συνήθως περιλαμβάνει τη συγγραφή σεναρίων ή παραδειγμάτων που περιγράφουν πώς το λογισμικό θα πρέπει να συμπεριφέρεται σε διάφορες καταστάσεις, συχνά χρησιμοποιώντας τη μορφή 'Δεδομένου-Όταν-Τότε' ('Given-When-Then'). Αυτά τα σενάρια λειτουργούν τόσο ως προδιαγραφές όσο και ως περιπτώσεις δοκιμών, βοηθώντας να διασφαλιστεί ότι το αναπτυγμένο λογισμικό ανταποκρίνεται στις προβλεπόμενες απαιτήσεις και συμπεριφέρεται σωστά από την οπτική γωνία του χρήστη. [96, 97, 98]



```
1 describe("When creating a feature", () => {
2   describe("and the user is not authenticated", () => {
3     it("should throw an error", async () => {
4       ...

```

Σχήμα 2.23: Παράδειγμα μεθόδου που αφορά την δημιουργία ενός ελέγχου, με χρήση των μεθόδων *describe* και *it*

```
PASS  __tests__/api/game.test.ts
When creating a game
  and the user is not authenticated
    ✓ should throw an error (6 ms)
  and the user is authenticated
    and the user is not an admin
      ✓ should throw an error (1 ms)
    and the user is an admin
      and either the publisher or the franchise don't exist
        ✓ should return an error (4 ms)
    and the publisher and franchise exist
      ✓ should create a game successfully (3 ms)
```

Σχήμα 2.24: Παράδειγμα αποτελέσματος ελέγχου σε διεπαφή τερματικού, με χρήση των μεθόδων *describe* και *it*, μέσω του εργαλείου Jest

Επειδή το μοντέλο δουλεύει με next token prediction, και έλεγχοι γράφονται ο ένας μετά τον άλλο, το μοντέλο κατά τη συγγραφή των ελέγχων, λόγω των περιγραφών που περιέχονται στις μεθόδους, προσπαθούσε να γράψει τον έλεγχο με βάση την περιγραφή, αντί του κάθικα της μεθόδου, με αποτέλεσμα να οδηγείται συχνά σε λάθη και ανακρίβειες. Αξιοσημείωτη είναι η προσθήκη της δυνατότητας του GitHub Copilot Chat να γράψει ελέγχους για μια μέθοδο μέσω μιας εντολής στο πλαίσιο προτροπής με το μοντέλο [99] τον Μάρτιο του 2024, ωστόσο κατά την διάρκεια της δοκιμής του μοντέλου, η συγκεκριμένη δυνατότητα δεν ήταν διαθέσιμη.

Για την διαφορά μεταξύ αναγκαίων αλλαγών στις περιπτώσεις των αρχείων που αφορούσαν το Unit Testing και το Integration Testing, συνέβαλε και το γεγονός στην περίπτωση των ελέγχων μονάδας, οι έλεγχοι που πραγματοποιούνται είναι πιο απλοί και πολλές φορές χρησιμοποιούνται τεχνικές εικονικής αναπαράστασης εξαρτήσεων, γνωστές και ως τεχνικές προσομοίωσης εξαρτήσεων (dependency mocking). Η προσομοίωση εξαρτήσεων είναι μια τεχνική που χρησιμοποιείται στους ελέγχους λογισμικού, όπου δημιουργούνται ψεύτικα ή προσομοιωμένα αντικείμενα που μιμούνται τη συμπεριφορά πραγματικών εξαρτήσεων ενός συστήματος. Αυτό επιτρέπει στους προγραμματιστές να ελέγχουν μια μονάδα κάθικα σε απομόνωση, χωρίς να επηρεάζονται από εξωτερικές

εξαρτήσεις όπως βάσεις δεδομένων, υπηρεσίες δικτύου ή πολύπλοκες βιβλιοθήκες. Η προσομοίωση εξαρτήσεων βοηθά στη δημιουργία πιο ελεγχόμενων και προβλέψιμων συνθηκών δοκιμής, επιτρέποντας τον έλεγχο διαφόρων σεναρίων και καταστάσεων σφάλματος που μπορεί να είναι δύσκολο να αναπαραχθούν με πραγματικές εξαρτήσεις [100]. Αντίθετα, στην περίπτωση των ελέγχων ενσωμάτωσης, οι εξαρτήσεις είναι πραγματικές, και οι ελέγχοι που πραγματοποιούνται είναι πιο πολύπλοκοι και απαιτούν την ύπαρξη πραγματικών εξαρτήσεων, και χρειάζονται περισσότερη σκέψη για την υλοποίησή τους. Παρατίθενται δύο παραδείγματα ελέγχων, ένα από την περίπτωση των ελέγχων μονάδας και ένα από την περίπτωση των ελέγχων ενσωμάτωσης, για τον ίδιο έλεγχο.

```

1  describe("When creating a game", () => {
2    describe("and the user is authenticated", () => {
3      describe("and the user is an admin", () => {
4        describe("and the publisher and franchise exist", () => {
5          it("should create a game successfully", async () => {
6            // Arrange
7            const caller = appRouter.createCaller({
8              prisma: mockCtx.prisma,
9              session: mockAdminSession,
10            });
11
12            const gameId = createId();
13            const franchiseId = createId();
14
15            const gameData = {
16              name: faker.company.name(),
17              description: faker.lorem.words(),
18              coverImage: faker.image.url(),
19              backgroundImage: faker.image.url(),
20              releaseDate: new Date(),
21              franchiseId: franchiseId,
22              publisherId: gameId,
23            };
24
25            mockCtx.prisma.game.create.mockResolvedValue({
26              ...gameData,
27              id: gameId(),
28            });
29
30            // Act
31            const result = await caller.game.create(gameData);
32
33            // Assert
34            expect(result.ok).toBe(true);
35            expect(mockCtx.prisma.game.create).toHaveBeenCalledTimes(1);
36            expect(mockCtx.prisma.game.create).toHaveBeenCalledWith({
37              data: {
38                name: gameData.name,
39                description: gameData.description,
40                coverImage: gameData.coverImage,
41                backgroundImage: gameData.backgroundImage,
42                releaseDate: gameData.releaseDate,
43                franchise: {
44                  connect: {
45                    id: gameData.franchiseId,
46                  },
47                },
48                publisher: {
49                  connect: {
50                    id: gameData.publisherId,
51                  },
52                },
53              },
54            });
55          });
56        });
57      });
58    });
59  });

```

Σχήμα 2.25: Παράδειγμα ελέγχου μονάδας για την επιτυχημένη δημιουργία παιχνιδιού



```
1  describe("When creating a game", () => {
2      describe("and the user is an admin", () => {
3          describe("and the franchise and publisher exist", () => {
4              it("should create a game", async () => {
5                  // Arrange
6                  const franchise = await prisma.franchise.create({
7                      data: {
8                          name: faker.company.name(),
9                          // REWRITE_2: add description and image
10                         description: faker.company.catchPhrase(),
11                         image: faker.image.url(),
12                         },
13                     });
14
15                  const publisher = await prisma.publisher.create({
16                      data: {
17                          name: faker.company.name(),
18                          // REWRITE_2: add description and image
19                          description: faker.company.catchPhrase(),
20                          image: faker.image.url(),
21                          },
22                     });
23
24                  const game: z.infer<typeof createGameSchema> = {
25                      name: faker.company.name(),
26                      description: faker.company.catchPhrase(),
27                      coverImage: faker.image.url(),
28                      backgroundImage: faker.image.url(),
29                      releaseDate: new Date(),
30                      franchiseId: franchise.id,
31                      publisherId: publisher.id,
32                  };
33
34                  // Act
35                  const result = await adminCaller.game.create(game);
36
37                  // Assert
38                  expect(result.ok).toBe(true);
39                  expect(result.val).toMatchObject(game);
40              });
41          });
42      });
43  });
```

Σχήμα 2.26: Παράδειγμα ελέγχου συμβότητας για την επιτυχημένη δημιουργία παιχνιδιού

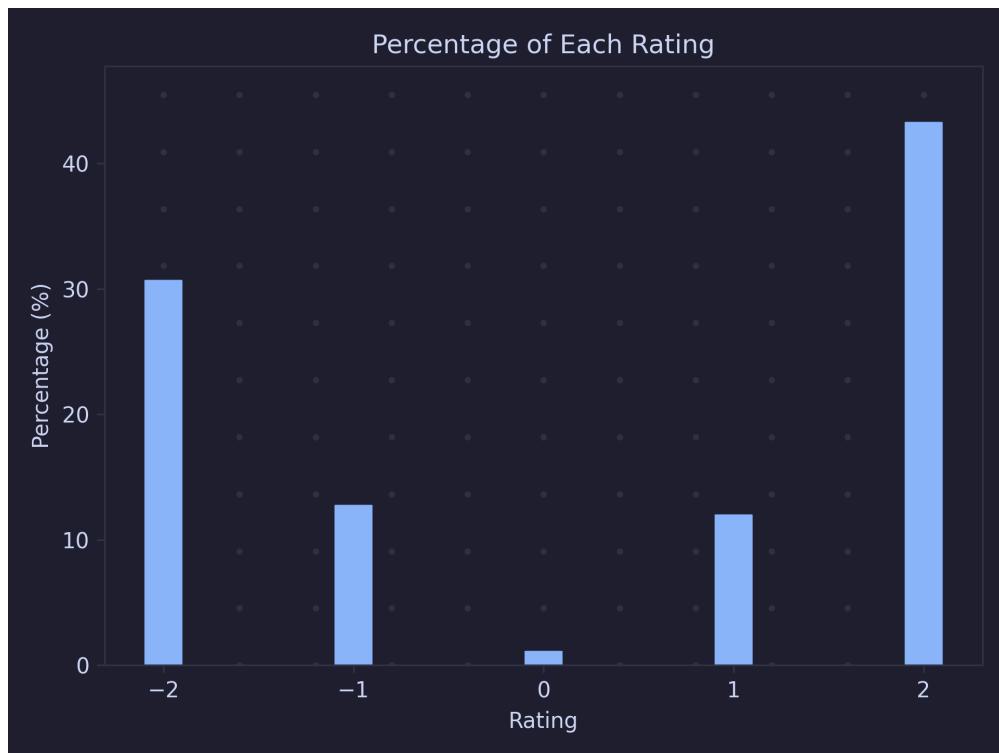
Εκ πρώτης όψεως, ο κώδικας στο Σχήμα ;; φαίνεται να είναι πιο περίπλοκος αυτού στο Σχήμα ;;, ωστόσο, στην περίπτωση του ελέγχου μονάδας, οι εξαρτήσεις είναι προσομοιωμένες. Δεν υπάρχει κάποιος πραγματικός χρήστης, ή κάποια πραγματική βάση δεδομένων. Αντίθετα, στην περίπτωση του ελέγχου συμβατότητας, οι εξαρτήσεις είναι πραγματικές, και ο έλεγχος πρέπει να ελέγξει την συμβατότητα του συστήματος με μια πραγματική βάση δεδομένων, ένα πραγματικό χρήστη και μετά από τον κάθε έλεγχο, πρέπει να είναι σίγουρο ότι εσωτερική κατάσταση του προγράμματος και της βάσης είναι κοινή για όλες τις περιπτώσεις, αποφεύγοντας περιπτώσεις ασταθών δοκιμών (flaky tests), όπου το αποτέλεσμα της δοκιμής δεν είναι πάντοτε αμετάβλητο. [101]

2.4 Αξιολόγηση Απαντήσεων

Κατά την αλληλεπίδραση με το μοντέλο, για κάθε απάντηση που δόθηκε έπειτα προτροπής, πραγματοποιήθηκε μια αξιολόγηση σε μια κλίμακα μεταξύ μείον δύο (-2) και δύο (2), με το μείον δύο να αντιστοιχεί σε μια απάντηση η οποία κρίθηκε ως εντελώς λάθος, το μείον ένα (-1) σε μια απάντηση η οποία κρίθηκε ως μερικώς λάθος, το μηδέν (0) σε μια απάντηση η οποία κρίθηκε ως αδιάφορη προς την ορθότητά της, το ένα (1) σε μια απάντηση η οποία κρίθηκε ως σωστή και το δύο (2) σε μια απάντηση η οποία κρίθηκε ως εντελώς σωστή.

Η επιλογή μιας κλίμακας η οποία εκτείνεται από ‘πολύ κακό’ έως ‘πολύ καλό’, αποσκοπεί στην καλύτερη αναπαράσταση της ποικιλίας και της πολυπλοκότητας της αξιολόγησης μιας απάντησης και στην δημιουργία ενός συνόλου ασαφούς λογικής (fuzzy logic) για την αξιολόγηση των απαντήσεων [102, 103, 104].

Η αξιολόγηση των απαντήσεων αποσκοπεί στην εξάσκηση ενός μοντέλου μηχανικής μάθησης, προκειμένου να βελτιωθεί η απόδοσή του στην παραγωγή απαντήσεων, με βάση την ορθότερη προτροπή, του οποίου η ανάλυση γίνεται στο επόμενο κεφάλαιο.



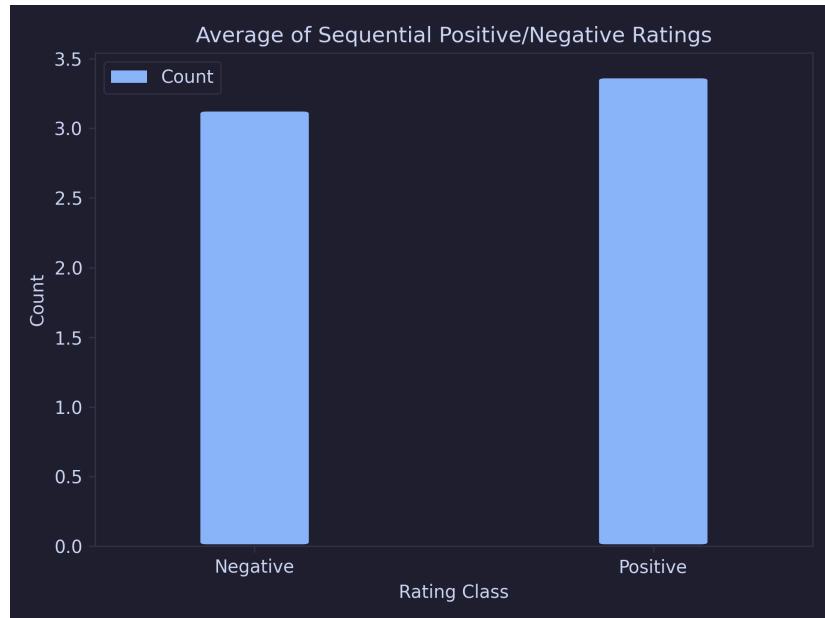
Σχήμα 2.27: Ποσοστό αξιολόγησης απαντήσεων



Σχήμα 2.28: Αριθμός αξιολόγησης απαντήσεων

Από τα σχήματα :: και ::, παρατηρείται ότι η πλειοψηφία των απαντήσεων αξιολογήθηκαν θετικά, με το πενήντα πέντε τοις εκατό (55.3435%), με σύνολο 290 θετικών αξιολογήσεων, έναντι του σαράντα τρία τοις εκατό (43.5115%), με με σύνολο 228 αρν-

ητικών αξιολογήσεων. Ο συνηθέστερος βαθμός αξιολόγησης ήταν το δύο (2), με σύνολο διακοσίων εικοσιεπτά (227) αξιολογήσεων, ενώ ο σπανιότερος βαθμός αξιολόγησης ήταν το μηδέν (0), με σύνολο έξι (6) αξιολογήσεων.



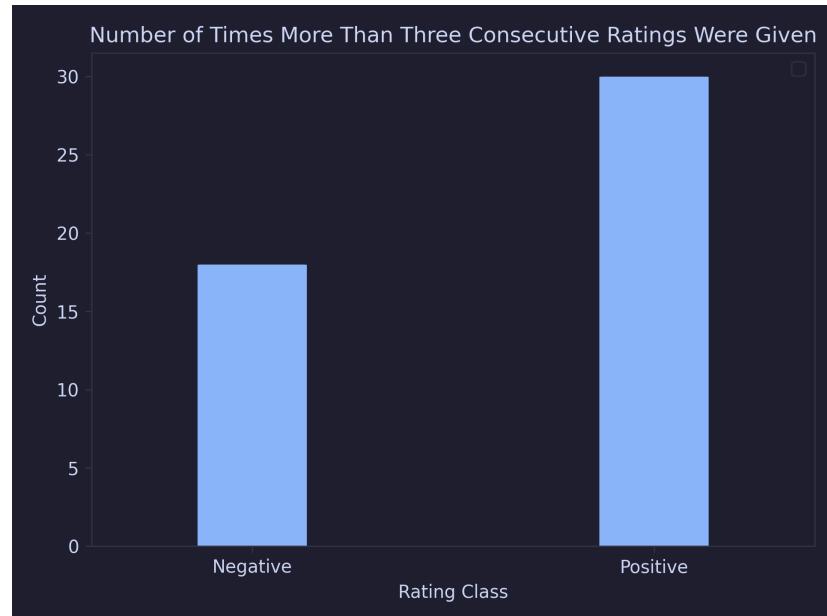
Σχήμα 2.29: Μέσος όρος συνεχόμενης αξιολόγησης απαντήσεων



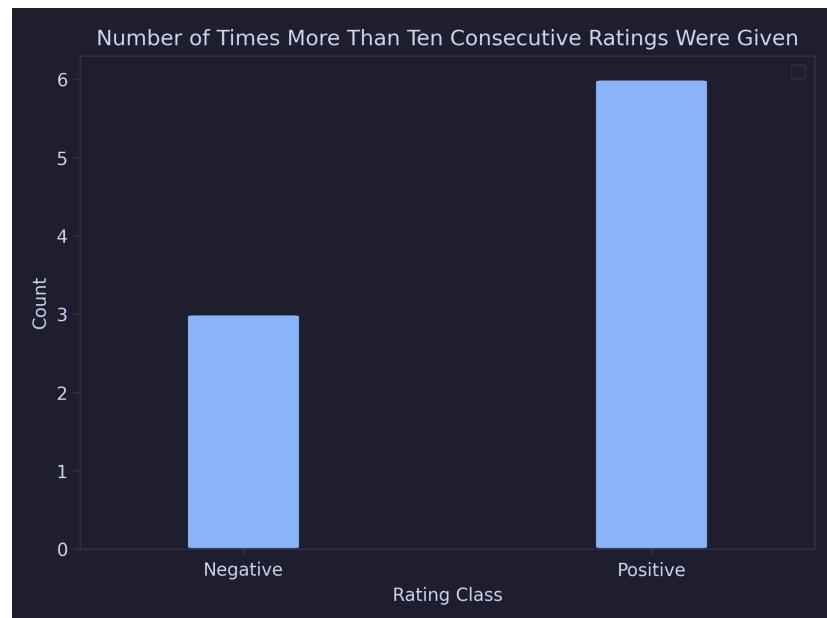
Σχήμα 2.30: Μέγιστη συνεχόμενη αξιολόγηση απαντήσεων

Το μοντέλο φαίνεται να έχει την τάση να παράγει απαντήσεις οι οποίες βασίζονται πολύ στην αρχική παρότρυνση από την προτροπή του προγραμματιστή. Αυτό αποδεικνύεται και από το σχήμα 2.30, όπου παρατηρείται ότι ο μέγιστος αριθμός συνεχόμενων αρνητικών αξιολογήσεων είναι δεκατέσσερεις (14), ενώ ο μέγιστος αριθμός συνεχόμενων θετικών αξιολογήσεων είναι δεκατρείς (16). Στην περίπτωση των συνεχόμενων θετικών αξιολογήσεων, οι προτροπές αφορούσαν 3 διαφορετικά θέματα, τους ελέγχους

της λειτουργίας των κριτικών, την ανάπτυξη της λειτουργίας των λιστών, καθώς και τους ελέγχους της. Στην περίπτωση των συνεχόμενων αρνητικών αξιολογήσεων, οι προτροπές αφορούσαν μόνο τους ελέγχους μιας λειτουργίας.

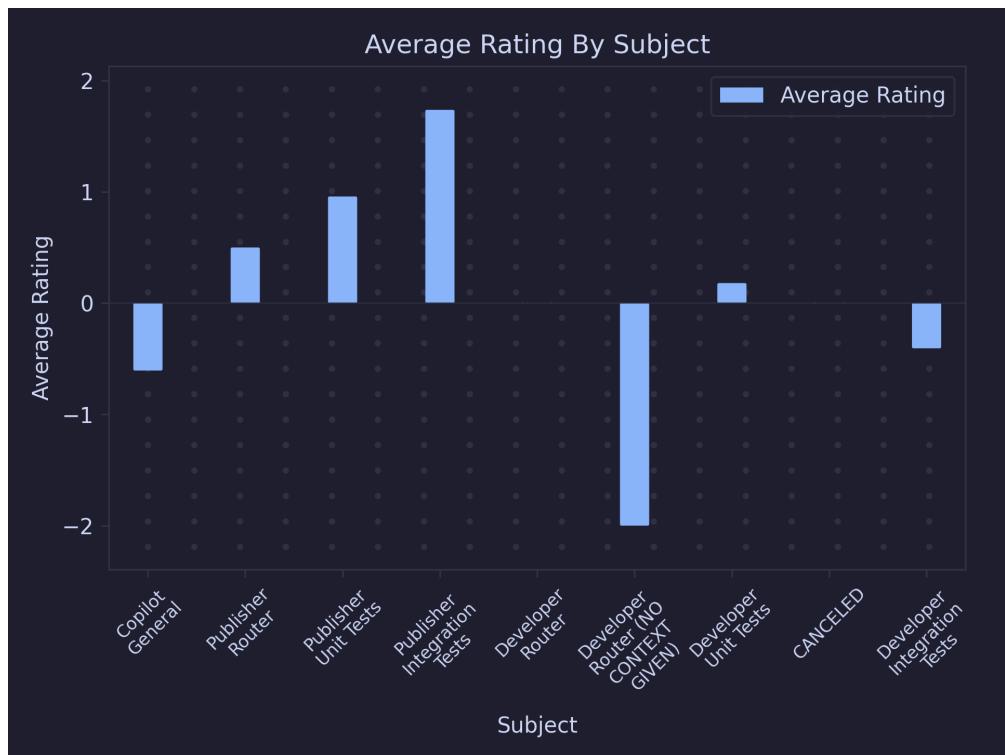


Σχήμα 2.31: Αριθμός εμφανίσεων συνεχόμενων αξιολογήσεων άνω των τριών (3)

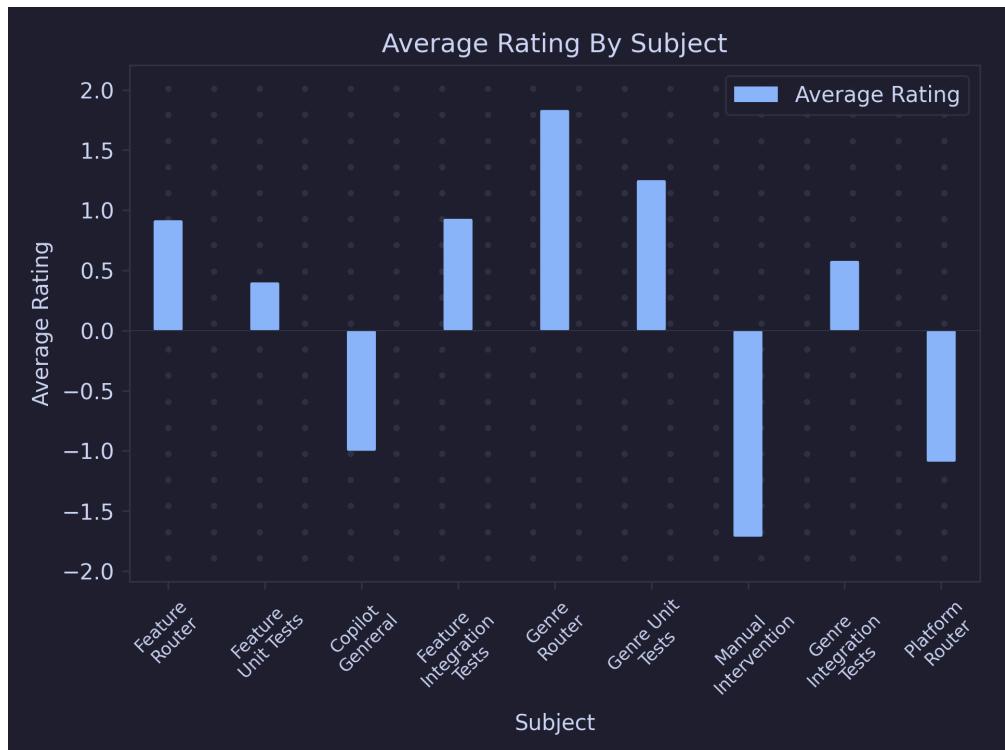


Σχήμα 2.32: Αριθμός εμφανίσεων συνεχόμενων αξιολογήσεων άνω των δέκα (10)

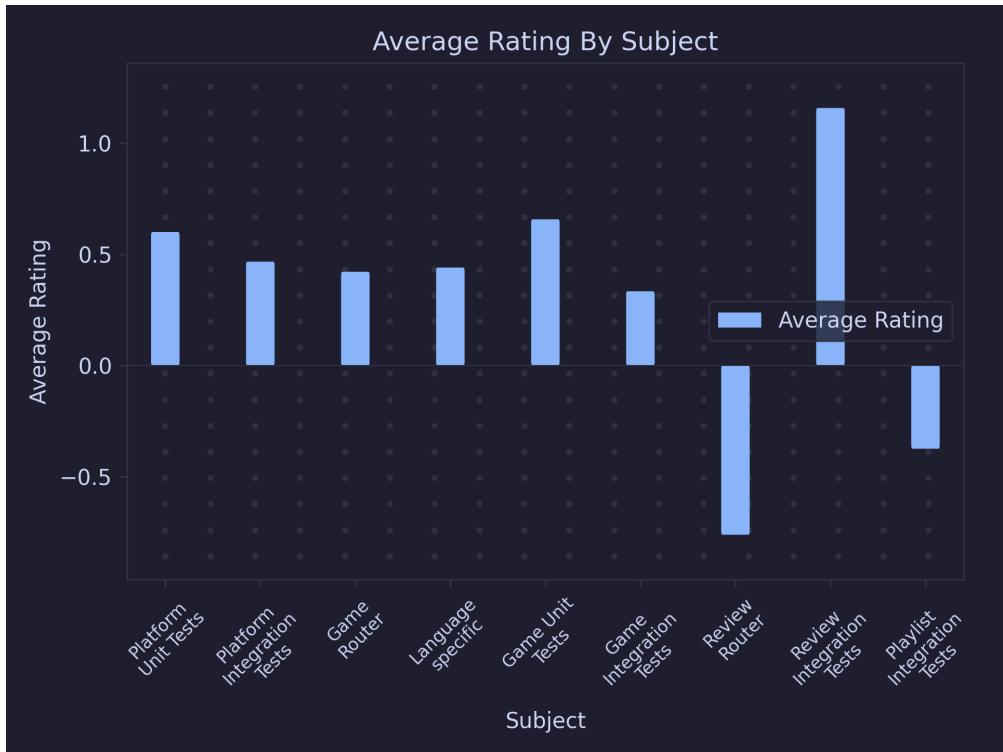
Για την περίπτωση των άνω των δέκα (10) συνεχόμενων θετικών αξιολογήσεων, κατά μέσο όρο, αφορούσαν δύο (2) διαφορετικά θέματα, ενώ ήταν διπλάσιες οι περιπτώσεις εμφάνισης των συνεχόμενων θετικών έναντι των αρνητικών αξιολογήσεων (;;), που αφορούσαν ένα θέμα. Φαίνεται πως αν η πρώτη προτροπή από τον προγραμματιστή δεν οδηγήσει ορθά το μοντέλο, τότε δύσκολα βρίσκεται τη λύση με τις επόμενες προτροπές, καθώς διατηρείται η αρχική κατεύθυνση που έλαβε.



Σχήμα 2.33: Απόδοση μοντέλων ανά τύπο θέματος



Σχήμα 2.34: Μέσος όρος αξιολόγησης απαντήσεων για θέματα, μέρος 2



Σχήμα 2.35: Μέσος όρος αξιολόγησης απαντήσεων για θέματα, μέρος 3

Από τα σχήματα ::, :: και ::, παρατηρείται ότι δεν υπάρχει κάποια συγκεκριμένη τάση στην αξιολόγηση των απαντήσεων ανά θέμα, με τα περίσσοτερα των θεμάτων να έχουν θετικό μέσο όρο. Αξιοσημείωτη η περίπτωση του θέματος της ανάπτυξης της λειτουργίας του προγραμματιστή του παιχνιδιού (Developer Router), όπου στις πρώτες προτροπές, δεν δόθηκε στο μοντέλο κάποια κατεύθυνση, με αποτέλεσμα οι απαντήσεις που παρήχθηκαν να αξιολογήθουν αρνητικά, ενώ στις επόμενες προτροπές, οι απαντήσεις που παρήχθηκαν αξιολογήθηκαν κατά μέσο όρο ίσο με μηδέν (0), παρατηρώντας την μεγαλύτερη αύξηση στο μέσο όρο αξιολογήσεων από θέμα σε θέμα.

Το θέμα Manual Intervention αφορούσε τις περιπτώσεις που ο προγραμματιστής αναγκάστηκε να επενέβει, με σκοπό να αποφευχθούν συνεχόμενες αρνητικές αξιολογήσεις στο ίδιο θέμα μετά από μια σειρά λάθος απαντήσεων.

2.5 Συμπεράσματα

Όντας μια καινοτόμος επιλογή τεχνολογικής στοίβας, η χρήση της Typescript, σε συνδυασμό με το tRPC και το Prisma, αποτέλεσε μια πρόκληση για το μοντέλο του GitHub Copilot. Η επιλογή αυτή έγινε με σκοπό την ορθότερη αξιολόγηση της ικανότητας του μοντέλου να ακολουθήσει την ταχεία εξέλιξη των τεχνολογιών και συγκεκριμένα στο οικοσύστημα της Javascript, καθώς έχει συνεχείς προσθήκες και αλλαγές στον τρόπο που γράφεται ο κώδικας και επίσης αποτελεί τη βασική προγραμματιστική γλώσσα του διαδικτύου.

Εκ του αποτελέσματος των δοκιμών, η ιδανικότερη χρήση του μοντέλου του GitHub Copilot φαίνεται είναι να ακολουθήσει τις επιλογές του προγραμματιστή για την ανάπτυξη του κώδικα, την αρχιτεκνική, τα πρότυπα σχεδίασης και τις βιβλιοθήκες που θα

χρησιμοποιηθούν κατά την ανάπτυξη της εφαρμογής, όπως και αποδείχθηκε από τα αποτελέσματα στις περιπτώσεις που δόθηκε η ευκαιρία στο μοντέλο να λάβει αποφάσεις, καθώς οι περισσότερες αλλαγές που πραγματοποιήθηκαν από την πλευρά του προγραμματιστή ήταν σε μεγάλες εκτάσεις κώδικα και συγκεκριμένα στις περιπτώσεις των ελέγχων, όπου εκεί το μοντέλο έλαβε πρωτοβουλίες κατά τη σύνταξη των ελέγχων, οδηγώντας σε λάθη.

Κεφάλαιο 3

Μοντέλο πρόβλεψης απόδοσης απαντήσεων μέσω προτροπής

Στο Κεφάλαιο 2 παρουσιάστηκε η μεθοδολογία που ακολουθήθηκε για την συλλογή των δεδομένων και την αξιολόγηση των απαντήσεων του μοντέλου του GitHub Copilot. Η κατηγοριοποίηση των δεδομένων, οδήγησε στην ιδέα να αναπτυχθεί ένα μοντέλο μηχανικής μάθησης με σκοπό την πρόβλεψη της απόδοσης μιας απάντησης του μοντέλου του GitHub Copilot, με βάση την προτροπή.

Με βάση την προτροπή, αρχικά έγινε η προσπάθεια για την πρόβλεψη της ακριβούς τιμής της αξιολόγησης, από πολύ αρνητική έως πολύ θετική, αλλά, λόγω του μικρού μεγέθους των δεδομένων, η παραπάνω λογική οδήγησε σε εξαιρετικά ασυνεπή αποτέλεσματα, με ακριβεία χαμηλότερη αυτής των πενήντα τοις εκατό (50%). Συνεπώς, αποφασίστηκε να γίνει η προσπάθεια κατηγοριοποίησης της απάντησης είτε σε θετική αξιολόγηση, είτε σε αρνητική.

3.1 Κατηγοριοποίηση των δεδομένων

Για την κατηγοριοποίηση των δεδομένων, η κάθε προτροπή επεξεργάστηκε, προκειμένου να αφαιρεθούν τα κομμάτια κώδικα από την υπόλοιπη προτροπή και έπειτα αφαιρέθηκαν οι "κενές λέξεις" (stopwords) με την χρήση της βιβλιοθήκης Natural Language Toolkit (NLTK). Για κάθε προτροπή που είχε κομμάτια κώδικα, ο αριθμός των συμβόλων (symbols), αποθηκεύθηκε σε μια άλλη κατηγορία.

Ένα από τα μεγαλύτερα προβλήματα των Γλωσσικών Μοντέλων, είναι ο μέγιστος αριθμός των συμβόλων (tokens) που μπορούν να επεξεργαστούν. Καθώς οι προτροπές ζητούσαν μεγάλα κώδικα, ειδικά στις περιπτώσεις των ελέγχων του κώδικα, το μοντέλο συχνά έφτανε στο όριο των δεδομένων που μπορούσε να παράξει στην απάντησή του. Καθώς το μοντέλο δεν γνώριζε εκ των προτέρων το μέγεθος της απάντησης, αν ο προγραμματιστής δεν το παρότρυνε να σταματήσει σε ένα συγκεκριμένο αριθμό συμβόλων, το μοντέλο θα συνέχιζε να παράγει κώδικα μέχρι να φτάσει στο όριο των δεδομένων που μπορούσε να παράξει, οδηγώντας σε μια σειρά από αποτυχημένες απαντήσεις.

Για την αντιμετώπιση του προβλήματος, ο προγραμματιστής ζήτησε από το μοντέλο να απαντάει για ένα συγκεκριμένο μέρος του προβλήματος, όπως παραδείγματος χάρη έναν συγκεκριμένο έλεγχο, αντί όλων των ελέγχων. Έπειτα, ζήτησε από το μοντέλο να

περιμένει την προτροπή με περιεχόμενο την λέξη "continue" (συνέχισε), προκειμένου να συνεχίσει με την προηγούμενη απάντησή του. Με αυτόν τον τρόπο, το μοντέλο θα μπορούσε να απαντήσει σε μια προτροπή, χωρίς να φτάσει στο όριο των δεδομένων που μπορούσε να παράξει.

Ωστόσο, αυτό οδήγησε σε αρκετές προτροπές με μοναδικό περιεχόμενο την λέξη "continue", που αφενός δεν είχαν καμία σχέση με το προηγούμενο κείμενο και αφετέρου, εφ' όσον ο προγραμματιστής παρότρυνε το μοντέλο να συνεχίσει με την απάντησή του, σήμαινε πως η τουλάχιστον αμέσως προηγούμενη απάντηση ήταν ικανοποιητική, οδηγώντας σε μια μεροληψία προς θετικά αποτελέσματα. Συνεπώς, αποφασίστηκε να υπολογίζεται ο συνεχόμενος αριθμός προτροπών με περιεχόμενο την λέξη "continue", συνάμα με το θέμα της προτροπής, ώστε να δημιουργούνται ομάδες προτροπών που στοχεύουν στο ίδιο ερώτημα, αλλά χρειάστηκε να διασπαστούν λόγω των περιορισμών του μοντέλου.

Περιπτώσεις προτροπών που είτε ματαιώθηκαν, είτε δεν είχαν απάντηση λόγω τεχνικών προβλημάτων, ήταν σημειωμένες εξαρχής και παραλείφθηκαν από την ανάλυση.

3.1.1 Προεπεξεργασία και Προετοιμασία Δεδομένων

Με την χρήση των βιβλιοθηκών pandas και scikit-learn, η διαδικασία προεπεξεργασίας και προετοιμασίας των δεδομένων περιελάμβανε τα ακόλουθα στάδια:

- Εισαγωγή Δεδομένων:** Τα δεδομένα εισήχθησαν από ένα αρχείο CSV χρησιμοποιώντας τη βιβλιοθήκη pandas για τον αποτελεσματικό χειρισμό τους.
- Κωδικοποίηση Κατηγορικών Μεταβλητών:** Οι κατηγορικές μεταβλητές "subject" (θέμα) και "type" (τύπος) κωδικοποιήθηκαν αριθμητικά με τη χρήση του LabelEncoder.
- Μετατροπή Χρονικών Δεδομένων:** Η στήλη "createdAt" μετατράπηκε σε αριθμητική μορφή, αντιπροσωπεύοντας τα δευτερόλεπτα από την εποχή UNIX (1/1/1970).
- Δυαδικοποίηση Αξιολογήσεων:** Η στήλη "rating" (αξιολόγηση) μετατράπηκε σε δυαδική μορφή, όπου οι θετικές αξιολογήσεις αντιστοιχήθηκαν στην τιμή ένα (1) και οι αρνητικές στην τιμή μηδέν (0).
- Ομαδοποίηση Δεδομένων:** Πραγματοποιήθηκε ομαδοποίηση των δεδομένων με βάση τη στήλη "group". Για κάθε ομάδα, υπολογίστηκαν διάφορα στατιστικά μέτρα, όπως ο μέσος όρος για τις αριθμητικές μεταβλητές και η επικρατούσα τιμή για τις κατηγορικές.
- Επεξεργασία Κειμένου:** Εφαρμόστηκε η τεχνική Term Frequency-Inverse Document Frequency (TF-IDF) για τη μετατροπή των κειμενικών προτροπών σε αριθμητικά χαρακτηριστικά. Η στήλη "prompt" (προτροπή) μετασχηματίστηκε σε πολλαπλές στήλες, κάθε μία εκ των οποίων αντιπροσωπεύει ένα διακριτό χαρακτηριστικό TF-IDF.
- Διαχείριση Ελλιπών Τιμών:** Οι ελλιπείς τιμές στις αριθμητικές στήλες αντικαταστάθηκαν με τη μέση τιμή της αντίστοιχης στήλης, διασφαλίζοντας την πληρότητα του συνόλου δεδομένων.

3.1.2 Επιλογή Χαρακτηριστικών

Η διαδικασία επιλογής χαρακτηριστικών περιελάμβανε τα εξής βήματα:

1. **Εφαρμογή SelectKBest:** Χρησιμοποιήθηκε η μέθοδος SelectKBest σε συνδυασμό με τη συνάρτηση (χ^2) για την επιλογή των δέκα (10) πιο σημαντικών χαρακτηριστικών. Αυτή η προσέγγιση συνέβαλε στη μείωση της διαστατικότητας του προβλήματος και στην επικέντρωση στα πλέον πληροφοριακά χαρακτηριστικά.
2. **Διαχωρισμός Δεδομένων:** Τα δεδομένα διαχωρίστηκαν σε σύνολα εκπαίδευσης και ελέγχου, με αναλογία ογδόντα τοις εκατό (80%) και είκοσι τοις εκατό (20%) αντίστοιχα, χρησιμοποιώντας τη συνάρτηση train_test_split.

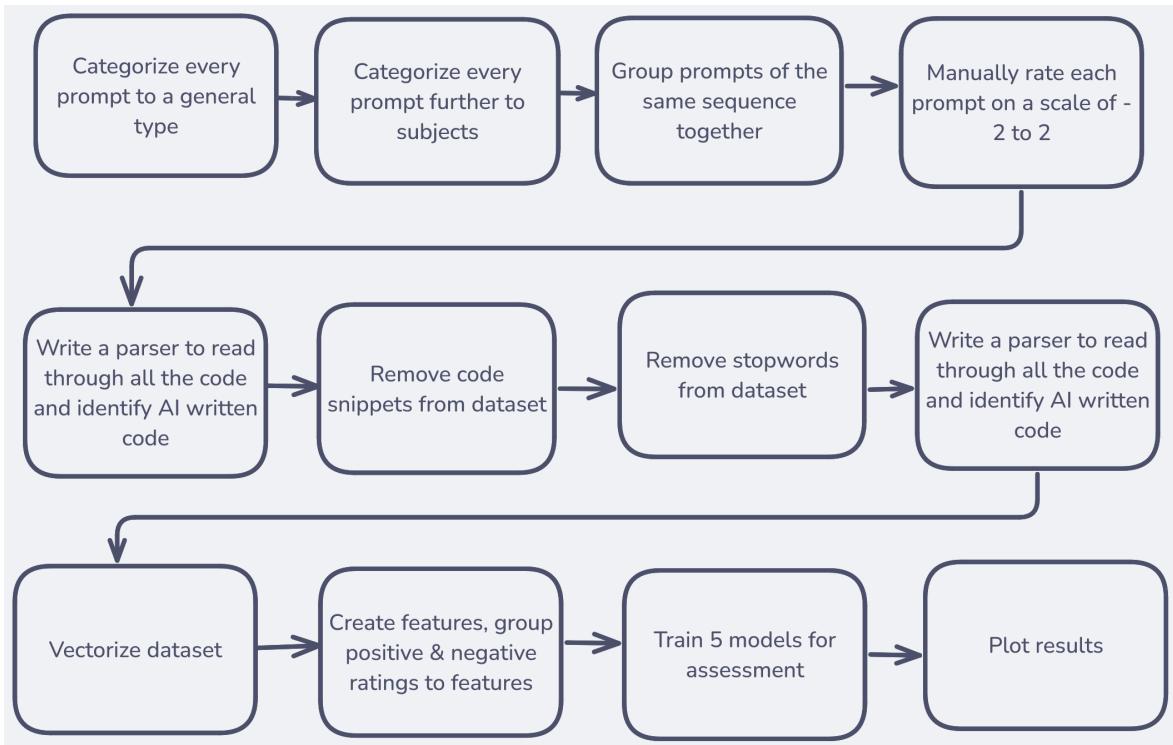
3.2 Εκπαίδευση και Αξιολόγηση Μοντέλων

Για την εκπαίδευση και αξιολόγηση των μοντέλων ακολουθήθηκε η εξής διαδικασία:

1. **Επιλογή Αλγορίθμων:** Επιλέχθηκαν πέντε διαφορετικοί αλγόριθμοι μηχανικής μάθησης: Random Forest, Support Vector Machine (SVM), Gradient Boosting, MetaCost, και Naive Bayes.
2. **Βελτιστοποίηση Υπερπαραμέτρων:** Για κάθε αλγόριθμο, πραγματοποιήθηκε εκτενής αναζήτηση πλέγματος (Grid Search) με δεκάπτυχη διασταυρωμένη επικύρωση για τη βελτιστοποίηση των υπερπαραμέτρων.
3. **Εκπαίδευση Μοντέλων:** Τα μοντέλα εκπαιδεύτηκαν χρησιμοποιώντας τις βέλτιστες υπερπαραμέτρους που προέκυψαν από την αναζήτηση πλέγματος.
4. **Αξιολόγηση Μοντέλων:** Η απόδοση των μοντέλων αξιολογήθηκε χρησιμοποιώντας το σύνολο ελέγχου. Υπολογίστηκαν μετρικές όπως η ακρίβεια, η ανάκληση, και το F1-score.
5. **Οπτικοποίηση Αποτελεσμάτων:** Δημιουργήθηκαν πίνακες σύγχυσης (confusion matrices) και συγκριτικά γραφήματα για την οπτική αναπαράσταση της απόδοσης των διαφόρων μοντέλων.

Η μεθοδολογία αυτή επέτρεψε τη συστηματική ανάπτυξη, εκπαίδευση και αξιολόγηση των μοντέλων, παρέχοντας μια ολοκληρωμένη εικόνα της απόδοσής τους στην ταξινόμηση των δεδομένων.

Πραγματοποιήθηκε επίσης έρευνα για την επιλογή ανάπτυξης ενός νευρωνικού δικτύου, αλλά λόγω του μικρού μεγέθους των δεδομένων, το αποτέλεσμα ήταν ανεπαρκή, με ακρίβεια κάτω του πενήντα τοις εκατό (50%). Με την εφαρμογή των παραπάνω βημάτων, η ακρίβεια των μοντέλων αυξήθηκε σημαντικά, με τα αποτελέσματα να αναλύονται παρακάτω.



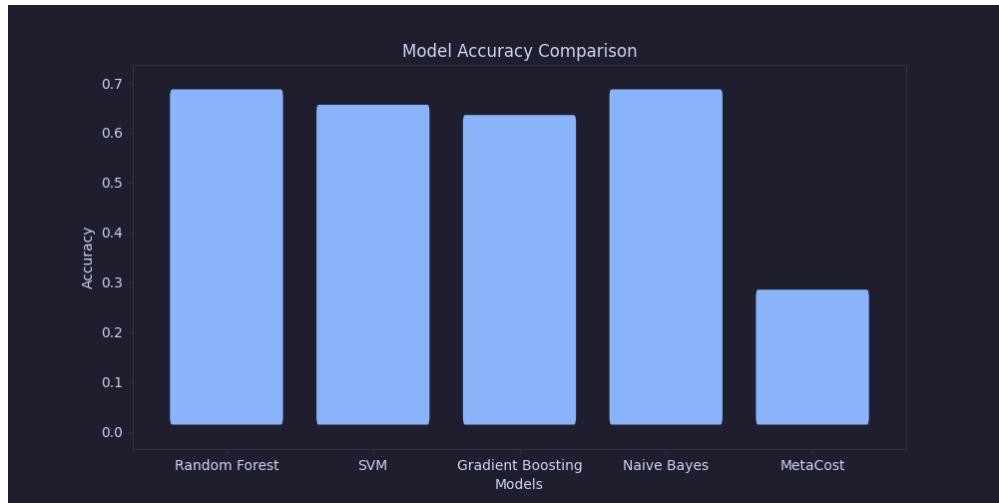
Σχήμα 3.1: Διάγραμμα Ανάπτυξης Μοντέλου

3.3 Αποτελέσματα

Η ανάπτυξη ενός μοντέλου πρόβλεψης της απόδοσης Γλωσσικών Μοντέλων μέσω πολλαπλών κριτηρίων, με ιδιαίτερη έμφαση στην προτροπή, αποτελεί ένα καινοτόμο ερευνητικό εγχείρημα. Ο όρος “μηχανική προτροπής” (Prompt Engineering) αναφέρεται στη εξειδικευμένη μεθοδολογία σχεδιασμού και βελτιστοποίησης προτροπών με στόχο τη μεγιστοποίηση της απόδοσης του μοντέλου [105, 106].

Οι επιστημονικές έρευνες στον ραγδαία εξελισσόμενο τομέα αυτό είναι ιδιαίτερα πρόσφατες, με την πλειονότητα των σημαντικών δημοσιεύσεων να εμφανίζεται μετά τη δημόσια κυκλοφορία του ChatGPT [107, 108]. Ωστόσο, η αποτελεσματική εφαρμογή και βελτιστοποίηση της μηχανικής προτροπής αποτελεί ένα από τα πιο κρίσιμα και πολύπλοκα προβλήματα ενός κόσμου ολοένα και περισσότερο ενισχυμένου από την τεχνητή νοημοσύνη.

Η εξαγωγή του μέγιστου δυνατού επιπέδου απόδοσης από τα Γλωσσικά Μοντέλα, και κατ' επέκταση η μείωση οφαλμάτων και μη ικανοποιητικών αποκρίσεων, με αποτέλεσμα την αποτελεσματική αύξηση της απόδοσης χωρίς την εισαγωγή επιπρόσθετων δεδομένων ή την τροποποίηση του υπάρχοντος κώδικα ανάπτυξης του μοντέλου, αποτελεί μια διαδικασία τόσο δύσκολη όσο και δυσνόητη. Η αξιοποίηση ενός μοντέλου μηχανικής μάθησης για την πρόβλεψη της απόδοσης ενός γλωσσικού μοντέλου μέσω της προτροπής δύναται να παράσχει πολύτιμα εργαλεία για τη μεγιστοποίηση της απόδοσης.

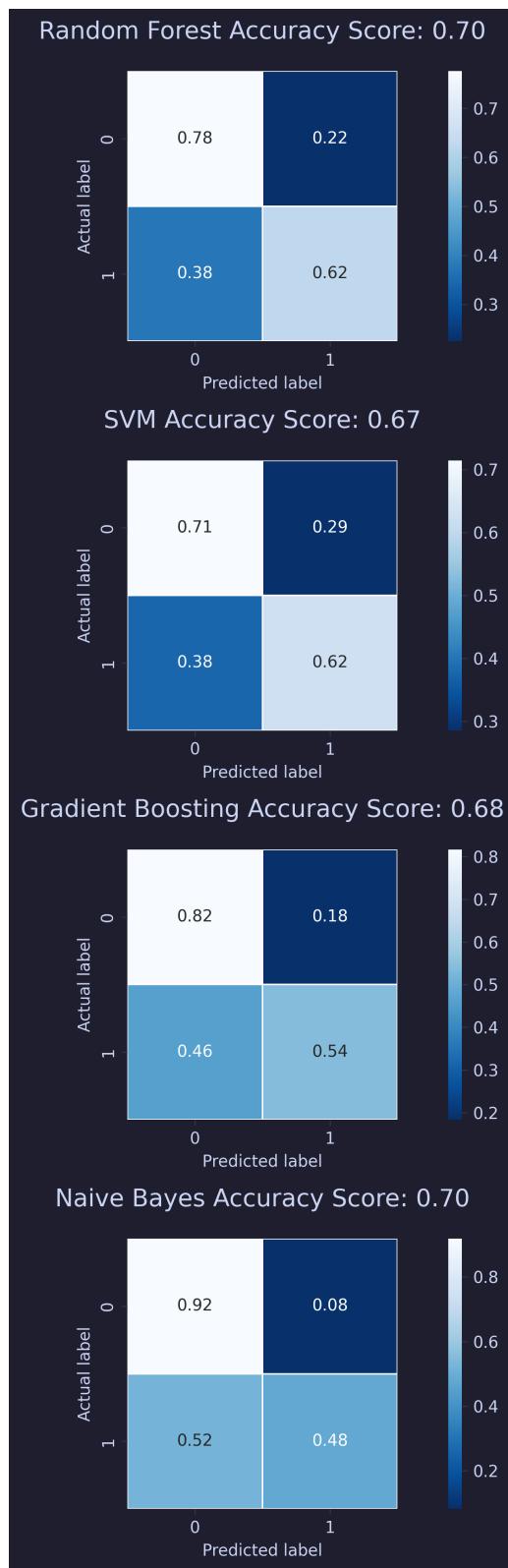


Σχήμα 3.2: Σύγκριση ακρίβειας μοντέλων

Το παραπάνω σχήμα παρουσιάζει την ακρίβεια των μοντέλων που αναπτύχθηκαν. Συγκεκριμένα:

1. Random Forest (Ακρίβεια: 70,10%):
2. SVM (Ακρίβεια: 67,01%):
3. Gradient Boosting (Ακρίβεια: 68,04%):
4. Naive Bayes (Ακρίβεια: 70,10%):
5. MetaCost (Ακρίβεια: 29.89%)

Το μοντέλο Random Forest μαζί με το Naive Bayes αποδείχθηκαν τα πιο αποτελεσματικά μοντέλα στην πρόβλεψη των απαντήσεων, με το μοντέλο MetaCost να έχει την χειρότερη απόδοση με διαφορά. Για μια πιο στοχευμένη ανάλυση των αποτελεσμάτων, παρατίθενται οι πίνακες σύγχυσης των μοντέλων:



Σχήμα 3.3: Πίνακες Σύγχυσης μοντέλων

Για κάθε αλγόριθμο, ο πίνακας σύγχυσης δείχνει την απόδοση στην πρόβλεψη δύο κλάσεων, 0 και 1, αναπαραστώντας τις αρνητικές και θετικές αξιολογήσεις αντίστοιχα. Αναλύοντας τους πίνακες:

1. Random Forest:

- Σωστές προβλέψεις για την κλάση 0: 78%
- Λανθασμένες προβλέψεις για την κλάση 0: 22%
- Σωστές προβλέψεις για την κλάση 1: 62%
- Λανθασμένες προβλέψεις για την κλάση 1: 38%

2. SVM:

- Σωστές προβλέψεις για την κλάση 0: 71%
- Λανθασμένες προβλέψεις για την κλάση 0: 29%
- Σωστές προβλέψεις για την κλάση 1: 62%
- Λανθασμένες προβλέψεις για την κλάση 1: 38%

3. Gradient Boosting:

- Σωστές προβλέψεις για την κλάση 0: 82%
- Λανθασμένες προβλέψεις για την κλάση 0: 18%
- Σωστές προβλέψεις για την κλάση 1: 54%
- Λανθασμένες προβλέψεις για την κλάση 1: 46%

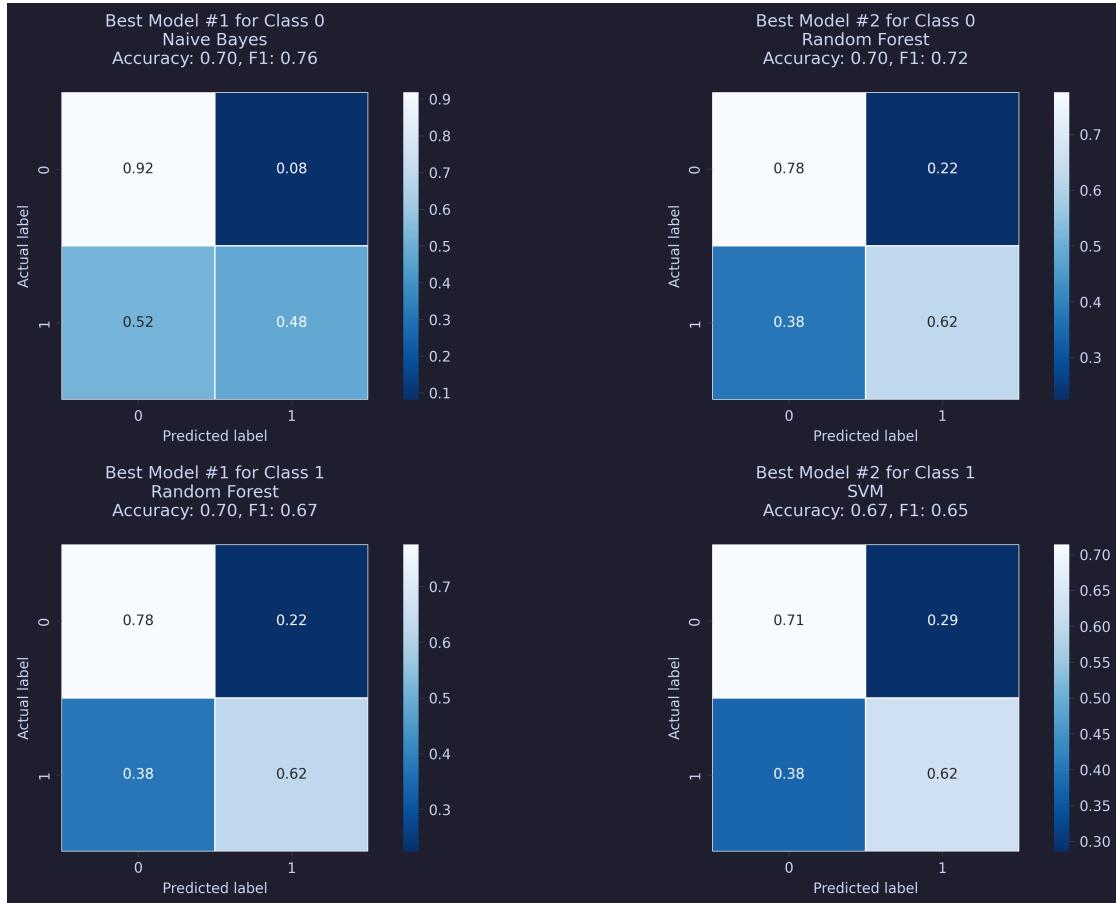
4. Naive Bayes:

- Σωστές προβλέψεις για την κλάση 0: 92%
- Λανθασμένες προβλέψεις για την κλάση 0: 8%
- Σωστές προβλέψεις για την κλάση 1: 48%
- Λανθασμένες προβλέψεις για την κλάση 1: 52%

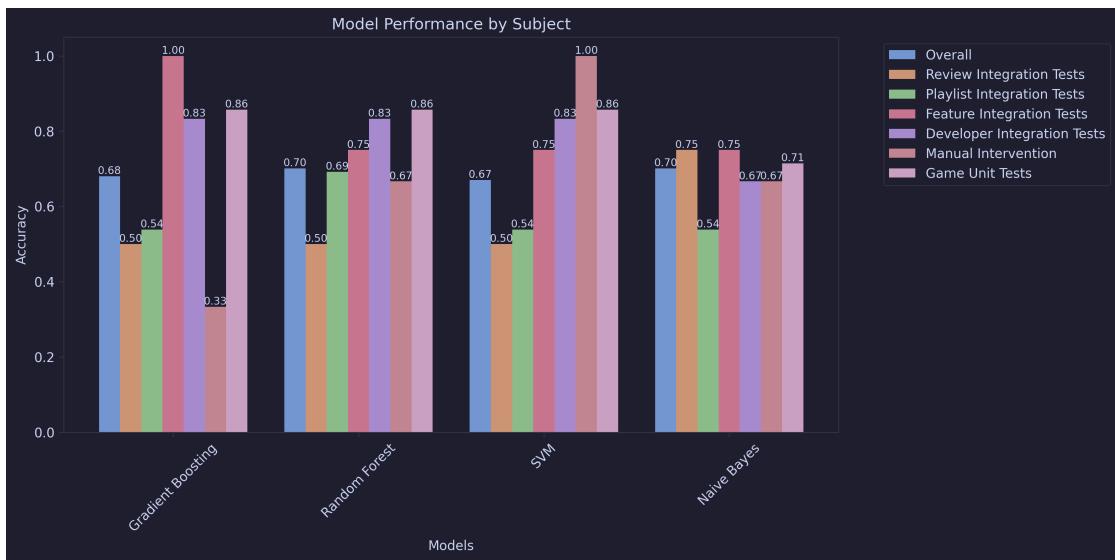
Παρατηρείται ότι ο Naive Bayes έχει την υψηλότερη ακρίβεια στην πρόβλεψη της κλάσης 0, αλλά τη χαμηλότερη για την κλάση 1. Ο Random Forest και το SVM παρουσιάζουν πιο ισορροπημένη απόδοση μεταξύ των δύο κλάσεων. Το Gradient Boosting έχει καλή απόδοση στην κλάση 0, αλλά υστερεί ελαφρώς στην κλάση 1 σε σύγκριση με τους άλλους αλγορίθμους.



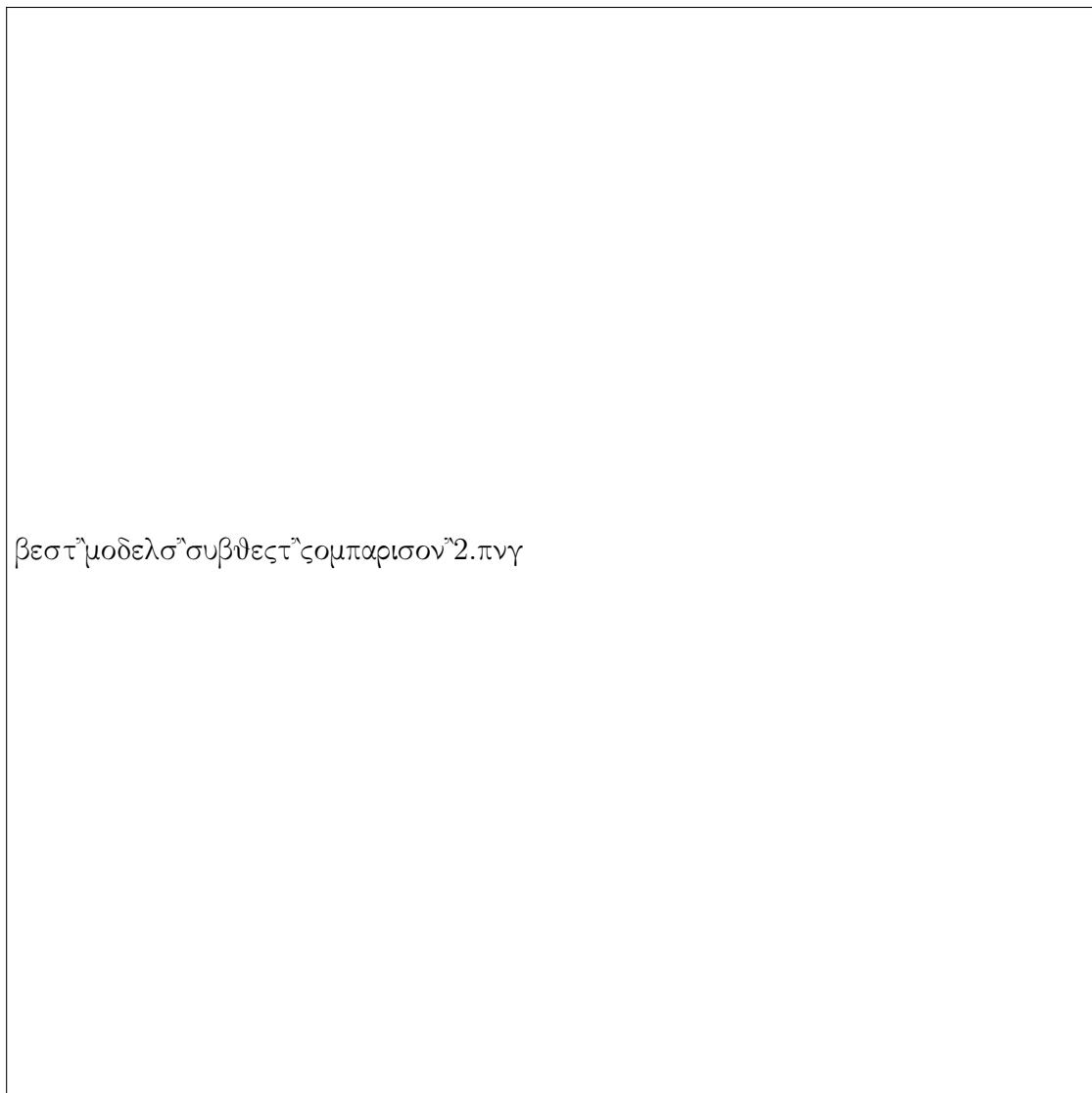
Σχήμα 3.4: Απόδοση μοντέλων ανά τύπο θέματος



Σχήμα 3.5: Απόδοση μοντέλων ανά κλάση



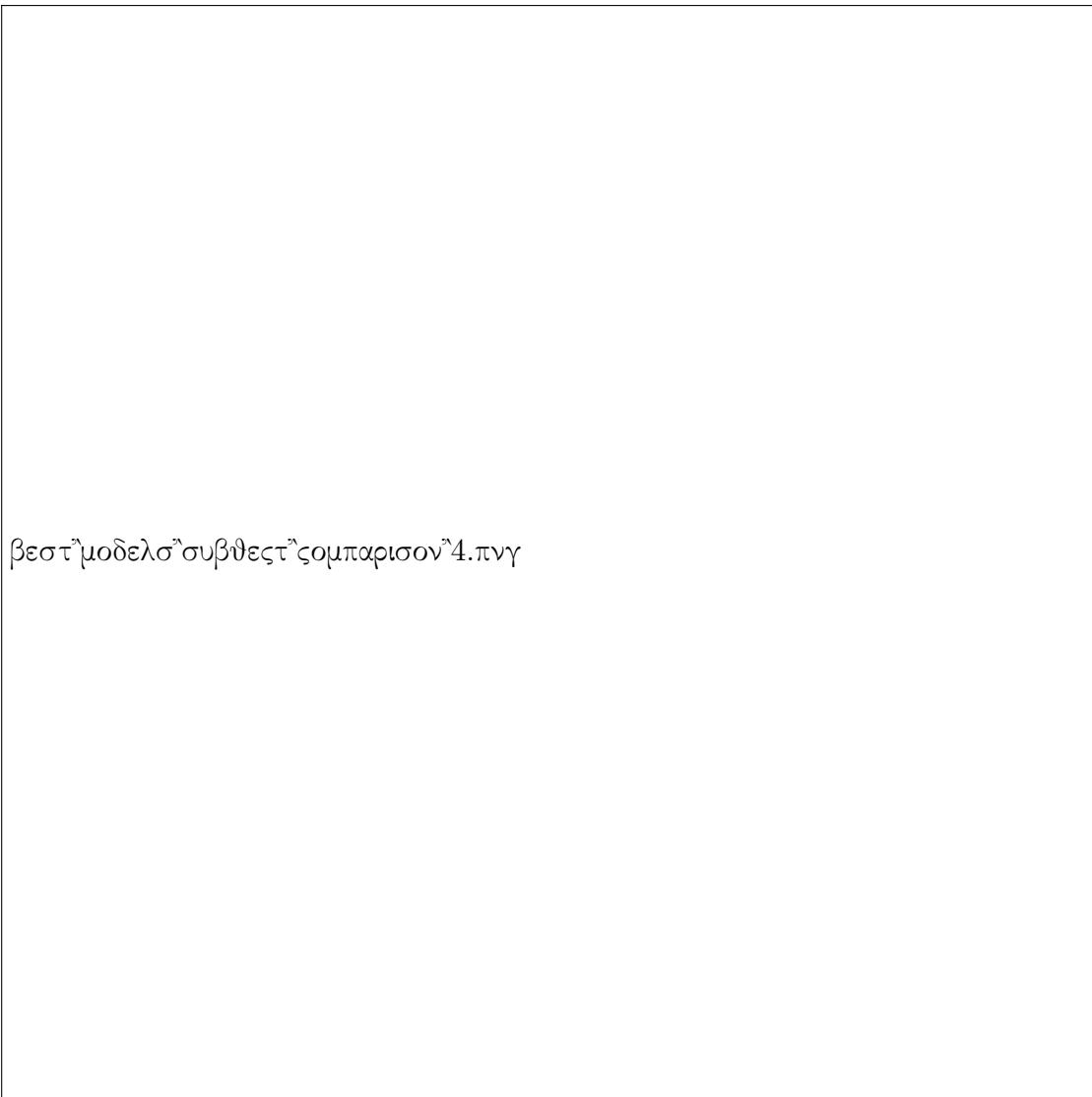
Σχήμα 3.6: Απόδοση μοντέλων ανά θέμα, μέρος 1



Σχήμα 3.7: Απόδοση μοντέλων ανά θέμα, μέρος 2

βεστ”μοδελσ”συβθεςτ”ζομπαρισον”3.πνγ

Σχήμα 3.8: Απόδοση μοντέλων ανά θέμα, μέρος 3



Σχήμα 3.9: Απόδοση μοντέλων ανά θέμα, μέρος 4

Το ποσοστό απόδοσης θα μπορούσε να αυξηθεί περαιτέρω με την εφαρμογή περίσσοτερων τεχνικών προεπεξεργασίας, όπως η εξισορρόπηση των κλάσεων, η επιλογή χαρακτηριστικών, και η βελτιστοποίηση των υπερπαραμέτρων. Επιπλέον, η επέκταση του συνόλου δεδομένων με περισσότερες αξιολογήσεις θα μπορούσε να βελτιώσει την απόδοση των μοντέλων.

Το βασικό πρόβλημα για την βελτιστοποίηση μέσω επέκτασης του συνόλου δεδομένων αποτελεί το ηθικό δίλημμα της συλλογής δεδομένων. Οι εταιρείες υπέυθυνες για την ανάπτυξη των Γλωσσικών Μοντέλων, έχουν την δυνατότητα συλλογής δεδομένων των χρηστών τους. Στην περίπτωση του GitHub Copilot, η συλλογή δεδομένων του χρήστη είναι ενεργή από προεπιλογή και ο χρήστης μπορεί να ελέγξει συγκεκριμένες μόνο παραμέτρους προκειμένου να προστατέψει τα δεδομένα του. [109]. Το ίδιο ισχύει και για την περίπτωση του ChatGPT [110].

Τα δεδομένα χρηστών είναι ένα από τα πολυτιμότερα ψηφιακά αγαθά για τις εταιρείες που αναπτύσσουν Γλωσσικά Μοντέλα, καθώς αποτελούν την πηγή της βελτιστοποίησης των μοντέλων τους. Παρόμοιο ζήτημα αποτελεί η συλλογή πηγαίου κώδικα για την

ανάπτυξη των μοντέλων από πλατφόρμες ελέγχου εκδόσεων. Συγκεκριμένα για την περίπτωση του GitHub, μιας πλατφόρμας που ως επί το πλείστον ειδικεύεται στην ανάπτυξη μιας υποδομής ασφαλούς αποθήκευσης κώδικα, όταν το μοντέλο του GitHub Copilot εκπαιδεύεται, οι περισσότερες άδειες ανοιχτού λογισμικού [111, 112, 113], δεν είχαν κατευθυντήριες γραμμές σε ό,τι αφορούσε την χρήση του κώδικα για την εκπαίδευση μοντέλου.

Η άδεια GNU General Public License (GPL) για παράδειμα, θεωρείται μια άδεια "-copyleft", που σημαίνει ότι απαιτεί τα παράγωγα έργα να διανέμονται υπό τους ίδιους όρους άδειας, το οποίο σημαίνει ότι ο κώδικας παραμένει ελεύθερος και ανοιχτός. Η GPL λειτουργεί ως εξής: 'Όταν χρησιμοποιείται κώδικας με άδεια GPL σε ένα έργο, ολόκληρο το έργο πρέπει να διανεμηθεί υπό την ίδια άδεια. Αυτό διασφαλίζει ότι ο κώδικας και οποιεσδήποτε τροποποιήσεις του παραμένουν ανοιχτές και προσβάσιμες στην κοινότητα. Οι χρήστες έχουν το δικαίωμα να τρέχουν, να μελετούν, να μοιράζονται και να τροποποιούν το λογισμικό. Εάν διανεμηθούν τροποποιημένες εκδόσεις, πρέπει να είναι διαθέσιμες υπό την ίδια άδεια, διατηρώντας έτσι την ελευθερία του λογισμικού. Επομένως, βάσει ορισμού, κώδικας αδειοδοτημένος από την GPL, δεν θα μπορούσε να χρησιμοποιηθεί κατά την ανάπτυξη ενός Γλωσσικού Μοντέλου κλειστού κώδικα, κάτι που στην περίπτωση του GitHub Copilot, δεν ισχύει, όπως αποδείχθηκε από την έρευνα που σύνταξε η ομάδα του ανταγωνιστικού προϊόντος Γλωσσικού Μοντέλου Codeium [114, 28]. Κατά την διάρκεια της έρευνας αυτής, το μοντέλο του GitHub Copilot φάνηκε να αναγνωρίζει και να ολοκληρώνει μέρη από αρχεία των διάφορων προγραμμάτων που φέρουν την GNU, όπως το FFmpeg, χρησιμοποιώντας τα σχόλια σαν αρχή της προτοπής, με το μοντέλο να ακολουθεί πιστά τον πηγαίο κώδικα.

Στα πλαίσια της τεχνητής νοημοσύνης, δεν υπάρχει ακόμα συγκεκριμένος ορισμός για την ανοιχτή και ελεύθερη τεχνητή νοημοσύνη, καθώς ακόμα ο οργανισμός Open Source Initiative, δεν έχει ακόμα εκδόσει επίσημο ορισμό για την ανοιχτή τεχνητή νοημοσύνη, αφού βρίσκεται ακόμα σε καθεστώς υποψήφιας έκδοσης του ορισμού (Release Candidate, RC) [115]. Ο οργανισμός Open Source Initiative είναι υπεύθυνος για την προώθηση, την εκπαίδευση και την υπεράσπιση των αρχών του ανοιχτού λογισμικού, τη διατήρηση του Ορισμού Ανοιχτού Κώδικα, και τη δημιουργία γεφυρών μεταξύ διαφορετικών κοινοτήτων στο οικοσύστημα του ανοιχτού κώδικα εν γένει.

Στην περίπτωση της τεχνητής νοημοσύνης, ο όρος Ανοιχτός Κώδικας δεν αφορά αποκλειστικά τον πηγαίο κωδικά, αλλά και τα δεδομένα που χρησιμοποιήθηκαν για την ανάπτυξη ενός μοντέλου. Παρεθένοντας τον ορισμό της ανοικτής τεχνητής νοημοσύνης από την υποψήφια έκδοση του Open Source Initiative, ως εξής:

Συγκεκριμένα, αυτό [οι πληροφορίες για τα δεδομένα] πρέπει να περιλαμβάνει: (1) μια λεπτομερή περιγραφή όλων των δεδομένων που χρησιμοποιούνται για την εκπαίδευση, συμπεριλαμβανομένων (εάν χρησιμοποιούνται) των μη κοινοποιήσιμων δεδομένων, αποκαλύπτοντας την προέλευση των δεδομένων, το εύρος και τα χαρακτηριστικά τους, πώς αποκτήθηκαν και επιλέχθηκαν τα δεδομένα, τις διαδικασίες επισήμανσης και τις μεθοδολογίες καθαρισμού δεδομένων· (2) μια λίστα όλων των δημόσια διαθέσιμων δεδομένων εκπαίδευσης και πού να τα αποκτήσετε· και (3) μια λίστα όλων των δεδομένων εκπαίδευσης που μπορούν να αποκτηθούν από τρίτους και πού να τα αποκτήσετε, συμπεριλαμβανομένων αυτών με χρέωση. '

Ιδιαίτερο ενδιαφέρον παρουσιάζει η περίπτωση του μοντέλου Llama της εταιρείας Meta (παλαιότερα Facebook)[116]. Το συγκεκριμένο μοντέλο, παρουσιάζεται από την ε-

ταιρεία Meta ως ένα μοντέλο ανοιχτού κώδικα, καθώς ο πηγαίος κώδικας είναι δημόσιος στην πλατφόρμα του GitHub. Ωστόσο, τόσο η συλλογή δεδομένων, όσο και η επιλογή της άδειας του κώδικα αποτελούν πηγή συζητήσεων σχετικά με την εγκυρότητά τους. Αρχικά, η πηγή από την οποία συλλέχθηκαν τα δεδομένα, δεν είναι γνωστή. [117] Η άδεια του λογισμικού όμως, είναι το μεγαλύτερο ζήτημα. Το Llama 3.2, είναι αδειοδοτημένο από την ομώνυμη άδεια, η οποία δεν είναι επίσημα υιοθετημένη από το Open Source Initiative. Αυτό οφείλεται σε ορισμένους κανόνες χρήσης του λογισμικού [118]:

- ‘Δεν θα χρησιμοποιήσετε τα Υλικά Llama ή οποιαδήποτε έξοδο ή αποτελέσματα των Υλικών Llama για τη βελτίωση οποιουδήποτε άλλου μεγάλου γλωσσικού μοντέλου (εξαιρουμένου του Llama 2 ή παράγωγων έργων αυτού).’ (Το Llama 3.2 υπόκειται στα παράγωγα έργα του Llama 2)
- ‘Πρόσθετοι Εμπορικοί Όροι. Εάν, κατά την ημερομηνία κυκλοφορίας της έκδοσης Llama 2, οι μηνιαίοι ενεργοί χρήστες των προϊόντων ή υπηρεσιών που διατίθενται από ή για τον Αδειούχο, ή τις συνδεδεμένες εταιρείες του Αδειούχου, υπερβαίνουν τους 700 εκατομμύρια μηνιαίους ενεργούς χρήστες κατά τον προηγούμενο ημερολογιακό μήνα, πρέπει να ζητήσετε άδεια από τη Meta, την οποία η Meta μπορεί να σας παραχωρήσει κατά την απόλυτη διακριτική της ευχέρεια, και δεν έχετε εξουσιοδότηση να ασκήσετε οποιοδήποτε από τα δικαιώματα βάσει αυτής της Συμφωνίας εκτός εάν ή μέχρις ότου η Meta σας παραχωρήσει ρητά τέτοια δικαιώματα.’

Οι παραπάνω όροι έρχονται σε έντονη αντίθεση με τον ορισμό του λογισμικού ανοιχτού κώδικα, περιορίζοντας τη χρήση του κώδικα με τρόπους που δεν διευκολύνουν την ανοιχτή και ελεύθερη διανομή λογισμικού.

Η εκπαίδευση ενός μοντέλου μηχανικής μάθησης για την πρόβλεψη της απάντησης ενός Γλωσσικού Μοντέλου μέσω της προτροπής, πρέπει να αποκτήσει τα απαραίτητα δεδομένα μέσω απόφασης χρηστών, το οποίο ξεπερνά το πλαίσιο αυτής της διατριβής.

Παράτημα Α'

Ακρωνύμια και συντομογραφίες

LLM Large Language Model

NLP Natural Language Processing

IDE Integrated Development Environment

LSP Language Server Protocol

API Application Programming Interface

ORM Object Relational Mapping

CRUD Create Read Update Delete

BDD Behavior Driven Development

NLTK Natural Language Toolkit

TF-IDF Frequency-Inverse Document Frequency

SVM Support Vector Machine

GPL GNU General Public License

Bibliography

- [1] Github copilot.
- [2] The VS Code Team. Visual studio code 1.0! *Visual Studio Code Blog*, April 2016. Retrieved June 1, 2024.
- [3] Visual studio code.
- [4] Roberto Rodriguez-Echeverria, Javier Luis Canovas Izquierdo, Manuel Wimmer, and Jordi Cabot. Towards a language server protocol infrastructure for graphical modeling. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '18, pages 370–380, New York, NY, USA, 2018. Association for Computing Machinery.
- [5] Staff. 2024 stack overflow developer survey. *Stack Overflow Blog*, May 2024. Retrieved Jul 29, 2024.
- [6] Benfeng Xu, An Yang, Junyang Lin, Quan Wang, Chang Zhou, Yongdong Zhang, and Zhendong Mao. Expertprompting: Instructing large language models to be distinguished experts, 2023.
- [7] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models, 2021.
- [8] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models, 2023.
- [9] Qian Zhou, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419*, 2023.
- [10] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson, 2009.
- [11] OpenAI. Chatgpt: Optimizing language models for dialogue, 2022.
- [12] K Staff. Microsoft-backed openai valued at \$80bn after company completes deal. *The Guardian*, Feb 2024. Retrieved May 30, 2024.
- [13] Cade Metz and Tripp Mickle. Openai completes deal that values the company at \$80 billion. *The New York Times*, Feb 2024. Retrieved May 30, 2024.

- [14] Emma Roth. Microsoft spent hundreds of millions of dollars on a chatgpt supercomputer. *The Verge*, Mar 2023. Archived from the original on May 30, 2023. Retrieved March 30, 2023.
- [15] Zhiye Liu. Chatgpt will command more than 30,000 nvidia gpus: Report. *Tom's Hardware*, Mar 2023. Archived from the original on May 30, 2024. Retrieved November 2, 2023.
- [16] Trendforce says with cloud companies initiating ai arms race, gpu demand from chatgpt could reach 30,000 chips as it readies for commercialization. *TrendForce*, Nov 2023. Archived from the original on May 30, 2024. Retrieved November 2, 2023.
- [17] S Adams, I Arel, J Bach, R Coop, R Furlan, B Goertzel, J S Hall, A Samsonovich, M Scheutz, M Schlesinger, et al. Mapping the landscape of human-level artificial general intelligence. *AI Magazine*, 33(1):25–42, 2012.
- [18] Ben Goertzel. Artificial general intelligence: Concept, state of the art, and future prospects. *Journal of Artificial General Intelligence*, 5(1):1–46, 2014. Submitted 2013-2-12, Accepted 2014-3-15.
- [19] GitHub. Introducing github copilot: your ai pair programmer. *GitHub Blog*, June 2021. Retrieved May 30, 2024.
- [20] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Heben Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.
- [21] Jonathan Kemper. Openai kills its codex code model, recommends gpt3.5 instead, March 2023. Retrieved May 27, 2024.
- [22] Shuyin ZhaoShuyin Zhao. Smarter, more efficient coding: Github copilot goes beyond codex with improved ai model. *GitHub Blog*, July 2023. Retrieved May 27, 2024.
- [23] Thomas Dohmke. Github copilot x: The ai-powered developer experience. *GitHub Blog*, March 2023. Retrieved May 30, 2024.
- [24] Staff. Github copilot – november 30th update. *GitHub Blog*, November 2023. Retrieved May 30, 2024.
- [25] Shuyin Zhao. Github copilot chat now generally available for organizations and individuals. *GitHub Blog*, December 2023. Retrieved May 30, 2024.

- [26] Unknown. Tabnine: Coding in vs code with the help of an ai assistant. learn.microsoft.com, March 2021. Retrieved April 19, 2024.
- [27] James Vincent. This ai-powered autocomplete software is gmail’s smart compose for coders. *The Verge*, July 2019. Retrieved May 19, 2024.
- [28] Sandy Carter. Exponential baby! navigating the ai convergence of tech with nvidia. *Forbes Digital Assets*, May 2024. Retrieved May 30, 2024.
- [29] Jeremy Bays. Aws announces amazon codewhisperer (preview), Jun 2022.
- [30] Github student developer pack.
- [31] Chu Luo. A report on automatic code completion, 03 2017.
- [32] H. H. Koester and S. Levine. Effect of a word prediction feature on user performance. *Augmentative and alternative communication*, 12(3):155–168, 1996.
- [33] M. Asaduzzaman, C. K. Roy, K. A. Schneider, and D. Hou. Cscc: Simple, efficient, context sensitive code completion. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 71–80. IEEE, 2014.
- [34] A. Svyatkovskoy, S. Lee, A. Hadjitofi, M. Riechert, J. Franco, and M. Allamanis. Fast and memory-efficient neural code completion. *arXiv preprint arXiv:2004.13651*, 2020.
- [35] V. Raychev, M. Vechev, and E. Yahav. Code completion with statistical language models. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 419–428. ACM, 2014.
- [36] C. Omar, Y.S. Yoon, T.D. LaToza, and B.A. Myers. Active code completion. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 859–869. IEEE, 2012.
- [37] Jonas Kjær Rask, Frederik Palludan Madsen, Nick Battle, Hugo Daniel Macedo, and Peter Gorm Larsen. The specification language server protocol: A proposal for standardised lsp extensions. 2022.
- [38] Hendrik Bunder. Decoupling language and editor - the impact of the language server protocol on textual domain-specific languages. In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development - MODELSWARD*, pages 131–142. INSTICC, SciTePress, 2019.
- [39] Maliheh Izadi, Roberta Gismondi, and Georgios Gousios. Codefill: Multi-token code completion by jointly learning from structure and naming sequences. In *Proceedings of the 44th International Conference on Software Engineering, ICSE ’22*, pages 401–412, New York, NY, USA, 2022. Association for Computing Machinery.
- [40] Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. Code prediction by feeding trees to transformers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 150–162. IEEE, 2021.
- [41] Yanlin Wang and Hui Li. Code completion by modeling flattened abstract syntax trees as graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 14015–14023, 2021.

- [42] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Dixin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online, Nov 2020. Association for Computational Linguistics.
- [43] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. An empirical study on the usage of bert models for code completion. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 108–119, 2021.
- [44] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Antonio Mastropaoletti, Emad Aghajani, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. An empirical study on the usage of transformer models for code completion. *IEEE Transactions on Software Engineering*, 2021.
- [45] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland, May 2022. Association for Computational Linguistics.
- [46] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1433–1443, 2020.
- [47] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Dixin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.
- [48] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations*, 2023.
- [49] Maliheh Izadi, Jonathan Katzy, Tim van Dam, Marc Otten, Razvan Mihai Popescu, and Arie van Deursen. Language models for code completion: A practical evaluation. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*, pages 1–13, New York, NY, USA, April 14–20 2024. ACM.
- [50] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.
- [51] Typescript.

- [52] Netscape and Sun. Netscape and sun announce javascript, the open, cross-platform object scripting language for enterprise networks and the internet. Press release, 12 1995. Retrieved June 01 2024.
- [53] TIOBE Software BV. Tiobe index for javascript. <https://www.tiobe.com/tiobe-index/javascript/>. Retrieved June 01 2024.
- [54] Staff. The top programming languages, 2022. Retrieved May 30, 2024.
- [55] Staff. 2022 stack overflow developer survey. *Stack Overflow Blog*, June 2022. Retrieved Jul 29, 2024.
- [56] Staff. 2023 stack overflow developer survey. *Stack Overflow Blog*, June 2023. Retrieved Jul 29, 2024.
- [57] V8 javascript engine.
- [58] Chromium.
- [59] Spidermonkey.
- [60] Gecko.
- [61] Javascriptcore.
- [62] Webkit.
- [63] trpc.
- [64] Node.js.
- [65] Mysql.
- [66] Prisma.
- [67] Nextauth.
- [68] React.
- [69] Next.js.
- [70] Jest.
- [71] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*, volume 1. Addison-Wesley, 1999.
- [72] J. Irena. *Software Testing Methods and Techniques*. 2008.
- [73] *Guide to the Software Engineering Body of Knowledge*. 2004. A project of the IEEE Computer Society Professional Practices Committee.
- [74] E. F. Miller. Introduction to software testing technology. In *Software Testing & Validation Techniques*, pages 4–16. IEEE, 1981.
- [75] M. Shaw. Prospects for an engineering discipline of software. *IEEE Software*, pages 15–24, Nov 1990.
- [76] create-t3-app. <https://github.com/t3-oss/create-t3-app>, 2022. Commit: f69fa29733d3bd0d6e5043d1200e250a938b633.

- [77] Github copilot chat in visual studio code. <https://github.com/microsoft/vscode-copilot-release>, 2023.
- [78] Cyclopt. Cyclopt: Automated software quality assurance, 2024.
- [79] R. D. Bunker, S. M. Datar, C. F. Kemerer, and D. Zweig. Software complexity and maintenance costs. *Communications of the ACM*, 36(11):81–94, 1993.
- [80] R. P. L. Buse and W. R. Weimer. Learning a metric for code readability. In *IEEE Transactions on Software Engineering*, volume 36, pages 546–558, 2010.
- [81] G. McGraw. Software security: Building security in. Technical report, Addison-Wesley Professional, 2006.
- [82] M. H. Halstead. *Elements of Software Science*. Elsevier North-Holland, Inc., New York, NY, USA, 1977.
- [83] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976.
- [84] Abid Jamil, Muhammad Arif, Normi Abubakar, and Akhlaq Ahmad. Software testing techniques: A literature review. pages 177–182, 11 2016.
- [85] Satish Kathiriya, Rajath Karangara, and Narayana Challa. Optimizing automated software testing with machine learning techniques. *International Journal of Science and Research (IJSR)*, 7:2319–7064, 03 2018.
- [86] Ron Patton. *Software Testing*. Sams Publishing, Indianapolis, IN, 2005.
- [87] Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. Sotopia: Interactive evaluation for social intelligence in language agents, 2024.
- [88] Arek Ait Baha, Mohamed El Hajji, Youssef Es-saady, and Hammou Fadili. The power of personalization: A systematic review of personality-adaptive chatbots. *SN Computer Science*, 4, 2023.
- [89] Thorsten Ball. *Writing An Interpreter In Go*. Thorsten Ball, 2018.
- [90] Faker.js.
- [91] Lorem ipsum - all the facts - lipsum generator. <https://www.lipsum.com/>. Retrieved June 11, 2024.
- [92] Prisma. Many-to-many relations. <https://www.prisma.io/docs/concepts/components/prisma-schema/relations/many-to-many-relations>, 2022. Retrieved 2024-06-11.
- [93] Alban Gabillon, Manuel Munier, Jean Jacques Bascou, Laurent Gallon, and Emmanuel Bruno. An access control model for tree data structures. In *Information Security, 5th International Conference, ISC 2002 Sao Paulo, Brazil, September 30 - October 2, 2002, Proceedings*, pages 117–135, Sao Paulo, Brazil, 2002.
- [94] Atlassian. Join relationships and joining tables. <https://www.atlassian.com/data/sql/joins>. Retrieved June 11, 2024.

- [95] Douglas Crockford. *JavaScript: The Good Parts*, pages 36–40. Yahoo Press. O'Reilly Media, Sebastopol, CA, 1 edition, 2008.
- [96] Carlos Solis Pineda and Xiaofeng Wang. A study of the characteristics of behaviour driven development. pages 383 – 387, 10 2011.
- [97] A. H. Mughal. Advancing bdd software testing: Dynamic scenario re-usability and step auto-complete for cucumber framework. 2024.
- [98] Shoaib Farooq, Uzma Omer, Amna Ramzan, Mansoor Rasheed, and Zabihullah Atal. Behavior driven development: A systematic literature review. *IEEE Access*, PP:1–1, 01 2023.
- [99] Staff. Using copilot chat in vs code. *Visual Studio Code Documentation*, March 2024. Retrieved June 20, 2024.
- [100] Steve Freeman and Nat Pryce. *Growing Object-Oriented Software, Guided by Tests*. Addison-Wesley Professional, Upper Saddle River, NJ, 2009.
- [101] Owain Parry, Gregory M. Kapfhammer, Michael Hilton, and Phil McMinn. A survey of flaky tests. *Transactions on Software Engineering and Methodology*, 31(1), 2022.
- [102] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.
- [103] George J Klir and Bo Yuan. *Fuzzy sets and fuzzy logic: theory and applications*. Prentice Hall, 1995.
- [104] Timothy J Ross. *Fuzzy logic with engineering applications*. John Wiley & Sons, 2010.
- [105] Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. How Can We Know What Language Models Know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 07 2020.
- [106] Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. Unleashing the potential of prompt engineering in large language models: a comprehensive review, 2024.
- [107] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection, 2023.
- [108] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt, 2023.
- [109] GitHub Ryan Salva, VP of Product. How github copilot handles data. Retrieved Aug 10, 2024.
- [110] OpenAI. Europe privacy policy, February 2024. Retrieved Aug 10, 2024.
- [111] Free Sofware Foundation. Free software definition. Retrieved Aug 10, 2024.
- [112] The Open Source Initiative. Free software definition, February 2024. Retrieved Aug 10, 2024.

- [113] Software in the Public Interest. Debian social contract, October 2022. Retrieved Aug 10, 2024.
- [114] Codeium Team. Github copilot emits gpl. codeium does not. *Codeium Blog*, April 2023. Retrieved Aug 30, 2024.
- [115] Open Source Initiative. The open source ai definition 1.0-rc. <https://opensource.org/deepdive/drafts/the-open-source-ai-definition-1-0-rc1>, October 2024. Version 1.0, Release Candidate 1 (Draft).
- [116] Llama 3.2 documentation. https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_2, 2024.
- [117] Introducing meta llama 3: The most capable openly available llm to date. <https://ai.meta.com/blog/meta-llama-3/>, 2024.
- [118] Meta Llama. License. <https://github.com/meta-llama/llama/blob/main/LICENSE>. Commit: 7c0a08ec24ce91880184ba9002227624262904a2.