ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

A coin with a head on it

Description automatically generated

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΑΞΙΟΠΟΙΗΣΗ ΜΕΓΑΛΩΝ ΓΛΩΣΣΙΚΩΝ ΜΟΝΤΕΛΩΝ ΓΙΑ ΤΗΝ ΑΝΑΠΤΥΞΗ ΑΥΤΟΜΑΤΩΝ ΕΛΕΓΧΩΝ ΑΠΟΔΟΧΗΣ ΛΟΓΙΣΜΙΚΟΥ»

ΠΛΑΤΙΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: P3200157

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:  
ΔΙΑΜΑΝΤΙΔΗΣ ΝΙΚΟΛΑΟΣ

ΑΘΗΝΑ, ΑΥΓΟΥΣΤΟΣ 2024

Πινακασ περιεχομενων

[ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ 1](#_Toc174459456)

[ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ 1](#_Toc174459457)

[ΠΕΡΙΛΗΨΗ 2](#_Toc174459458)

[1. ΕΙΣΑΓΩΓΗ 3](#_Toc174459459)

[1.1 Μεγάλα γλωσσικά μοντέλα, BDD και αυτοματοποίηση κώδικα 3](#_Toc174459460)

[1.2 Σκοπός της παρούσας διπλωματικής εργασίας 4](#_Toc174459461)

[1.3 Δομή 4](#_Toc174459462)

[2. ΜΕΓΑΛΑ ΓΛΩΣΣΙΚΑ ΜΟΝΤΕΛΑ 6](#_Toc174459463)

[2.1 Τι είναι ένα νευρωνικό δίκτυο και πώς λειτουργεί 6](#_Toc174459464)

[2.1.1 Ορισμός νευρωνικών δικτύων 6](#_Toc174459465)

[2.1.2 Ορισμός της μείωσης βαρών με καθοδική κλίση 7](#_Toc174459466)

[2.1.3 Ορισμός του αλγορίθμου ανάστροφης μετάδοσης 9](#_Toc174459467)

[2.2 Τι είναι ένα μεγάλο γλωσσικό μοντέλο 10](#_Toc174459468)

[2.2.1 Ορισμός μεγάλου γλωσσικού μοντέλου 10](#_Toc174459469)

[2.2.2 Τι κάνει ένα Transformer 10](#_Toc174459470)

[2.2.3 Τι είναι το Attention Layer 11](#_Toc174459471)

[2.2.4 Τι είναι το Feed Forward Step 12](#_Toc174459472)

[2.3 Ιστορία και εξέλιξη των μεγάλων γλωσσικών μοντέλων 12](#_Toc174459473)

[2.3.1 Το chatbot ELIZA 12](#_Toc174459474)

[2.3.2 Άνοδος των νευρωνικών δικτύων 13](#_Toc174459475)

[2.3.3 Δημιουργία των LSTM 14](#_Toc174459476)

[2.3.4 Δημιουργία Gated Recurrent Network 14](#_Toc174459477)

[2.3.5 Άνοδος του συστατικού Attention 14](#_Toc174459478)

[2.3.6 Η εφεύρεση των Transformers 15](#_Toc174459479)

[2.3.7 Εμφάνιση μεγάλων γλωσσικών μοντέλων 15](#_Toc174459480)

[2.4 Κύριες εφαρμογές και χρήσεις 16](#_Toc174459481)

[2.4.1 Ανάλυση ήχου 16](#_Toc174459482)

[2.4.2 Δημιουργία περιεχομένου 16](#_Toc174459483)

[2.4.3 Υποστήριξη πελατών 16](#_Toc174459484)

[2.4.4 Μετάφραση γλωσσών 17](#_Toc174459485)

[2.4.5 Εκπαίδευση 17](#_Toc174459486)

[2.4.6 Κυβερνοασφάλεια 17](#_Toc174459487)

[2.5 Παραδείγματα Μεγάλων Γλωσσικών Μοντέλων 17](#_Toc174459488)

[2.5.1 BERT 17](#_Toc174459489)

[2.5.2 GEMINI 18](#_Toc174459490)

[2.5.3 GPT –3 18](#_Toc174459491)

[2.5.4 GPT -3.5 και GPT –3.5 Turbo 18](#_Toc174459492)

[2.5.5 GPT –4 19](#_Toc174459493)

[2.5.6 GPT –4o 19](#_Toc174459494)

[3. ΜΕΓΑΛΑ ΓΛΩΣΣΙΚΑ ΜΟΝΤΕΛΑ ΣΤΗΝ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ 19](#_Toc174459495)

[3.1 Ρόλος των μεγάλων γλωσσικών μοντέλων στην ανάπτυξη λογισμικού 19](#_Toc174459496)

[3.1.1 Εισαγωγή στην έννοια των γλωσσικών μοντέλων στην ανάπτυξη λογισμικού 19](#_Toc174459497)

[3.1.2 Παραδείγματα χρήσης μεγάλων γλωσσικών μοντέλων στην ανάπτυξη λογισμικού 20](#_Toc174459498)

[3.1.3 Αυτοματοποίηση κώδικα 20](#_Toc174459499)

[3.1.4 Ανάλυση και διόρθωση λαθών 21](#_Toc174459500)

[3.1.5 Ανάλυση των απαιτήσεων μετατροπή τους σε τεχνικές προδιαγραφές 21](#_Toc174459501)

[3.2 Χρήσεις σε διάφορα στάδια της ανάπτυξης λογισμικού 21](#_Toc174459502)

[3.2.1 Στάδιο ανάλυσης απαιτήσεων 21](#_Toc174459503)

[3.2.2 Στάδιο σχεδιασμού 22](#_Toc174459504)

[3.2.3 Στάδιο κωδικοποίησης 22](#_Toc174459505)

[3.2.4 Στάδιο ελέγχου και βελτιστοποίησης 23](#_Toc174459506)

[3.3 Πλεονεκτήματα και περιορισμοί 24](#_Toc174459507)

[3.3.1 Πλεονεκτήματα 24](#_Toc174459508)

[3.3.2 Περιορισμοί 25](#_Toc174459509)

[3.4 Μελλοντικές προοπτικές και ευκαιρίες ανάπτυξης 25](#_Toc174459510)

[3.4.1 Σύμπραξη μεγάλων γλωσσικών μοντέλων 25](#_Toc174459511)

[3.4.2 Κατανόηση διαφορετικού τύπου εισόδων 26](#_Toc174459512)

[3.4.3 Προσαρμογή στις ανάγκες των χρηστών 26](#_Toc174459513)

[3.4.4 Βελτίωση στις ήδη υπάρχουσες εργασίες 26](#_Toc174459514)

[4. BEHAVIOR-DRIVEN DEVELOPMENT (BDD) 26](#_Toc174459515)

[4.1 Test Driven Development και Behavior Driven Development 27](#_Toc174459516)

[4.2 Η σημασία της συνεργασίας στο BDD 29](#_Toc174459517)

[4.2.1 Ομοιογενής γλώσσα (Ubiquitous Language) 29](#_Toc174459518)

[4.2.2 Σενάρια και ιστορίες χρηστών στο BDD 29](#_Toc174459519)

[4.3 Οι τρείς πρακτικές του BDD 30](#_Toc174459520)

[4.4 Τεχνικές Ανάλυσης Συνεργασίας 31](#_Toc174459521)

[4.5 Αυτοματισμός Δοκιμών με το Cucumber 32](#_Toc174459522)

[4.5.1 Εισαγωγή στο Cucumber 32](#_Toc174459523)

[4.5.2 Η σύνταξη Gherkin 33](#_Toc174459524)

[4.5.3 Δημιουργία αρχείων χαρακτηριστικών (Feature Files) 34](#_Toc174459525)

[4.5.4 Ανάπτυξη των Step Definitions 35](#_Toc174459526)

[4.6 Αναφορές του Cucumber 36](#_Toc174459527)

[4.7 Ζωντανή τεκμηρίωση 37](#_Toc174459528)

[4.8 Καλές πρακτικές για γραφή με Cucumber 38](#_Toc174459529)

[5. ΧΡΗΣΗ ΤΩΝ ΜΕΓΑΛΩΝ ΓΛΩΣΣΙΚΩΝ ΜΟΝΤΕΛΩΝ ΓΙΑ BDD 38](#_Toc174459530)

[5.1 Μεθοδολογία 38](#_Toc174459531)

[5.1.1 Το πρόβλημα της βιβλιοθήκης 39](#_Toc174459532)

[5.1.2 Μετατροπή των use cases σε σενάρια Cucumber 39](#_Toc174459533)

[5.1.3 Ανάπτυξη BDD tests 42](#_Toc174459534)

[5.1.4 Εκτέλεση και επαλήθευση των BDD Tests 45](#_Toc174459535)

[5.1.5 Επιλογή συγκεκριμένων μεγάλων γλωσσικών μοντέλων 45](#_Toc174459536)

[5.1.6 Σχεδιασμός διαφορετικών συνομιλιών βάσει προϋπάρχουσας γνώσης 46](#_Toc174459537)

[5.2 Κριτήρια αξιολόγησης 48](#_Toc174459538)

[5.3 Εμπειρική αξιολόγηση 54](#_Toc174459539)

[5.3.1 Αξιολόγηση της φάσης 1 54](#_Toc174459540)

[5.3.2 Αξιολόγηση της φάσης 2 57](#_Toc174459541)

[5.3.3 Αξιολόγηση της φάσης 3 58](#_Toc174459542)

[5.3.4 Αξιολόγηση της φάσης 4 60](#_Toc174459543)

[6. ΣΥΜΠΕΡΑΣΜΑΤΑ 63](#_Toc174459544)

[7. ΒΙΒΛΙΟΓΡΑΦΙΑ 65](#_Toc174459545)

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

[Εικόνα 1. Παράδειγμα απλού νευρωνικού δικτύου 4](#_Toc174098167)

[Εικόνα 2. Παράδειγμα αναπαράστασης του gradient descent 5](#_Toc174098168)

[Εικόνα 3. Παράδειγμα μαθηματικής αναπαράστασης του ανάδελτα 6](#_Toc174098169)

[Εικόνα 4. Παράδειγμα επικοινωνίας των transformers σε ένα μεγάλο γλωσσικό μοντέλο 8](#_Toc174098170)

[Εικόνα 5. Ένα παράδειγμα συζήτησης με το chatbot ELIZA 10](#_Toc174098171)

[Εικόνα 6. Σύγκριση της απόδοσης Attention και RNN 12](#_Toc174098172)

[Εικόνα 7. Ο κύκλος ανάπτυξης του TDD 25](#_Toc174098173)

[Εικόνα 8. Παράδειγμα ιστορίας χρήστη 27](#_Toc174098174)

[Εικόνα 9. Οι τρείς πρακτικές του BDD 28](#_Toc174098175)

[Εικόνα 10. The Three Amigos Meeting 29](#_Toc174098176)

[Εικόνα 11. Παράδειγμα σεναρίου με την μορφή Given-When-Then 31](#_Toc174098177)

[Εικόνα 12. Παράδειγμα σεναρίου με την μορφή Given-When-Then 31](#_Toc174098178)

[Εικόνα 13. Παράδειγμα δομής ενός feature file 32](#_Toc174098179)

[Εικόνα 14. Παράδειγμα δομής ενός αρχείου Step Definitions 33](#_Toc174098180)

[Εικόνα 15. Παράδειγμα αναφοράς με το Cucumber 34](#_Toc174098181)

[Εικόνα 16. Παράδειγμα του use case δανεισμού αντιτύπων του συστήματος της βιβλιοθήκης 38](#_Toc174098182)

[Εικόνα 17. Παράδειγμα σεναρίων Cucumber για το use case του δανεισμού αντιτύπων 39](#_Toc174098183)

[Εικόνα 18. Παράδειγμα κώδικα σε Java για τον έλεγχο των σεναρίων 41](#_Toc174098184)

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

[Πίνακας 1. Κριτήρια αξιολόγησης των μεγάλων γλωσσικών μοντέλων 46](#_Toc174440746)

[Πίνακας 2. Φάση 1 αξιολόγησης των μεγάλων γλωσσικών μοντέλων 52](#_Toc174440747)

[Πίνακας 3. Φάση 2 αξιολόγησης των μεγάλων γλωσσικών μοντέλων 55](#_Toc174440748)

[Πίνακας 4. Φάση 3 αξιολόγησης των μεγάλων γλωσσικών μοντέλων 57](#_Toc174440749)

[Πίνακας 5. Φάση 4 αξιολόγησης των μεγάλων γλωσσικών μοντέλων 59](#_Toc174440750)

ΠΕΡΙΛΗΨΗ

# ΕΙΣΑΓΩΓΗ

## Μεγάλα γλωσσικά μοντέλα, BDD και αυτοματοποίηση κώδικα

Στην σύγχρονη εποχή της τεχνολογίας, η τεχνητή νοημοσύνη και τα μεγάλα γλωσσικά μοντέλα, όπως το GPT-3.5, το GPT-4 και το GPT-4o της OpenAI, έχουν γίνει πολύ σημαντικά εργαλεία σε πολλούς τομείς της καθημερινότητας και της τεχνολογίας της επιστήμης υπολογιστών. Συγκεκριμένα, τα μεγάλα γλωσσικά μοντέλα έχουν αλλάξει ριζικά τον τρόπο που οι άνθρωποι χειρίζονται την ανάπτυξη λογισμικού, αφού έχουν φέρει την επανάσταση στην επεξεργασία της φυσικής γλώσσας, προσφέροντας δυνατότητες που πριν από λίγα χρόνια θεωρούνταν αδιανόητες. Αυτά τα μοντέλα, μέσω της εκπαίδευσής τους σε τεράστιους όγκους δεδομένων, έχουν αναπτύξει ικανότητες κατανόησης της φυσικής γλώσσας σε επίπεδο που δεν υπάρχει πια διαφοροποίηση από τους ανθρώπους και την ανθρώπινη επικοινωνία.

Η προοπτική που παρουσιάζουν τα μεγάλα γλωσσικά μοντέλα αυτά στην βελτίωση της διαδικασίας της ανάπτυξης λογισμικού παρουσιάζει ιδιαίτερο ενδιαφέρον. Ειδικότερα, η χρήση αυτών των μοντέλων για την δημιουργία αυτοματοποιημένου κώδικα σε σενάρια Behavior-Driven Development έχει προκαλέσει ιδιαίτερο ενδιαφέρον, καθώς μπορεί να μειώσει δραστικά τον χρόνο ανάπτυξης και να βελτιώσει σε μεγάλο βαθμό την ποιότητα του κώδικα. Το BDD (Behavior-Driven Development) αποτελεί μία μεθοδολογία ανάπτυξης λογισμικού που εστιάζει στην κατανόηση και την επαλήθευση της συμπεριφοράς του συστήματος από την πλευρά του χρήστη, απαιτώντας σαφή και κατανοητή περιγραφή των απαιτήσεων και των λειτουργιών του συστήματος, ενώ παράλληλα προωθεί την συνεργασία μεταξύ των διαφορετικών μερών ενός συστήματος λογισμικού.

Παρά τη σημαντική πρόοδο που έχουν επιτύχει τα μεγάλα γλωσσικά μοντέλα, η εφαρμογή τους σε συγκεκριμένες μεθοδολογίες ανάπτυξης, όπως το BDD, ενδέχεται να αντιμετωπίζει προκλήσεις. Τα μοντέλα αυτά πρέπει να είναι σε θέση να κατανοούν και να παράγουν ακριβή και λειτουργικό κώδικα που ανταποκρίνεται στις απαιτήσεις των BDD, συμπεριλαμβανομένων των Step Definitions και των απαιτήσεων του BDD. Ειδικότερα, οι προκλήσεις περιλαμβάνουν τη σωστή κατανόηση των απαιτήσεων του χρήστη και τη δημιουργία κώδικα που να είναι σύμφωνος με τα σενάρια BDD που έχουν δοθεί.

## Σκοπός της παρούσας διπλωματικής εργασίας

Η παρούσα διπλωματική εργασία ασχολείται με το πρόβλημα της αξιολόγησης της απόδοσης των μεγάλων γλωσσικών μοντέλων στην παραγωγή αυτοματοποιημένου κώδικα για σενάρια BDD, έχοντας όμως κάθε φορά ένα διαφορετικό μέγεθος προϋπάρχουσας γνώσης δοσμένο στο σύστημα. Το μέγεθος της προϋπάρχουσας αυτής γνώσης χωρίστηκε σε τέσσερις φάσεις, με κάθε μία να προσθέτει επιπλέον πληροφορίες από την προηγούμενη. Επίσης, η εργασία αυτή έχει και ως σκοπό την εύρεση μίας βέλτιστης μεθοδολογίας για την επικοινωνία με τα μεγάλα γλωσσικά μοντέλα και της παρουσίασης της γνώσης σε αυτά με τέτοιο τρόπο, ώστε να παράγονται τα βέλτιστα αποτελέσματα στον λιγότερο χρόνο.

## Δομή

Αρχικά, παρουσιάζεται μια ανάλυση της έννοιας των μεγάλων γλωσσικών μοντέλων, περιγράφοντας τις αρχές τους, την αρχιτεκτονική τους, τον τρόπο με τον οποίο λειτουργούν αλλά και την ιστορία τους μέχρι το σήμερα. Στην συνέχεια, εξετάζεται η τεχνολογία πίσω από την λειτουργία τους, όπως η εκπαίδευσή τους και οι αλγόριθμοι που χρησιμοποιούνται πίσω από αυτά τα πολύ δημοφιλή μοντέλα.

Επιπρόσθετα, παρουσιάζεται η χρήση των μεγάλων γλωσσικών μοντέλων στον τομέα της ανάπτυξης λογισμικού, όπως η χρήση τους για την δημιουργία αυτοματοποιημένου κώδικα, την βελτίωση της ποιότητας του κώδικα και την ενίσχυση της παραγωγικότητας των προγραμματιστών. Εξετάζονται παραδείγματα εφαρμογών τους, τα πλεονεκτήματα , οι προκλήσεις αλλά και οι μελλοντικές προοπτικές τους για βελτίωση.

Συνεχίζοντας, γίνεται μία αναλυτική περιγραφή της έννοιας της μεθοδολογίας Behavior-Driven Development, εξηγώντας τον βασικό σκοπό της, ποια είναι τα χαρακτηριστικά της, οι αρχές της και πως εφαρμόζεται στον τομέα της ανάπτυξης λογισμικού. Ακόμη, παρουσιάζεται ο τρόπος χρήσης του εργαλείου Cucumber σε συνδυασμό με το BDD, ο τρόπος δημιουργίας αυτοματοποιημένων τεστ με το εργαλείο αυτό αλλά και όλες οι απαιτούμενες θεωρητικές γνώσεις που απαιτούνται για την υιοθέτηση του εργαλείου αυτού, ακολουθούμενα από πραγματικά παραδείγματα.

Τέλος, παρουσιάζεται η χρήση των μεγάλων γλωσσικών μοντέλων για την υποστήριξη του BDD.

Συγκεκριμένα, παρουσιάζεται η μεθοδολογία που ακολουθήθηκε κατά την διεξαγωγή του πειράματος και της αξιολόγησης των μεγάλων γλωσσικών μοντέλων, τα κριτήρια που χρησιμοποιήθηκαν για την αξιολόγηση αυτή αλλά και τα συμπεράσματα για τα τελικά αποτελέσματα.

# ΜΕΓΑΛΑ ΓΛΩΣΣΙΚΑ ΜΟΝΤΕΛΑ

## Τι είναι ένα νευρωνικό δίκτυο και πώς λειτουργεί

### Ορισμός νευρωνικών δικτύων

Ξεκινώντας τη συζήτηση για τα μεγάλα γλωσσικά μοντέλα και τη χρήση τους, δεν θα μπορούσε να παραληφθεί η αναφορά στη βασική συνιστώσα που τα αποτελεί, το **«νευρωνικό δίκτυο»** αλλά και στους τρόπους με τους οποίους αυτό λειτουργεί. Ένα νευρωνικό δίκτυο, όπως γίνεται αντιληπτό από το όνομά του, είναι ένα δίκτυο πολλών συνδεδεμένων νευρώνων, χωρισμένων σε διαφορετικά «στρώματα». Κάθε νευρώνας μπορεί να θεωρηθεί ως μία μονάδα που λαμβάνει κάποιες εισόδους, εκτελεί κάποιους πολύ απλούς υπολογισμούς με αυτές και στη συνέχεια παράγει κάποια έξοδο, ένα νούμερο, το οποίο με την σειρά του περνάει ως είσοδος στους επόμενους νευρώνες.

Οι απλοί αυτοί υπολογισμοί είναι το άθροισμα των εισόδων πολλαπλασιασμένοι με έναν αριθμό, το λεγόμενο «βάρος», το οποίο το νευρωνικό δίκτυο μαθαίνει από χιλιάδες δεδομένα κατά την εκπαίδευσή του. Στην συνέχεια, αφού εφαρμοστεί ένας αριθμητικός μετασχηματισμός, το αποτέλεσμα αυτό μεταφέρεται στους νευρώνες του επόμενου στρώματος ως είσοδος και η διαδικασία αυτή συνεχίζεται μέχρι το τελευταίο στρώμα νευρώνων του νευρωνικού.

Για παράδειγμα, ένας νευρώνας να προωθεί στους νευρώνες του επόμενου στρώματος τον αριθμό μηδέν εάν οι είσοδοι που δέχτηκε ήταν αρνητικοί ή κοντά στο μηδέν, και τον αριθμό ένα στην αντίθετη περίπτωση. Αυτό είναι γνωστό ως «σιγμοειδής συνάρτηση ενεργοποίησης».

Συνεχίζοντας, τα μοντέλα βαθιάς μάθησης χρησιμοποιούν εκατομμύρια νευρώνες, χωρισμένους σε πολυάριθμα στρώματα με δισεκατομμύρια βάρη και πολύπλοκες διατάξεις νευρώνων, αλλά η βασική ιδέα παραμένει η ίδια. (Ανδρουτσόπουλος, 2024). Στην **Εικόνα 1**, παρουσιάζεται μία απλοϊκή μορφή ενός νευρωνικού δικτύου, η οποία αποτελείται από ένα στρώμα εισόδων, ένα εσωτερικό στρώμα και ένα στρώμα εξόδων.

Εικόνα 1. Παράδειγμα απλού νευρωνικού δικτύου

A diagram of a neural network

Description automatically generated

Σημείωση: Ανάκτηση από «Τεχνητή Νοημοσύνη και Μεγάλα Γλωσσικά Μοντέλα», Ι.Ανδρουτσόπουλος, ΟΠΑ News Εφημερίδα Οικονομικού Πανεπιστημίου Αθηνών Τεύχος 51, 2024, σελ.8,([σύνδεσμος](https://www.aueb.gr/el/opanews/51/teyhos-51o-fevroyarios-2024)).

### Ορισμός της μείωσης βαρών με καθοδική κλίση

Αυτό που κάνει τα νευρωνικά δίκτυα τόσο ενδιαφέροντα, και κατά συνέπεια τη μηχανική μάθηση, είναι ότι στην ουσία, ποτέ δεν γράφεται κάποιος αλγόριθμος που να ορίζει συγκεκριμένα τι θα πρέπει να κάνει ένα νευρωνικό δίκτυο και πως θα πρέπει να παράγει αποτελέσματα. Αντίθετα, γράφεται ένας αλγόριθμος που, μέσω της εισαγωγής εκατομμυρίων παραδειγμάτων και δεδομένων με τις ορθές «ετικέτες» τους (δηλαδή το επιθυμητό αποτέλεσμα), μπορεί να μεταβάλλει τα εκατομμύρια βάρη από τα οποία αποτελείται το δίκτυο, ώστε να αποδίδει καλύτερα στα παραδείγματα αυτά.

Τα δεδομένα αυτά ονομάζονται «δεδομένα εκπαίδευσης», και μαζί με τα «δεδομένα δοκιμής», (παραδείγματα που το νευρωνικό δίκτυο δεν έχει «ξαναδεί» και χρησιμοποιούνται για την αξιολόγηση της αποτελεσματικότητάς του), αποτελούν την συνολική αξιολόγηση του συστήματος. Το πρόβλημα αυτό της μεταβολής των βαρών που καλείται να λύσει ο αλγόριθμος καταλήγει να είναι η εύρεση του ελαχίστου μιας «συνάρτησης κόστους».

Η συνάρτηση αυτή υπολογίζεται με βάση τα αποτελέσματα που παράγει το νευρωνικό δίκτυο και τα επιθυμητά αποτελέσματα που έχουμε για κάθε παράδειγμα, και συνεπώς έχει μεγάλη τιμή εάν τα αποτελέσματα διαφέρουν σε μεγάλο βαθμό από τα επιθυμητά, και μικρή τιμή στην αντίθετη περίπτωση. Σκοπός του αλγορίθμου είναι, με βάση τον μέσο όρο όλων αυτών των τιμών κόστους για κάθε παράδειγμα, να προσπαθήσει να μεταβάλλει τις τιμές των βαρών της έτσι ώστε η συνάρτηση κόστους να φτάσει σε ένα τοπικό ελάχιστο.

Εικόνα 2. Παράδειγμα αναπαράστασης του gradient descent

A computer graphics of a red and yellow wavy surface

Description automatically generated with medium confidence

Σημείωση: Ανάκτηση από «Gradient descent, how neural networks learn», G. Sanderson-3blue1brown,2017 ([σύνδεσμος](https://www.3blue1brown.com/lessons/gradient-descent)).

Η τεχνική αυτή ονομάζεται «**καθοδική κλίση»** ή όπως είναι γνωστή, gradient descent, καθώς προσπαθεί να βρει ένα τοπικό ελάχιστο μίας συνάρτησης με πολλές χιλιάδες μεταβλητές (τις εισόδους και τα βάρη). Εάν αναπαρασταθεί σε έναν διανυσματικό χώρο, η συνάρτηση έχει την εικόνα ενός «γεωγραφικού τοπίου» στο οποίο πρέπει να βρεθεί ένα τοπικό «χαμηλότερο σημείο», όπως φαίνεται και στην ***Εικόνα 2***. Χρησιμοποιώντας μαθηματικές έννοιες, όλα τα παραπάνω καταλήγουν τελικά στην εύρεση του αρνητικού του «ανάδελτα» ή ∇ της συνάρτησης κόστους, το οποίο δείχνει προς την κατεύθυνση της πιο απότομης μείωσης της συνάρτησης (Sanderson, 2017).

Εικόνα 3. Παράδειγμα μαθηματικής αναπαράστασης του ανάδελτα

A black and white text with numbers and symbols

Description automatically generated with medium confidence

Σημείωση: Ανάκτηση από «Gradient descent, how neural networks learn», G. Sanderson--3blue1brown,2017 ([σύνδεσμος](https://www.3blue1brown.com/lessons/gradient-descent)).

Κατά συνέπεια, σε αυτό το σημείο θα αναφερθεί περιληπτικά ο ορισμός του αλγορίθμου που προσπαθεί να επιτύχει όλα τα παραπάνω, δηλαδή να υπολογίσει αυτό το τοπικό ελάχιστο της συνάρτησης κόστους, ο οποίος αναφέρεται ως «**αλγόριθμος ανάστροφης μετάδοσης**».

### Ορισμός του αλγορίθμου ανάστροφης μετάδοσης

Όπως παρουσιάστηκε παραπάνω, ο αλγόριθμος ανάστροφης μετάδοσης είναι ένας αλγόριθμος που βρίσκει το τοπικό ελάχιστο μέσω του υπολογισμού του αρνητικού ανάδελτα μίας συνάρτησης κόστους. Αρχικά, ο αλγόριθμος αρχικοποιεί όλα τα βάρη του νευρωνικού δικτύου με τυχαίες μικρές τιμές. Για μία δεδομένη είσοδο ή παράδειγμα εκπαίδευσης, υπολογίζει το συνολικό σφάλμα ή τη συνάρτηση κόστους στην τελική έξοδο, συγκρίνοντας την πραγματική έξοδο με την επιθυμητή έξοδο. Στην συνέχεια, το σφάλμα μεταδίδεται από την έξοδο προς την είσοδο, υπολογίζοντας παράλληλα τους παραγώγους ως προς κάθε ξεχωριστό βάρος με τον κανόνα της αλυσίδας. Κάθε βάρος ενημερώνεται ώστε να κατευθύνεται, με την χρήση της καθοδικής κλίσης, προς την μείωση του σφάλματος. Η διαδικασία αυτή επαναλαμβάνεται για κάθε παράδειγμα εκπαίδευσης που δίνεται στο νευρωνικό δίκτυο, οι οποίες ονομάζονται «εποχές». Η εκπαίδευση τελειώνει είτε όταν το σύστημα ξεπεράσει έναν μέγιστο αριθμό εποχών, είτε όταν το συνολικό σφάλμα μειωθεί σε έναν επιθυμητό αριθμό (Ανδρουτσόπουλος, 2023).

## Τι είναι ένα μεγάλο γλωσσικό μοντέλο

### Ορισμός μεγάλου γλωσσικού μοντέλου

Όπως αναφέρει ο Ανδρουτσόπουλος(2024), ένα μεγάλο γλωσσικό μοντέλο είναι ένα νευρωνικό δίκτυο που δέχεται ως εισόδους λέξεις ή καλύτερα «tokens», σε μορφή αριθμών, τα οποία μπορεί να αποτελούν ένα ημιτελές κείμενο, και παράγει ως εξόδους όλες τις πιθανές λέξεις που θα μπορούσαν να είναι η επόμενη λέξη, επιλέγοντας αυτήν με την μεγαλύτερη πιθανότητα ορθότητας. Έτσι, βασιζόμενοι στο μεγάλο γλωσσικό μοντέλο, μπορούμε να πάρουμε την νέα πρόταση που παρήγαγε, να την ξαναδώσουμε σαν είσοδο και να παράγει ξανά άλλη μία επόμενη πιθανή λέξη. Μετά από έναν αριθμό επαναλήψεων της διαδικασίας αυτής, παράγεται μια ολοκληρωμένη απάντηση που βγάζει νόημα.

### Τι κάνει ένα Transformer

Πιο συγκεκριμένα, τα μεγάλα γλωσσικά μοντέλα διασπούν το κείμενο που δέχονται ως είσοδο σε διαφορετικά tokens, τα οποία δεν είναι απαραίτητα λέξεις, αλλά μπορεί να είναι και κομμάτια λέξεων, και στην συνέχεια τα αναπαριστούν σε διανύσματα. Τα διανύσματα αυτά αποτελούνται από χιλιάδες μεταβλητές και αριθμούς και τοποθετούνται σε έναν πολυδιάστατο χώρο, όπου λέξεις με παρόμοια σημασία, όπως «γάτα» και «σκύλος», βρίσκονται πολύ κοντά. Η χρήση διανυσμάτων επιτρέπει στα μεγάλα γλωσσικά μοντέλα να πραγματοποιούν μαθηματικές πράξεις που αποκαλύπτουν σχέσεις μεταξύ λέξεων. Για παράδειγμα έχει διαπιστωθεί ότι αν από το διάνυσμα της λέξης "μεγαλύτερος" αφαιρεθεί το διάνυσμα της λέξης "μεγάλος" και προστεθεί το διάνυσμα της λέξης "μικρός", το αποτέλεσμα θα είναι το διάνυσμα της λέξης "μικρότερος".

Τα μεγάλα γλωσσικά μοντέλα μπορούν να αναπαριστούν τις λέξεις με διαφορετικά διανύσματα ανάλογα με τα συμφραζόμενα. Αυτό επιτυγχάνεται με τη χρήση ενός αρχιτεκτονικού μοντέλου νευρωνικού δικτύου, γνωστού ως «**transformer»,** που ενημερώνει τα διανύσματα των λέξεων μέσω πολλαπλών επιπέδων. Κάθε μεγάλο γλωσσικό μοντέλο αποτελείται από πολλά στρώματα transformers συνδεδεμένα μεταξύ τους, με σκοπό να εμπλουτίζουν κάθε token με πληροφορίες βάσει των συμφραζομένων, τροποποιώντας έτσι το διάνυσμα του ώστε να έχει την ορθή σημασία που αποκαλύπτεται από τα συμφραζόμενα. Κάθε transformer προσθέτει πληροφορία

Εικόνα 4. Παράδειγμα επικοινωνίας των transformers σε ένα μεγάλο γλωσσικό μοντέλο

A diagram of words and arrows

Description automatically generated with medium confidence

Σημείωση: Ανάκτηση από «Large Language Models, explained with a minimum of math and jargon», T. Lee and S. Trott, 2023, ([σύνδεσμος](https://www.understandingai.org/p/large-language-models-explained-with)).

και βελτιώνει τα διανύσματα έως το τελικό στρώμα (Lee, 2023).

### Τι είναι το Attention Layer

Ένα κύριο συστατικό που επιτρέπει στο transformer να εμπλουτίζει τα διανύσματα των λέξεων είναι το **«attention layer».** Συγκεκριμένα, το στοιχείο αυτό επιτρέπει στα tokens να ανταλλάσσουν πληροφορίες μεταξύ τους και να εμπλουτίζουν το ένα το άλλο, έτσι ώστε να προκύπτει η σωστή σημασία κάθε λέξης. Για παράδειγμα, η λέξη «μοντέλο» έχει διαφορετική σημασία ανάλογα τα συμφραζόμενα, όπως στις φράσεις «μοντέλο μαθηματικών» και «μοντέλο του Χόλυγουντ». Η λειτουργία του attention layer είναι να πραγματοποιεί τον σωστό αυτό διαχωρισμό για κάθε διαφορετικό token με βάση τις λέξεις που το περιτριγυρίζουν. Η διαδικασία αυτή επιτρέπει στα μεγάλα γλωσσικά μοντέλα να μαντεύουν σωστά την επόμενη λέξη σε κάθε κείμενο (Lee, 2023)

### Τι είναι το Feed Forward Step

Μετά τη μεταφορά πληροφοριών ανάμεσα στα διανύσματα λέξεων από τα attention heads, τo transformer περιλαμβάνει ένα ακόμα συστατικό, το στρώμα **«feed forward»**. Το στρώμα αυτό «εξετάζει» κάθε διάνυσμα λέξης και προσπαθεί να προβλέψει την επόμενη λέξη. Σε αυτό το στάδιο, δεν γίνεται ανταλλαγή πληροφοριών μεταξύ των λέξεων, το στρώμα feed forward αναλύει κάθε λέξη μεμονωμένα. Ωστόσο, το στρώμα feed forward έχει πρόσβαση σε οποιαδήποτε πληροφορία έχει αντιγραφεί από όλα τα προηγούμενα attention heads, ώστε να μπορέσει αποτελεσματικότερα να προβλέψει την επόμενη λέξη (Lee, 2023).

## Ιστορία και εξέλιξη των μεγάλων γλωσσικών μοντέλων

### Το chatbot ELIZA

Ξεκινώντας μια ιστορική αναδρομή της εξέλιξης των μεγάλων γλωσσικών μοντέλων, θα αναφερθούμε στο chatbot **«ELIZA»**, το οποίο κατασκευάστηκε το 1996 και θεωρείται το πρώτο chatbot που δημιουργήθηκε από ανθρώπους. Ο δημιουργός του, Joseph Weizenbaum, τοι ανέπτυξε στο πανεπιστήμιο του MIT. Ο τρόπος λειτουργίας του ELIZA, ήταν να δημιουργήσει την ψευδαίσθηση συνομιλίας με την τεχνική της αναδιατύπωσης δηλώσεων των χρηστών ως ερωτήσεις. Εκείνη την εποχή, δημιουργήθηκαν πολλές παραλλαγές του συγκεκριμένου chatbot οι οποίες λειτουργούσαν με παρόμοιο τρόπο και μια από τις πιο γνωστές ονομάζεται «DOCTOR», το οποίο ανταποκρινόταν σαν ψυχοθεραπευτής. Αυτή η αρχή έβαλε τις βάσεις για περεταίρω έρευνα στον τομέα των chatbots και της επεξεργασίας φυσικής γλώσσας. (Pi, 2024)

Εικόνα 5. Ένα παράδειγμα συζήτησης με το chatbot ELIZA

A screenshot of a computer program

Description automatically generated

Σημείωση: Ανάκτηση από «ELIZA», In Wikipedia, The Free Encyclopedia (2024, July 15), ([σύνδεσμος](https://en.wikipedia.org/w/index.php?title=ELIZA&oldid=1234749943)).

### Άνοδος των νευρωνικών δικτύων

Όπως αναφέρει ο Pi(2024) στο άρθρο του, προς τα τέλη του 20ου αιώνα εμφανίστηκαν τα νευρωνικά δίκτυα, εμπνευσμένα βαθιά από τον ανθρώπινο εγκέφαλο, όπως γίνεται φανερό από την ονομασία τους αλλά και την αρχιτεκτονική τους με διασυνδεδεμένους νευρώνες. Το 1986, αναφέρεται ως η χρονιά που έκαναν την εμφάνισή τους τα «Επαναλαμβανόμενα Νευρωνικά Δίκτυα (RNN), τα οποία σε αντίθεση με τα παραδοσιακά προωθητικά νευρωνικά δίκτυα, όπου η ροή των πληροφοριών είχε μονάχα μία κατεύθυνση, μπορούσαν να θυμούνται προηγούμενες εισόδους και να απαντούν με βάση το ευρύτερο πλαίσιο. Έτσι, εκπαιδεύονταν να επεξεργάζονται και να μετατρέπουν μια ακολουθία δεδομένων εισόδου σε συγκεκριμένη ακολουθία δεδομένων εξόδου. Ωστόσο, τα RNN είχαν τον πολύ μεγάλο περιορισμό στην «μνήμη», κάτι αντίστοιχο με το σημερινό context size των μεγάλων γλωσσικών μοντέλων όπως του ChatGPT, το οποίο τα κάνει να φαίνονται σαν να «ξεχνούν» πληροφορίες από προηγούμενα μηνύματα.

### Δημιουργία των LSTM

Το 1997 εμφανίστηκε η Μνήμη Μακράς Βραχείας Διάρκειας (LSTM), μια εξειδικευμένη μορφή RNN που βελτίωνε το πρόβλημα της διατήρησης πληροφορίας για μεγάλες ακολουθίες. Συγκεκριμένα, το εργαλείο αυτό είχε μια μοναδική αρχιτεκτονική που αποτελούνταν από πύλες εισόδου, λήθης και πύλες εξόδου, επιτρέποντας έτσι τη διατήρηση και τη διαχείριση πληροφοριών για μεγαλύτερα χρονικά διαστήματα (Pi, 2024).

### Δημιουργία Gated Recurrent Network

Το 2014, εμφανίστηκαν οι Μονάδες Επαναλαμβανόμενων Δικτύων με Πύλες (GRU), σχεδιασμένες για να επιλύουν τα ίδια προβλήματα με τα LSTM, αλλά με απλούστερη δομή. Οι GRU χρησιμοποιούσαν δύο πύλες: μια πύλη ενημέρωσης και μια πύλη επαναφοράς. (Pi, 2024)

### Άνοδος του συστατικού Attention

Τελικά, οι τεχνολογίες RNN, LSTM και GRU αποδείχτηκε ότι δεν είναι τόσο αποτελεσματικές στην διατήρηση των συμφραζόμενων όταν αυτά επεκτείνονται σε μεγάλο βαθμό. Έτσι, δημιουργήθηκε ο μηχανισμός που ονομάστηκε **«Attention»**, ο οποίος προσέφερε μία νέα οπτική στα μεγάλα γλωσσικά μοντέλα. Συγκεκριμένα, το attention επέτρεψε στο μοντέλο να «κοιτάει» πίσω σε ολόκληρο το διαθέσιμο υλικό δυναμικά, επιλέγοντας τα πιο σημαντικά κομμάτια που προσθέτουν σημασία στις υπόλοιπες προτάσεις. (Pi, 2024)

Εικόνα 6. Σύγκριση της απόδοσης Attention και RNN

A graph with a line drawn on it

Description automatically generated

Σημείωση: Ανάκτηση από «Brief Introduction to the History of Large Language Models (LLMs)», W. P,2024 ([σύνδεσμος](https://medium.com/@researchgraph/brief-introduction-to-the-history-of-large-language-models-llms-3c2efa517112))

### Η εφεύρεση των Transformers

Το 2017 ήταν το έτος που πρωτοεμφανίστηκε η έννοια των **«Transformers»,** στο paper με τίτλο «Attention is all you need» από τον Vaswani και τους συνεργάτες του στην Google. Η νέα αυτή αρχιτεκτονική, όπως εξηγήθηκε και στην προηγούμενη υποενότητα, χρησιμοποιούσε ως βασικό της εργαλείο τον μηχανισμό attention για να επεξεργαστεί τα δεδομένα εισόδου και ήταν ικανή να επεξεργάζεται ακολουθίες παράλληλα, χωρισμένη σε πολλά στρώματα. Αυτή η προσέγγιση έθεσε τα θεμέλια για μεταγενέστερα μοντέλα όπως το ChatGPT (Pi, 2024).

### Εμφάνιση μεγάλων γλωσσικών μοντέλων

Με την μεγάλη επιτυχία των transformers, το επόμενο λογικό βήμα ήταν η αύξηση της κλίμακας. Αυτό ξεκίνησε με το μοντέλο BERT της Google το 2018 και συνεχίστηκε με την κυκλοφορία των GPT-2 το 2019,του GPT-3 το 2020, καθώς και των νέων εκδόσεων αυτού όπως το GPT-3.5, GPT-4 και GPT-4o (Pi, 2024).

## Κύριες εφαρμογές και χρήσεις

Με την τεράστια εξέλιξη των μεγάλων γλωσσικών μοντέλων, αυτά έχουν γίνει καθημερινό εργαλείο για πολλούς ανθρώπους, εξοικονομώντας χρόνο, καθώς το μέγεθός τους από δεδομένα και η πολυπλοκότητά τους επιτρέπει σε στιγμιαίες απαντήσεις χωρίς καθυστέρηση. Το κόστος τους είναι ελάχιστο, αφού τα περισσότερα είναι διαθέσιμα δωρεάν σε απλούστερες αλλά πολύ αποτελεσματικές μορφές, ενώ η χρήση τους απαιτεί ελάχιστες γνώσεις. Όλα τα παραπάνω έχουν ωθήσει τα μεγάλα γλωσσικά μοντέλα να βρουν πολυάριθμες χρήσεις σε διάφορους τομείς. Όπως αναφέρει και ο Sumrak(2024), κάποια από τα δημοφιλέστερα παραδείγματα χρήσης των μεγάλων γλωσσικών μοντέλων περιλαμβάνουν την ανάλυση ήχου, τη δημιουργία περιεχομένου, την υποστήριξη πελατών, τη μετάφραση γλωσσών, την εκπαίδευση και την κυβερνοασφάλεια.

### Ανάλυση ήχου

Τα μεγάλα γλωσσικά μοντέλα έχουν αναθεωρήσει τον τρόπο με τον οποίο οι άνθρωποι χειρίζονται τα δεδομένα ήχου. Συγκεκριμένα, έχουν την ικανότητα να ακούν πολύωρες συζητήσεις και να παράγουν αποτελεσματικές περιλήψεις καθώς και να απαντούν ερωτήσεις σχετικά με μακροσκελείς συναντήσεις. Ακόμη, μπορούν, με βάση ένα μεγάλο ποσοστό κλήσεων ως δεδομένα, να εξάγουν πολύπλοκα αποτελέσματα και συμβουλές βελτίωσης (Sumrak, 2024).

### Δημιουργία περιεχομένου

Τα μεγάλα γλωσσικά μοντέλα χρησιμοποιούνται αποτελεσματικά από συγγραφείς και εμπόρους για την δημιουργία αρχικών σχεδίων(drafts), την πρόταση διάφορων αλλαγών και για την ταχεία εύρεση άρθρων και αναφορών στο διαδίκτυο. Αυτά τα εργαλεία επιταχύνουν σημαντικά την παραγωγικότητα, επιτρέποντάς στους χρήστες να επικεντρωθούν παραπάνω στο πιο απαιτητικό και δημιουργικό κομμάτι της δουλείας τους, αφήνοντας τα μοντέλα να ασχοληθούν με τα μηχανικά στοιχεία της εργασίας τους (Sumrak, 2024).

### Υποστήριξη πελατών

Ένας τομέας στον οποίο τα μεγάλα γλωσσικά μοντέλα έχουν κάνει ραγδαία άνοδο και χρησιμοποιούνται κατά κόρον, είναι η εξυπηρέτηση πελατών. Πολλές εταιρείες τηλεφωνίας, σελίδες του δημοσίου τομέα και μεγάλες επιχειρήσεις έχουν υιοθετήσει μοντέλα εξυπηρέτησης πελατών μέσω μεγάλων γλωσσικών μοντέλων. Αυτά είναι διαθέσιμα 24/7 χωρίς την ανάγκη ανθρώπινης παρέμβασης, προσφέροντας συνεχή βοήθεια σε χρήστες από όλο τον κόσμο με πολύ χαμηλό κόστος (Sumrak, 2024).

### Μετάφραση γλωσσών

Τα μεγάλα γλωσσικά μοντέλα βοηθούν στην άρση των γλωσσικών φραγμών, δίνοντας την δυνατότητα στις επιχειρήσεις να προσεγγίζουν πελάτες αλλά και να προσλαμβάνουν άτομα από όλες τις χώρες. Αυτά τα μοντέλα προσφέρουν ακριβείς υπηρεσίες μετάφρασης σε πραγματικό χρόνο, κάνοντας ιστότοπους, εφαρμογές και ψηφιακό περιεχόμενο παγκοσμίως προσβάσιμα (Sumrak, 2024).

### Εκπαίδευση

Στον τομέα της εκπαίδευσης, τα μεγάλα γλωσσικά μοντέλα χρησιμοποιούνται για την παροχή εξατομικευμένης μάθησης προσαρμόζοντας το περιεχόμενο στις ατομικές ανάγκες των μαθητών, δημιουργώντας ερωτήσεις κατανόησης και παρέχοντας λεπτομερείς εξηγήσεις προσαρμοσμένες στις ανάγκες των μαθητών (Sumrak, 2024).

### Κυβερνοασφάλεια

Τα μεγάλα γλωσσικά μοντέλα μπορούν να χρησιμοποιηθούν στην ανάλυση και την ερμηνεία μεγάλων ποσών δεδομένων κυβερνοασφάλειας, προβλέποντας, αναγνωρίζοντας και ανταποκρινόμενα σε πιθανές απειλές ασφαλείας. Η στοχευμένης εκπαίδευσή τους επιτρέπει ταχύτερη και πιο ακριβή ανίχνευση και ανταπόκριση στις απειλές, ενισχύοντας την ασφάλεια των επιχειρήσεων (Sumrak, 2024).

## Παραδείγματα Μεγάλων Γλωσσικών Μοντέλων

Ολοκληρώνοντας την ανάλυση των μεγάλων γλωσσικών μοντέλων, είναι σημαντικό να αναφερθούν κάποια από τα πιο πολυχρησιμοποιούμενα μοντέλα και οι κυριότερες λειτουργίες και διαφοροποιήσεις τους.

### BERT

Το πρώτο μεγάλο γλωσσικό μοντέλο που αναπτύχθηκε ήταν το **BERT**(Bidirectional Encoder Representations from Transformers), δημιουργημένο από την Google το έτος 2018. Το BERT είναι ένα μοντέλο που χρησιμοποιεί την αρχιτεκτονική των transformers με πολυάριθμα στρώματα συνδεδεμένα μεταξύ τους και διαθέτει 342 εκατομμύρια παραμέτρους για την επεξεργασία εισόδων. Εκπαιδεύτηκε σε τεράστιο όγκο δεδομένων για να παράγει απαντήσεις σε φυσική γλώσσα που να είναι κατανοητές από τους ανθρώπους (Lutkevich, 2024).

### GEMINI

Ακολουθώντας την Google, το ανανεωμένο μοντέλο της ονομάστηκε **GEMINI**, το οποίο έχει την ονομασία του από το chatbot της εταιρείας. Το GΕΜΙΝΙ είναι ένα πολυτροπικό μοντέλο, ικανό να επεξεργάζεται ήχο, εικόνα, βίντεο και κείμενο, σε αντίθεση με άλλα γλωσσικά μοντέλα που περιορίζονται μόνο σε κείμενο. Το GEMINI διατίθεται σε τρία «μεγέθη», το Ultra, το Pro και το Nano, από το μεγαλύτερο και πιο ικανό έως το μικρότερο και λιγότερο ικανό. Υπάρχουν αναφορές ότι το GEMINI υπερτερεί σε δύναμη από το μοντέλο GPT -4 της OpenAI, στο οποίο θα γίνει αναφορά στην συνέχεια. (Lutkevich, 2024).

### GPT –3

Το **GPT -3**(Generative Pre-trained Transformer 3) αποτελεί το πρώτο ισχυρό μοντέλο της OpenAI το οποίο παρουσιάστηκε το 2020 με περισσότερες από 175 δισεκατομμύρια παραμέτρους (το BERT έχει 342 εκατομμύρια). Ενσωματώνει την αρχιτεκτονική των συνδεδεμένων στρωμάτων transformers και είναι δέκα φορές μεγαλύτερο από τον προκάτοχο του, το GPT -2. Είναι εκπαιδευμένο με εκατομμύρια δεδομένα και αποτελεί το τελευταίο μοντέλο από την σειρά μοντέλων που παρήγαγε η OpenAI, για το οποίο ήταν δημόσια γνωστός ο αριθμός των παραμέτρων που χρησιμοποιούσε (Lutkevich, 2024).

### GPT -3.5 και GPT –3.5 Turbo

Το **GPT –3.5** αποτελεί την ενημερωμένη έκδοση του GPT –3 με λιγότερες παραμέτρους αλλά ποιοτικότερη εκπαίδευση, και διαδραμάτισε κρίσιμο ρόλο στην πλατφόρμα ChatGPT, η οποία το 2023 άλλαξε ριζικά τη δημοτικότητα και χρήση των μεγάλων γλωσσικών μοντέλων. Το GPT –3.5 εκπαιδεύτηκε με γνώσεις μέχρι και τον Σεπτέμβρη του 2021 και δεν έχει την δυνατότητα πρόσβασης στο διαδίκτυο συγκριτικά με άλλα μεγάλα γλωσσικά μοντέλα Παράλληλα, η πιο ισχυρή έκδοση του GPT-3.5 είναι το **GPT-3.5 Turbo**, που χρησιμοποιείται από το GitHub Copilot της Microsoft για την υποστήριξη των προγραμματιστών με παραγωγή κώδικα και ανίχνευση σφαλμάτων (Lutkevich, 2024).

### GPT –4

Το **GPT-4** αποτελεί το μεγαλύτερο μοντέλο της σειράς GPT της OpenAI, κυκλοφορώντας το 2023. Όπως και τα προηγούμενα μοντέλα, βασίζεται στην αρχιτεκτονική των transformers. Ωστόσο, ο αριθμός των παραμέτρων του δεν έχει δημοσιοποιηθεί, με φήμες να αναφέρουν πάνω από 170 **τρισεκατομμύρια** παραμέτρους. Η OpenAI περιγράφει το GPT-4 ως ένα πολυτροπικό μοντέλο(όπως και το GEMINI), πράγμα που σημαίνει ότι είναι ικανό να επεξεργάζεται κείμενο, ήχο και εικόνα, σε αντίθεση με τα προηγούμενα μοντέλα της OpenAI που περιορίζονταν μόνο στο κείμενο. Το GPT-4, υποστηρίζεται ότι πλησίασε την τεχνητή γενική νοημοσύνη (AGI), που σημαίνει ότι είναι εξίσου έξυπνο ή εξυπνότερο από έναν άνθρωπο και σημαντικότερα έχει την δυνατότητα της πρόσβασης στο διαδίκτυο, σε αντίθεση με το GPT –3.5 (Lutkevich, 2024).

### GPT –4o

Το **GPT -4ο (**GPT -4 Omni)είναι ο διάδοχος του GPT-4 και το πιο πρόσφατο μεγάλο γλωσσικό μοντέλο της OpenAI . Προσφέρει αρκετές βελτιώσεις σε σχέση με τοGPT -4 με μια πιο φυσική ανθρώπινη αλληλεπίδραση για το ChatGPT. Το GPT-4o είναι επίσης πολυτροπικό, αλλά με τη δυνατότητα να βλέπει φωτογραφίες ή οθόνες και να κάνει σχετικές ερωτήσεις κατά τη διάρκεια της αλληλεπίδρασης.

# ΜΕΓΑΛΑ ΓΛΩΣΣΙΚΑ ΜΟΝΤΕΛΑ ΣΤΗΝ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ

## Ρόλος των μεγάλων γλωσσικών μοντέλων στην ανάπτυξη λογισμικού

### Εισαγωγή στην έννοια των γλωσσικών μοντέλων στην ανάπτυξη λογισμικού

Τα μεγάλα γλωσσικά μοντέλα είναι προηγμένα συστήματα τεχνητής νοημοσύνης που ανταποκρίνονται σε ερωτήσεις και συζητήσεις χρηστών με απαντήσεις σε ανθρώπινη γλώσσα. Έχουν την ικανότητα να κατανοούν βαθιά τα συμφραζόμενα και τις ανάγκες του χρήστη, κάνοντάς τα χρήσιμα σε πολλούς τομείς της καθημερινότητας και ιδιαίτερα στην ανάπτυξη λογισμικού. Τα γλωσσικά μοντέλα μπορούν να βοηθήσουν προγραμματιστές, αυξάνοντας την παραγωγικότητά τους και μειώνοντας τον χρόνο εκσφαλμάτωσης, καθώς έχουν εκπαιδευτεί σε δισεκατομμύρια έργα ανοιχτού κώδικα και ενημερώνονται συνεχώς με νέα δεδομένα.

Ο εσωτερικός τρόπος λειτουργίας των μεγάλων γλωσσικών μοντέλων τα καθιστά εξαιρετικά στην υποβοήθηση πάρα πολλών έργων και στην λύση των δυσκολιών που μπορεί να προκύψουν για έναν προγραμματιστή κατά την διάρκεια της διαδικασίας της ανάπτυξης λογισμικού, αυξάνοντας έτσι σημαντικά την παραγωγικότητα, μειώνοντας τον χρόνο εκσφαλμάτωσης και συνεπώς και τον χρόνο που απαιτείται για την ολοκλήρωση ενός έργου. Όλα αυτά οφείλονται στο γεγονός ότι τα μεγάλα γλωσσικά μοντέλα έχουν την ικανότητα να γράφουν, να διορθώνουν και να βελτιστοποιούν τον κώδικα πολύ γρήγορα και με μεγάλη ακρίβεια, καθώς έχουν εκπαιδευτεί σε δισεκατομμύρια έργα ανοιχτού κώδικα, όπως αυτά στο GitHub, ενώ παράλληλα ενημερώνονται συνεχώς με νέα δεδομένα.

### Παραδείγματα χρήσης μεγάλων γλωσσικών μοντέλων στην ανάπτυξη λογισμικού

Ορισμένα από τα σημαντικότερα παραδείγματα χρήσης των μεγάλων γλωσσικών μοντέλων στην ανάπτυξη λογισμικού περιλαμβάνουν την αυτοματοποίηση γραφής κώδικα, την ανάλυση και διόρθωση λαθών καθώς και την ανάλυση των απαιτήσεων και την μετατροπή τους σε τεχνικές προδιαγραφές.

Η αυτοματοποίηση γραφής κώδικα μπορεί να εξοικονομήσει πολύ χρόνο και να μειώσει τα λάθη, καθώς τα γλωσσικά μοντέλα μπορούν να δημιουργούν κώδικα βάσει των περιγραφών των προγραμματιστών και να προτείνουν βελτιώσεις. Η ανάλυση και η διόρθωση λαθών επιτρέπει στους προγραμματιστές να εντοπίζουν και να διορθώνουν σφάλματα γρήγορα και αποτελεσματικά και η μετατροπή των απαιτήσεων σε τεχνικές προδιαγραφές διευκολύνει την κατανόηση των αναγκών του έργου και την ακριβή μετάφρασή τους σε τεχνικές λεπτομέρειες.

### Αυτοματοποίηση κώδικα

Στον τομέα του αυτοματοποιημένου κώδικα, τα μεγάλα γλωσσικά μοντέλα μπορούν να προτείνουν ή να γράφουν αποσπάσματα κώδικα βάσει των περιγραφών των προγραμματιστών. Παρέχουν επίσης προτάσεις κώδικα και σχόλια σε πραγματικό χρόνο, όπως το GitHub Copilot της Microsoft, το οποίο αποτελεί ένα από τα μοντέλα που εξετάστηκαν στην εργασία.

Τα μεγάλα γλωσσικά μοντέλα μπορούν να κατανοούν περιγραφές σε φυσική γλώσσα και να μετατρέπουν αυτές τις περιγραφές σε λειτουργικό κώδικα, με αποτέλεσμα να μειώνεται η ανάγκη για λεπτομερή καθοδήγηση και να επιτρέπει στους προγραμματιστές να εστιάσουν σε πιο δημιουργικές και πολύπλοκες πτυχές της ανάπτυξης λογισμικού. Η ικανότητα να παρέχουν συνεχείς ενημερώσεις και βελτιώσεις στον κώδικα καθιστά τα μεγάλα γλωσσικά μοντέλα έναν πολύτιμο συνεργάτη στην ανάπτυξη λογισμικού.

### Ανάλυση και διόρθωση λαθών

Τα μεγάλα γλωσσικά μοντέλα είναι ικανά να αναγνωρίζουν με μεγάλη ευκολία και ταχύτητα σφάλματα που έχουν συμβεί κατά την διάρκεια της συγγραφής κώδικα, να τα αναλύουν και να προσφέρουν λύσεις με βάση πολύπλοκες μεθόδους εκσφαλμάτωσης, ενώ μπορούν να δίνουν και τις ίδιες τις διορθώσεις σε όλα τα πιθανά λάθη που έχουν εντοπιστεί. Ακόμη, βελτιώνουν την ποιότητα του κώδικα εντοπίζοντας σημεία που επιδέχονται βελτίωση και παρέχοντας προτάσεις για πιο ποιοτικό αποτέλεσμα.

Η ικανότητα των μεγάλων γλωσσικών μοντέλων να κατανοούν σε τόσο μεγάλο βάθος τον κώδικα, τους επιτρέπει να εντοπίζουν λάθη που μπορεί να μην είναι προφανή άμεσα σε έναν προγραμματιστή. Αυτό μπορεί να περιλαμβάνει σφάλματα σύνταξης, λογικά σφάλματα ή ακόμη και προβλήματα βελτιστοποίησης. Η παροχή συγκεκριμένων προτάσεων για τη διόρθωση αυτών των λαθών βοηθά στην εξασφάλιση της ποιότητας και της αξιοπιστίας του κώδικα.

### Ανάλυση των απαιτήσεων μετατροπή τους σε τεχνικές προδιαγραφές

Τα μεγάλα γλωσσικά μοντέλα μπορούν να διαβάζουν και να κατανοούν τα διάφορα έγγραφα απαιτήσεων, να τα διορθώνουν και να χρησιμοποιούν φυσική γλώσσα για να εξάγουν τις βασικές ανάγκες και προδιαγραφές που πρέπει να περιγραφούν στους εμπλεκόμενους ενός μεγάλου έργου.

Η κατανόηση και η μετάφραση των απαιτήσεων σε τεχνικές προδιαγραφές είναι ένα κρίσιμο βήμα στην διαδικασία της ανάπτυξης λογισμικού. Συγκεκριμένα, τα μεγάλα γλωσσικά μοντέλα μπορούν να αναλύουν πολύπλοκες περιγραφές και να τις μετατρέπουν σε κατανοητές προδιαγραφές που μπορούν να χρησιμοποιηθούν από όλα τα εμπλεκόμενα μέρη. Συνεπώς αυτό μειώνει την πιθανότητα ασαφειών(έννοια που περιγράφεται σε επόμενη υποενότητα) και εξασφαλίζει ότι όλες οι απαιτήσεις πληρούνται με ακρίβεια.

## Χρήσεις σε διάφορα στάδια της ανάπτυξης λογισμικού

### Στάδιο ανάλυσης απαιτήσεων

Η ανάλυση των απαιτήσεων είναι μία κρίσιμη διαδικασία για την επιτυχή ανάπτυξη ενός συστήματος λογισμικού. Ένα σύνηθες πρόβλημα κατά το στάδιο της ανάλυσης των απαιτήσεων, είναι η ύπαρξη ασαφειών, δηλαδή η διαφορετική ερμηνεία της ίδιας απαίτησης από πολλά άτομα, το οποίο μπορεί να οδηγήσει σε σοβαρά προβλήματα στα υπόλοιπα στάδια της ανάπτυξης λογισμικού. Σύμφωνα με τον Hou και συν. (2023), το ChatGPT, υπερτερεί σε μεγάλο βαθμό σε σχέση με τα άλλα γλωσσικά μοντέλα. Παράλληλα, όπως ανέδειξε και μια έρευνα που αναφέρει ο Hou και οι συν. στην οποία δόθηκαν κάποιες απαιτήσεις με μεγάλες προκλήσεις στην ασάφεια, το γλωσσικό μοντέλο κατάφερε να εντοπίσει με ακρίβεια τα λάθη και να τα διορθώσει σε κάθε περίπτωση. Επομένως, τα μεγάλα γλωσσικά αποτελούν ένα πολύ σημαντικό εργαλείο στην εξάλειψη των ασαφειών στο στάδιο της ανάλυσης απαιτήσεων, συμβάλλοντας σε ένα ποιοτικότερο αποτέλεσμα.

### Στάδιο σχεδιασμού

Στο στάδιο του σχεδιασμού είναι επίσης κρίσιμο σημείο για την επιτυχία ενός έργου ανάπτυξης λογισμικού. Όπως αναφέρει ο Hou και συν.(2023), ενώ η έρευνα γύρω από τα μεγάλα γλωσσικά μοντέλα στον τομέα του σχεδιασμού δεν έχει προχωρήσει σε βάθος σε σύγκριση με άλλα στάδια της ανάπτυξης, όπως η κωδικοποίηση και η βελτιστοποίηση, τα μεγάλα γλωσσικά μοντέλα προσφέρουν σημαντική υποστήριξη με ποικίλους άλλους τρόπους. Ειδικότερα, τα γλωσσικά μοντέλα μπορούν να βελτιώσουν την υπάρχουσα αρχιτεκτονική και να προτείνουν διάφορες εναλλακτικές λύσεις, εάν αυτό κριθεί απαραίτητο. Για παράδειγμα, μπορούν να προτείνουν συγκεκριμένα πρότυπα σχεδίασης που να ταιριάζουν περισσότερο με τις ανάγκες του έργου.

Ακόμη, τα μεγάλα γλωσσικά μοντέλα μπορούν να συνεισφέρουν στην δημιουργία διαγραμμάτων, όχι μέσω της παραγωγής τους απευθείας αλλά μέσω της περιγραφής μέσω μηνυμάτων, στις περιγραφές συστημάτων και στις τεχνικές προδιαγραφές.

Επιπρόσθετα, τα μεγάλα γλωσσικά μοντέλα μπορούν να εντοπίζουν σημεία όπου μπορεί να υπάρξουν πιθανά σφάλματα απόδοσης ή ασφάλειας και να προτείνουν βελτιώσεις, συμβάλλοντας σε μία αποδοτικότερη και ασφαλέστερη αρχιτεκτονική.

### Στάδιο κωδικοποίησης

Το στάδιο της κωδικοποίησης είναι το πιο γνωστό και συχνά το πιο χρονοβόρο στάδιο στην ανάπτυξη λογισμικού. Σε αυτό το στάδιο, οι προγραμματιστές γράφουν τον κώδικα που θα υλοποιήσει τις απαιτήσεις και τις προδιαγραφές του έργου τους. Τα μεγάλα γλωσσικά μοντέλα παρουσιάζουν συνεχώς βελτιωμένες επιδόσεις με κάθε νέο μοντέλο που κυκλοφορεί. Όπως αναφέρει ο Hou και συν. (2023), μοντέλα όπως τα GPT-4, GPT-3, GPT-3.5, BERT series, Codex, CodeGen, InCoder, Copilot και CodeGeeX έχουν πολύ σημαντικό ρόλο στην κωδικοποίηση. Η εκπαίδευσή τους σε μεγάλους όγκους δεδομένων και συγκεκριμένα κειμένων, τους επιτρέπει να κατανοούν βαθιά τη φυσική γλώσσα και να τη μετατρέπουν σε κώδικα, ενώ παράλληλα τα μοντέλα αποδίδουν εξαιρετικά στην παραγωγή κώδικα με περιγραφές σε φυσική γλώσσα, προτείνοντας διορθώσεις και συμπληρώσεις σε πραγματικό χρόνο.

Ακόμη, τα μοντέλα αυτά συνεισφέρουν και στην δημιουργία τεκμηρίωσης, ένα σημαντικό αλλά συχνά παραμελημένο κομμάτι της ανάπτυξης λογισμικού. Τα γλωσσικά μοντέλα μπορούν να παράγουν αυτόματα σχόλια και τεκμηρίωση για τον κώδικα, διευκολύνοντας την κατανόηση και τη συντήρηση του κώδικα από άλλους προγραμματιστές.

Επιπλέον, τα μεγάλα γλωσσικά μοντέλα υποστηρίζουν πολλές γλώσσες προγραμματισμού, καθιστώντας τα χρήσιμα για πολυγλωσσικά έργα που απαιτούν χρήση πολλών γλωσσών, εφόσον μπορούν να αναγνωρίζουν τα συμφραζόμενα και να προσαρμόζουν τις προτάσεις τους ανάλογα με τη γλώσσα προγραμματισμού που χρησιμοποιείται κάθε φορά (Zharovskikh, 2023)

### Στάδιο ελέγχου και βελτιστοποίησης

Το στάδιο του ελέγχου και της βελτιστοποίησης είναι ζωτικής σημασίας για τη διασφάλιση της ποιότητας και της απόδοσης του λογισμικού. Κατά τη διάρκεια αυτού του σταδίου, οι προγραμματιστές και οι δοκιμαστές εντοπίζουν και διορθώνουν σφάλματα, βελτιστοποιούν την απόδοση και διασφαλίζουν ότι το λογισμικό πληροί τις απαιτήσεις των χρηστών. Ο έλεγχος και η βελτιστοποίηση διαρκούν συνήθως αρκετό καιρό, καθώς απαιτείται λεπτομερής έλεγχος όλων των κομματιών κώδικα που έχουν δημιουργηθεί. Επιπλέον, η διαδικασία αυτή συνεχίζεται και μετά την παράδοση του τελικού προϊόντος, εφόσον προκύψουν νέες απαιτήσεις από τους χρήστες και τον εμπλεκόμενους του έργου.

Τα μεγάλα γλωσσικά μοντέλα προσφέρουν πολύτιμη υποστήριξη σε αυτό το στάδιο, μέσω της δημιουργίας αυτοματοποιημένων τεστ, τον εντοπισμό σφαλμάτων και την βελτιστοποίηση του κώδικα.

Συγκεκριμένα, τα μεγάλα γλωσσικά μοντέλα μπορούν να δημιουργούν αυτόματα μονάδες ελέγχου (unit tests) και ενσωματωμένα τεστ (integration tests) βάση του κώδικα που έχει γραφτεί. Με αυτόν τον τρόπο εξασφαλίζεται ότι κάθε κομμάτι κώδικα λειτουργεί σωστά και αποδοτικά.

Ακόμη, τα μεγάλα γλωσσικά μοντέλα μπορούν να αναλύουν μεγάλα κομμάτια κώδικα και να εντοπίζουν πιθανά συντακτικά και λογικά σφάλματα που έχουν συμβεί. Παρέχουν επίσης προτάσεις για τη διόρθωση αυτών των σφαλμάτων, διευκολύνοντας τους προγραμματιστές στη διαδικασία της εκσφαλμάτωσης.

Ακόμη, τα μοντέλα αυτά έχουν την δυνατότητα να εντοπίζουν αναποτελεσματικά κομμάτια κώδικα που μπορεί να δίνουν το επιθυμητό αποτέλεσμα, αλλά όχι με τον βέλτιστο τρόπο, και να προτείνουν βέλτιστες πρακτικές και βελτιώσεις, οδηγώντας σε πιο ποιοτικό κώδικα.

## Πλεονεκτήματα και περιορισμοί

Η χρήση των μεγάλων γλωσσικών μοντέλων στην ανάπτυξη λογισμικού προσφέρει πληθώρα πλεονεκτημάτων και νέες δυνατότητες στους προγραμματιστές, με ορισμένους όμως περιορισμούς στην ορθή χρήση και τον έλεγχο των αποτελεσμάτων.

### Πλεονεκτήματα

Καταρχάς, η ικανότητα των μεγάλων γλωσσικών μοντέλων να δημιουργούν με τόσο μεγάλη ταχύτητα κώδικα βελτιώνει την αποδοτικότητα και μειώνει τα σφάλματα κατά τη διαδικασία της κωδικοποίησης, ενώ παράλληλα λειτουργεί και ως ένα εκπαιδευτικό εργαλείο για τον προγραμματιστή, αφού προσφέρει άμεσα εξηγήσεις και απαντήσεις σχετικά με τις διορθώσεις του κώδικα (Hurani & Idris, 2024).

Ακόμη, μέσω των διορθώσεων και των συμπληρώσεων σε πραγματικό χρόνο που προσφέρουν αυτά τα μοντέλα, επιταχύνουν την ανάπτυξη λογισμικού και διασφαλίζουν υψηλή ποιότητα στον παραγόμενο κώδικα.

Επιπλέον, η δυνατότητά τους να υποστηρίζουν πολλές γλώσσες προγραμματισμού τα καθιστά ιδιαίτερα χρήσιμα για πολυγλωσσικά έργα, όπου η κατανόηση μεγάλου όγκου κώδικα σε διαφορετικές γλώσσες προγραμματισμού μπορεί να είναι μία πρόκληση για τους εμπλεκόμενους προγραμματιστές. (Zharovskikh, 2023)

Τέλος, η εκπαίδευσή των μεγάλων γλωσσικών μοντέλων σε μεγάλα σύνολα δεδομένων και projects, επιτρέπει βαθιά κατανόηση και εφαρμογή των βέλτιστων τεχνικών της ανάπτυξης λογισμικού, κάνοντάς τα πολύτιμα εργαλεία για προγραμματιστές και μεγάλες ομάδες ανάπτυξης.

### Περιορισμοί

Σε συνδυασμό με τα πλεονεκτήματα, η εκτεταμένη χρήση των μεγάλων γλωσσικών μοντέλων κατά την διαδικασία της ανάπτυξης λογισμικού συνεπάγεται και ορισμένους περιορισμούς.

Συγκεκριμένα, ενώ τα μεγάλα γλωσσικά μοντέλα διαπρέπουν σε απλά σενάρια και έργα, σε πολλές περιπτώσεις παρουσιάζουν περιορισμένη ακρίβεια σε πιο εξειδικευμένα ή σύνθετα σενάρια, λόγω της πιθανής ελλιπής εκπαίδευσής τους σε παρόμοια δεδομένα, με αποτέλεσμα την παραγωγή ανεπαρκών αποτελεσμάτων. (Hurani & Idris, 2024)

Ακόμη, όπως αναφέρουν οι Hurani και Idris (2024), η χρήση κώδικα η οποία έχει παραχθεί από μεγάλα γλωσσικά μοντέλα, αυξάνει την ανάγκη για επαλήθευση και διορθώσεις από τους προγραμματιστές, καθώς το μοντέλο ενδέχεται να μην κατανοεί πλήρως τις απαιτήσεις ή να παράγει κάτι παραπλήσιο από αυτό που ζητήθηκε. Αυτό μπορεί να αυξήσει σημαντικά τον χρόνο διόρθωσης σφαλμάτων, σε σύγκριση με τη συγγραφή κώδικα από τους ίδιους τους προγραμματιστές.

Τέλος, από τον τρόπο κατασκευής τους, τα μεγάλα γλωσσικά μοντέλα εξαρτώνται άμεσα από το σύνολο των δεδομένων όπου έχουν εκπαιδευτεί, με αποτέλεσμα η ποιότητα και το περιεχόμενο των δεδομένων να επηρεάζουν σε μεγάλο βαθμό τα παραγόμενα αποτελέσματα.

## Μελλοντικές προοπτικές και ευκαιρίες ανάπτυξης

Τα μεγάλα γλωσσικά μοντέλα, ήδη ισχυρά εργαλεία στην ανάπτυξη λογισμικού, συνεχώς εξελίσσονται, με νέες και υποσχόμενες εφαρμογές και βελτιώσεις να μπορούν να επεκτείνουν περαιτέρω τις δυνατότητές τους. Με τις βελτιώσεις αυτές, στις οποίες θα γίνει αναφορά στην συνέχεια, αναμένεται ότι τα μοντέλα αυτά θα προσφέρουν ακόμη πιο προηγμένα εργαλεία και λειτουργικότητες για να κάνουν χρήση οι προγραμματιστές.

### Σύμπραξη μεγάλων γλωσσικών μοντέλων

Μία υποσχόμενη προοπτική η οποία θα άξιζε να ερευνηθεί σε μεγαλύτερο βαθμό, όπως αναφέρει και ο Hou και συν.(2023), είναι η ενσωμάτωση πολλαπλών διαφορετικών μεγάλων γλωσσικών μοντέλων σε ένα ενιαίο σύστημα, όπου τα μοντέλα θα συνεργάζονται για την επίλυση σύνθετων εργασιών στην ανάπτυξη λογισμικού. Αυτή η σύμπραξη θα αξιοποιούσε τα διαφορετικά πλεονεκτήματα κάθε μοντέλου, δημιουργώντας ένα εξαιρετικά αποτελεσματικό σύστημα.

### Κατανόηση διαφορετικού τύπου εισόδων

Επιπρόσθετα, μια σημαντική ευκαιρία για την ανάπτυξη των μεγάλων γλωσσικών μοντέλων στον τομέα της ανάπτυξης λογισμικού, είναι η βελτίωση της κατανόησης διαφορετικών τύπων εισόδου, όπως εικόνες, σχήματα και φωνή, καθώς και η αυτόματη παραγωγή πολύπλοκων διαγραμμάτων όπως UML. Παρόλο που πολυτροπικά μοντέλα, όπως το GPT-4, έχουν σημειώσει πρόοδο σε αυτούς τους τομείς, δεν έχουν ακόμα φτάσει σε ένα πλήρως υποσχόμενο επίπεδο ώστε να αποτελούν ένα έμπιστο εργαλείο για την υλοποίηση των παραπάνω(Hou και συν., 2023).

### Προσαρμογή στις ανάγκες των χρηστών

Επιπλέον, όπως αναφέρει και η OpenAI, μία πολύ σημαντική εξέλιξη αναμένεται να ενσωματωθεί στα ήδη μεγάλα γλωσσικά μοντέλα της, τα οποία θα διαθέτουν πιο προηγμένες ικανότητες μάθησης και προσαρμογής, επιτρέποντάς τους να μαθαίνουν από τις αλληλεπιδράσεις με τους χρήστες και να προσαρμόζονται στις συγκεκριμένες ανάγκες και προτιμήσεις τους. Αυτές οι δυνατότητες θα μπορούσαν να βελτιώσουν την ακρίβεια των προτάσεων κώδικα και να προσφέρουν εξατομικευμένες λύσεις που ανταποκρίνονται καλύτερα στις απαιτήσεις των έργων, αφού το μεγάλο γλωσσικό μοντέλο θα επικεντρώνεται σε συγκεκριμένους τομείς.

### Βελτίωση στις ήδη υπάρχουσες εργασίες

Ωστόσο, είναι εξίσου σημαντικό τα μεγάλα γλωσσικά μοντέλα να συνεχίσουν να βελτιώνονται και στις ήδη υπάρχουσες εργασίες που εκτελούν καλά, μέσω της εκπαίδευσής τους σε περισσότερα και ποιοτικότερα δεδομένα. Αυτό περιλαμβάνει την περαιτέρω βελτίωση της ικανότητάς τους να παράγουν κώδικα με βάση τις απαιτήσεις σε φυσική γλώσσα, την ακριβή πρόταση διορθώσεων και συμπληρώσεων και τη μείωση των σφαλμάτων κατά τη διάρκεια της ανάπτυξης. Η ενίσχυση σε αυτούς τους τομείς θα βελτιώσει τη συνολική αποδοτικότητα των εργαλείων αυτών και θα συμβάλει στην ακόμα καλύτερη ενσωμάτωσή τους και αποδοχή στη διαδικασία ανάπτυξης λογισμικού.

# BEHAVIOR-DRIVEN DEVELOPMENT (BDD)

Το κεφάλαιο αυτό εστιάζει στη μεθοδολογία του **Behavior Driven Development** (BDD), μια εξέλιξη του Test Driven Development (TDD) που στοχεύει στην προώθηση της συνεργασίας μεταξύ προγραμματιστών, ειδικών ελέγχου ποιότητας και πελατών. Η μεθοδολογία αυτή ενισχύει τη συνεργασία και την επικοινωνία μεταξύ των μελών μιας ομάδας ανάπτυξης λογισμικού, βελτιώνοντας έτσι την ποιότητα και τη σαφήνεια των απαιτήσεων και της λειτουργικότητας του λογισμικού.

## Test Driven Development και Behavior Driven Development

Ξεκινώντας το κεφάλαιο, θα γίνει μία συνοπτική αναφορά στην μεθοδολογία TDD και τις αρχές τις, έτσι ώστε στην συνέχεια να γίνει κατανοητή η έννοια του BDD που αποτελεί μία εξέλιξη αυτής.

Το **Test Driven Development (TDD)** είναι μία μεθοδολογία ανάπτυξης λογισμικού που δίνει έμφαση στην δημιουργία δοκιμών (tests) πριν αρχίσει η διαδικασία της κωδικοποίησης, κάτι το οποίο φαίνεται αντίθετο με τις παραδοσιακές πρακτικές, που τοποθετούν τις δοκιμές σαν το τελευταίο στάδιο στην διαδικασία ανάπτυξης λογισμικού.

Το TDD υποστηρίζει ότι οι δοκιμές αρχικά σχεδιάζονται με την δεδομένη συνθήκη ότι θα αποτύχουν, αφού δεν υπάρχει ακόμη ο απαραίτητος κώδικας για να περάσουν οι έλεγχοι. Στη συνέχεια, αναπτύσσεται ο κώδικας της νέας μονάδας ή λειτουργικότητας με την πιο απλή λύση ώστε απλά να περάσει το τεστ. Μετά την υλοποίηση, οι δοκιμές εκτελούνται ξανά για να επαληθευτεί ότι η υλοποίηση καλύπτει τις απαιτήσεις των δοκιμών και τέλος θα πρέπει να γίνουν οι απαραίτητες τροποποιήσεις για να βελτιωθεί η ποιότητα του υλοποιημένου ελέγχου της μονάδας (Test-driven development, 2024).

Εικόνα 7. Ο κύκλος ανάπτυξης του TDD

A diagram of a test

Description automatically generated

Σημείωση: Ανάκτηση από «Why Test-Driven Development (TDD)», MARNSER,([σύνδεσμος](https://marsner.com/blog/why-test-driven-development-tdd/))

Το κλειδί στο TDD είναι ότι η κάλυψη του κώδικα δεν είναι ο τελικός στόχος, αλλά το αποτέλεσμα της σωστής εφαρμογής της μεθοδολογίας TDD. Το TDD προάγει την ποιότητα του κώδικα, καθώς και τη συνεχή βελτίωση μέσα από επαναλαμβανόμενους κύκλους δοκιμών και υλοποίησης.

Η μεθοδολογία **Behavior Driven Development** (BDD) βασίζεται κυρίως στην ιδέα του TDD και αποτελεί μία εξέλιξη της, επηρεασμένη παράλληλα και από διάφορες άλλες agile μεθοδολογίες, η οποία έχει ως βασικό της στόχο την ενθάρρυνση της συνεργασίας μεταξύ όλων των ενδιαφερόμενων μερών στην διαδικασία της ανάπτυξης του λογισμικού ενός έργου. Συγκεκριμένα, όπως αναφέρει και η Fitzgibbons(2021), το BDD προωθεί την τεκμηρίωση και τον σχεδιασμό μίας εφαρμογής με βάσει της συμπεριφοράς που αναμένεται να βιώσει ο χρήστης, ενθαρρύνοντας τους προγραμματιστές να επικεντρώνονται στις σημαντικές συμπεριφορές που πρέπει να έχει το σύστημα. Έτσι, αποφεύγεται η δημιουργία περιττού κώδικα και υπερβολικών χαρακτηριστικών.

## Η σημασία της συνεργασίας στο BDD

Το BDD επιτυγχάνει τον στόχο της ενθάρρυνσης της συνεργασίας μεταξύ όλων των ενδιαφερόμενων μερών της ανάπτυξης ενός έργου, μέσω της χρήσης μίας κοινής γλώσσας, κατανοητής από όλους, η οποία αναφέρεται ως «**Ομοιογενής Γλώσσα**» (Ubiquitous Language). Χρησιμοποιώντας μία κοινή γλώσσα, βελτιώνεται η κατανόηση των απαιτήσεων του έργου και η ποιότητα του παραγόμενου λογισμικού, αφού όλοι οι εμπλεκόμενοι, ανεξαρτήτως της θέσης τους, μπορούν να συμμετέχουν και να έχουν λόγο στο τελικό αποτέλεσμα.

### Ομοιογενής γλώσσα (Ubiquitous Language)

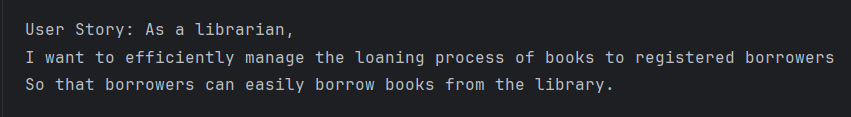
Το κεντρικό σημείο της συνεργασίας μεταξύ όλων των ενδιαφερόμενων μερών είναι η χρήση μίας γλώσσας κατανοητής από όλα τα εμπλεκόμενα δια τμηματικά μέλη των ομάδων. Αυτή η γλώσσα ονομάζεται **Ομοιογενής Γλώσσα** και χρησιμοποιείται σε κάθε επικοινωνία σχετικά με το λογισμικό. Ένα παράδειγμα μίας τέτοιας γλώσσας, που χρησιμοποιείται στην πράξη στο εργαλείο Cucumber και θα γίνει αναλυτική αναφορά παρακάτω, είναι η **Gherkin**. Η Gherkin είναι μία δομημένη γλώσσα με την οποία δημιουργούνται οι έλεγχοι στο BDD και χρησιμοποιεί μία συγκεκριμένη σύνταξη, εύκολη στην ανάγνωση από όλους και γραμμένη σε φυσική γλώσσα (Cucumber, n.d.).

### Σενάρια και ιστορίες χρηστών στο BDD

Η πρώτη διαδικασία με την οποία αρχίζει κάθε έργο που αναπτύσσεται με την μεθοδολογία BDD, είναι ο ορισμός της συμπεριφοράς του συστήματος από την οπτική των τελικών χρηστών. Αυτό περιλαμβάνει την δημιουργία **ιστοριών χρηστών** και **σεναρίων** που περιγράφουν πώς θα πρέπει να συμπεριφέρεται το σύστημα σε διάφορες καταστάσεις που μπορεί να έρθει αντιμέτωπο κατά την λειτουργία του.

Οι **ιστορίες χρηστών** γράφονται σε φυσική γλώσσα και είναι της μορφής «Ως [ρόλος του χρήστη], Θέλω να [κάποια ενέργεια του χρήστη με το σύστημα], Ώστε να [κάποιο τελικό αποτέλεσμα, στόχος ή πλεονέκτημα]». Ένα παράδειγμα είναι αυτό της **Εικόνας 8**, όπου παρουσιάζεται μία υποθετική ιστορία χρήστη σε ένα σύστημα βιβλιοθήκης, όπου ο χρήστης ή βιβλιοθηκάριος στην συγκεκριμένη περίπτωση, θέλει μέσω του συστήματος να μπορεί να χειρίζεται αποδοτικά την διαδικασία δανεισμού ώστε οι δανειζόμενοι να μπορούν να δανείζονται εύκολα βιβλία από την βιβλιοθήκη. (Cucumber, n.d.)

Εικόνα 8. Παράδειγμα ιστορίας χρήστη



Πολλά **σενάρια** μπορούν να ανήκουν σε ένα κοινό χαρακτηριστικό του συστήματος, το ονομαζόμενο «**feature**». Για παράδειγμα, για το χαρακτηριστικό/feature "Εγγραφή χρήστη", μπορούν να υπάρξουν σενάρια που εξετάζουν τι γίνεται στην περίπτωση της επιτυχούς εγγραφής, στην περίπτωση που ο χρήστης έχει ήδη λογαριασμό και πολλά άλλα.

Τα σενάρια γράφονται με την ομοιογενή γλώσσα **Gherkin**, όπως αναφέρθηκε παραπάνω, η οποία έχει την μορφή «**Given-When-Then**»(συνήθως ακολουθείται αυτή η σειρά) και στην οποία θα γίνει αναφορά παρακάτω.

Τα σενάρια που γράφονται με αυτή τη δομή μπορούν εύκολα να είναι κατανοητά από όλους τους ενδιαφερόμενους και βοηθούν στην εξασφάλιση ότι οι απαιτήσεις των χρηστών ικανοποιούνται πλήρως από το σύστημα (Cucumber, n.d.).

## Οι τρείς πρακτικές του BDD

Όπως αναφέρεται και στο documentation της επίσημης σελίδας του Cucumber, το BDD, όπως και το TDD, ενθαρρύνει την εργασία σε γρήγορες επαναλήψεις(rapid iterations), διασπώντας τα προβλήματα των χρηστών σε μικρά κομμάτια ώστε να μπορούν να επιλυθούν γρήγορα και ορθά.

Ουσιαστικά, η καθημερινότητα ενός έργου που αναπτύσσεται με BDD αποτελείται από τρείς διαδικασίες:

1. **Ανάλυση παραδειγμάτων** : Ξεκινά με μια μικρή επερχόμενη αλλαγή, μια ιστορία χρήστη, όπως αναλύθηκε και πριν, και την αναλύει σε πραγματικά παραδείγματα (σενάρια).
2. **Αυτοματοποίηση παραδειγμάτων** : Τροποποιεί τα παραδείγματα ώστε να μπορούν να αυτοματοποιηθούν ως σενάρια Gherkin.
3. **Υλοποίηση κώδικα** : Υλοποιεί τον κώδικα του συστήματος που περιγράφεται από αυτά τα παραδείγματα.

Εικόνα 9. Οι τρείς πρακτικές του BDD

A diagram of software development

Description automatically generated

Σημείωση: Ανάκτηση από «Behavior-Driven Development», Cucumber ([σύνδεσμος](https://cucumber.io/docs/bdd/))

## Τεχνικές Ανάλυσης Συνεργασίας

Ένα κομβικό σημείο στην εύρυθμη λειτουργία της μεθοδολογίας BDD, είναι μία συνάντηση η οποία ονομάζεται «Three Amigos Meeting», όπου όπως θα αναλυθεί και παρακάτω, αποτελεί μία «συζήτηση» μεταξύ των διαφορετικών ενδιαφερομένων για την κατανόηση και δημιουργία των απαιτήσεων ή σεναρίων.

Η συνάντηση των τριών φίλων, ή όπως είναι γνωστή «**Three Amigos Meeting**», έχει ως βασικό σκοπό της την μετατροπή των ιστοριών χρηστών σε κατανοητά και ολοκληρωμένα από όλους σενάρια Gherkin. Περιέχει τουλάχιστον τρία διαφορετικά μέλη :

1. **Ιδιοκτήτης του προϊόντος (Product Owner)** - Εστιάζει στο εύρος της εφαρμογής και μετατρέπει τις ιστορίες χρηστών σε features , αποφασίζοντας τι είναι εντός και εκτός του εύρους.
2. **Δοκιμαστής (Tester)** - Δημιουργεί σενάρια και ακραίες περιπτώσεις, εξετάζοντας ποιες ιστορίες χρηστών δεν έχουν καλυφθεί.
3. **Προγραμματιστής (Developer)** - Προσθέτει βήματα στα σενάρια και εξετάζει τις λεπτομέρειες της εφαρμογής, όπως την εκτέλεση και τα πιθανά εμπόδια κατά την δημιουργία του κώδικα.

Οι παραπάνω συζητήσεις δημιουργούν πολύ καλούς ελέγχους, διότι ο το κάθε άτομο βλέπει το σύστημα από διαφορετική οπτική γωνία. Για τον λόγο αυτό, κρίνεται αναγκαίο όλα τα παραπάνω άτομα να παίρνουν μέρος στην συνάντηση, με σκοπό την ανακάλυψη νέων παραδειγμάτων χρήσης. (Cucumber, n.d.)

Εικόνα 10. The Three Amigos Meeting

A group of people sitting at a table with laptops

Description automatically generated

Σημείωση: Ανακτήθηκε από «The 3 Amigos Meeting», AWH, 2020, Medium. ([σύνδεσμος](https://awhnet.medium.com/the-3-amigos-meeting-one-meeting-is-all-you-need-8af8b4a58ba2))

## Αυτοματισμός Δοκιμών με το Cucumber

### Εισαγωγή στο Cucumber

Το **Cucumber** είναι ένα εργαλείο λογισμικού το οποίο υποστηρίζει την μεθοδολογία του Behavior Driven Development βασιζόμενο στην χρήση της γλώσσας **Gherkin**, για την οποία έγινε μία μικρή εισαγωγή παραπάνω. Η Gherkin επιτρέπει την περιγραφή των απαιτήσεων του συστήματος και την αυτοματοποίηση των ελέγχων. Συγκεκριμένα, το Cucumber έχει έναν βοηθητικό ρόλο ως προς τον αυτοματισμό των ελέγχων ενός συστήματος, προσφέροντας μία κοινή γλώσσα που είναι κατανοητή από όλους τους ενδιαφερόμενους.

### Η σύνταξη Gherkin

H Gherkin είναι μία δομημένη γλώσσα που χρησιμοποιείται από το εργαλείο Cucumber για τη δημιουργία ελέγχων στο BDD. Η σύνταξή της είναι απλή και κατανοητή, επιτρέποντας σε τεχνικούς και μη τεχνικούς χρήστες να μπορούν να γράφουν σενάρια.

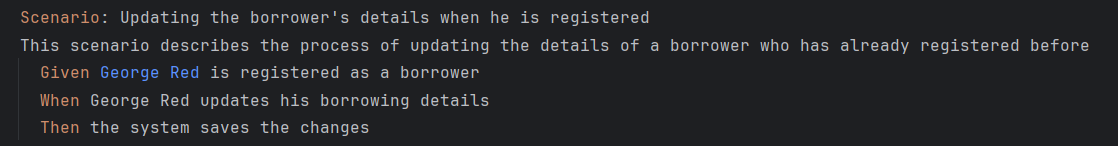
Η γλώσσα Gherkin χρησιμοποιεί την μορφή «Given-Then-When» και με αυτό τον τρόπο επιτρέπει την εύκολη γραφή σεναρίων κατανοητών από όλα τα μέλη.

Οι λέξεις αυτές κλειδιά αποτελούν τον βασικό τρόπο γραφής των σεναρίων ενός feature και κάθε μία από αυτές έχει μία διαφορετική σημασία, όπως αναφέρεται και από το documentation της επίσημης σελίδας του Cucumber :

1. Η λέξη κλειδί **Given**, ορίζει το γενικό πλαίσιο του σεναρίου και βάζει το σύστημα σε μία συγκεκριμένη κατάσταση ώστε να ξεδιπλωθεί το σενάριο στα επόμενα βήματα. Αυτό μπορεί να περιλαμβάνει την προετοιμασία του συστήματος, τη δημιουργία δεδομένων ή την εξασφάλιση ότι το σύστημα βρίσκεται σε μια συγκεκριμένη αρχική κατάσταση.
2. Η λέξη κλειδί **When**, είναι μια ενέργεια, η οποία μεταβάλλει το σύστημα από την κατάσταση «ηρεμίας» του και το αναγκάζει να προβεί σε κάποιο αποτέλεσμα. Αυτή η ενέργεια είναι συνήθως η κεντρική πράξη του σεναρίου και αποτελεί το γεγονός που θα προκαλέσει την αντίδραση του συστήματος.
3. Η λέξη κλειδί **Then**, είναι το αποτέλεσμα, δηλαδή η ενέργεια που αναμένουμε να κάνει το σύστημα όταν γίνεται η ενέργεια στο βήμα When. Αυτό το τμήμα του σεναρίου περιγράφει την αναμενόμενη συμπεριφορά του συστήματος και επιτρέπει την επαλήθευση ότι το σύστημα λειτουργεί όπως πρέπει.

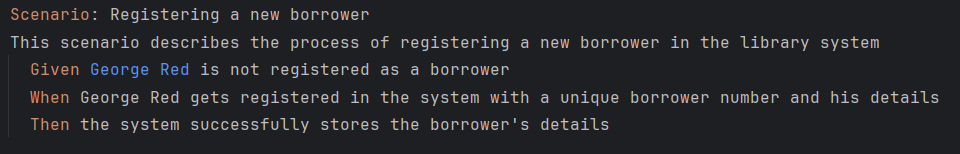
Στην **Εικόνα 11** παρακάτω, παρουσιάζεται ένα παράδειγμα ενός σεναρίου γραμμένου σε Gherkin στο Cucumber, το οποίο αφορά την ενημέρωση των στοιχείων ενός δανειζόμενου σε ένα υποθετικό σύστημα βιβλιοθήκης και το οποίο ανήκει σε ένα feature το οποίο αφορά την «Διαχείριση Δανειζόμενων». Όπως φαίνεται, κάθε σενάριο περιέχει επίσης έναν σύντομο τίτλο ο οποίος περιγράφει την περίπτωση χρήσης που πραγματεύεται το σενάριο που ακολουθεί.

Εικόνα 11. Παράδειγμα σεναρίου με την μορφή Given-When-Then



Στην **Εικόνα 12** παρουσιάζεται ένα ακόμη παράδειγμα ενός υποθετικού σεναρίου στο ίδιο σύστημα με την Εικόνα 11, το οποίο αφορά την περίπτωση της επιτυχούς εγγραφής ενός δανειζόμενου σε ένα σύστημα βιβλιοθήκης.

Εικόνα 12. Παράδειγμα σεναρίου με την μορφή Given-When-Then



### Δημιουργία αρχείων χαρακτηριστικών (Feature Files)

Τα **αρχεία χαρακτηριστικών** (feature files), όπως αναφέρθηκε και παραπάνω, είναι αρχεία που περιέχουν τα σενάρια Gherkin που περιγράφουν τις διάφορες περιπτώσεις χρήσης ενός συγκεκριμένου feature του συστήματος. Η δομή των feature files φαίνεται στην **Εικόνα 13** και αποτελείται από την λέξη κλειδί «feature» που αρχικοποιεί το αρχείο. Στην συνέχεια, ακολουθεί το όνομα του feature, το οποίο θα είναι και το όνομα του αρχείου και μία μικρή περιγραφή του feature και των σεναρίων που θα ακολουθήσουν. Ακόμη, μία καλή πρακτική αποτελεί η προσθήκη μίας ιστορία χρήστη, για να γίνει καλύτερα κατανοητό το σύνολο των σεναρίων που θα ακολουθήσουν.

Εικόνα 13. Παράδειγμα δομής ενός feature file

A screenshot of a computer program

Description automatically generated

Μια καλή πρακτική είναι τα feature files να περιέχουν το πολύ δέκα σενάρια και να εστιάζουν σε ένα συγκεκριμένο χαρακτηριστικό.

### Ανάπτυξη των Step Definitions

Τα **Step Definitions** είναι τμήματα κώδικα που συνδέουν τα σενάρια γραμμένα σε Gherkin με την πραγματική υλοποίηση κώδικα ώστε να περάσουν οι έλεγχοι και να δοκιμαστεί το σύστημα. Τα Step Definitions μπορούν να γραφτούν σε διάφορες γλώσσες προγραμματισμού όπως Java, Ruby κ.ά., και προσδιορίζουν την συμπεριφορά κάθε βήματος ενός σεναρίου Gherkin.

Το Cucumber εκτελεί τα σενάρια που βρίσκονται στα feature files τρέχοντας τον αντίστοιχο κώδικα των Step Definitions, διασφαλίζοντας ότι τα βήματα εκτελούνται με σωστή σειρά, προσομοιάζοντας την συμπεριφορά που περιγράφεται στα σενάρια Gherkin.

Η δομή των Step Definitions, όπως φαίνεται και στην **Εικόνα 14**, περιλαμβάνει βοηθητικές δομές δεδομένων και αντικείμενα που χρησιμοποιούνται στον κώδικα των Step Definitions. Για να συνδέσουμε τα βήματα ενός σεναρίου με ένα Step Definition, χρησιμοποιούμε τη μορφή @Given, @When, ή @Then ακολουθούμενη από το κείμενο του βήματος στο σενάριο Gherkin.

Εικόνα 14. Παράδειγμα δομής ενός αρχείου Step Definitions

A computer screen shot of a program

Description automatically generated

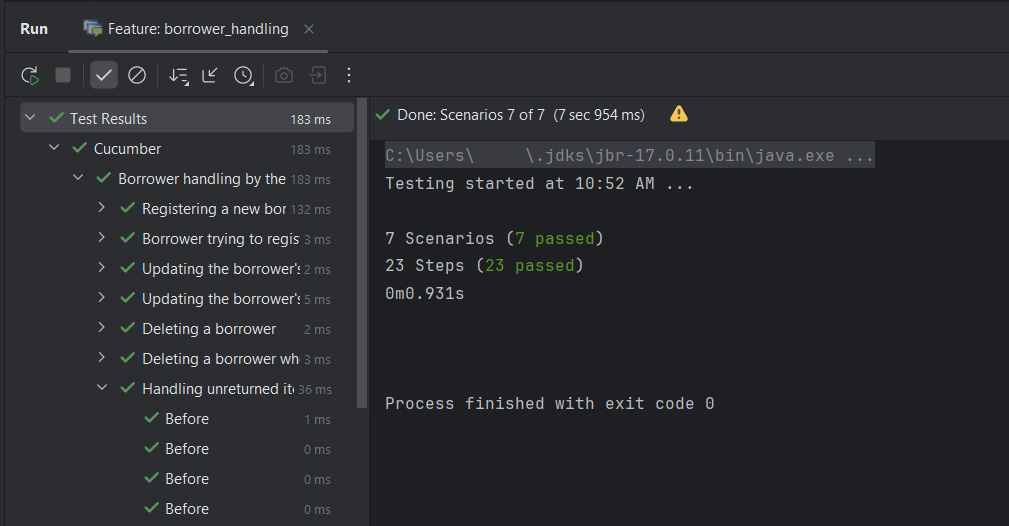
Στην περίπτωση της **Εικόνας 14,** το πρώτο βήμα του Step Definition αρχικοποιεί το περιβάλλον και τα αντικείμενα που θα χρησιμοποιηθούν στα παρακάτω βήματα του σεναρίου, σε ένα υποθετικό σύστημα μίας εφαρμογής που παραδίδει μηνύματα σε άτομα μίας συγκεκριμένης απόστασης.

Τα Step Definitions πρέπει να είναι μοναδικά ανεξαρτήτως feature file ώστε κάθε βήμα ενός σεναρίου να μπορεί να αντιστοιχιστεί με ένα μόνο Step Definition, αποφεύγοντας την σύγχυση και διασφαλίζοντας την σωστή εκτέλεση των ελέγχων.

## Αναφορές του Cucumber

Μετά την εκτέλεση των βημάτων των σεναρίων των Step Definitions, το Cucumber παράγει **λεπτομερείς αναφορές** και σχόλια για την εκτέλεση που ακολούθησε. Αυτές οι αναφορές περιλαμβάνουν πληροφορίες για τα βήματα που πέρασαν επιτυχώς, τα βήματα που απέτυχαν και εκείνα που παραλήφθηκαν λόγω κάποιου σφάλματος. Οι αναφορές αυτές αποτελούν σημαντικό εργαλείο για προγραμματιστές και τους ειδικούς ελέγχου, καθώς τους βοηθούν να κατανοήσουν την κατάσταση του συστήματος και να εντοπίσουν τυχόν προβλήματα. (Cucumber, n.d.)

Εικόνα 15. Παράδειγμα αναφοράς με το Cucumber



## Ζωντανή τεκμηρίωση

Όταν χρησιμοποιείται το BDD ως μεθοδολογία ανάπτυξης ενός συστήματος, γράφονται παραδείγματα ή σενάρια τα οποία καθοδηγούν την ανάπτυξη. Αυτά τα παραδείγματα χρησιμεύουν επίσης ως κριτήρια αποδοχής (acceptance criteria) και είναι γραμμένα σε μια κοινή γλώσσα, ώστε να είναι κατανοητά από όλους τους εμπλεκόμενους. Επομένως, τα παραδείγματα ή σενάρια αυτά λειτουργούν και ως μία «**ζωντανή» τεκμηρίωση** του συστήματος. Σε αντίθεση με τα παραδοσιακά έγγραφα κριτηρίων αποδοχής, αυτή η τεκμηρίωση ενημερώνεται συνεχώς με κάθε νέα αλλαγή ή προσθήκη λειτουργικότητας στο σύστημα.

Η σημασία της ζωντανής τεκμηρίωσης είναι ιδιαίτερα σημαντική σε ένα περιβάλλον ανάπτυξης λογισμικού, καθώς επιτρέπει σε όλα τα μέλη της ομάδας να κατανοούν τις απαιτήσεις του συστήματος. Αυτό είναι ιδιαίτερα χρήσιμο για νέα μέλη της ομάδας, που δεν γνωρίζουν από πριν το σύστημα που αναπτύσσεται. Η ζωντανή τεκμηρίωση διασφαλίζει ότι η γνώση του συστήματος διατηρείται και διαμοιράζεται αποτελεσματικά σε όλη την ομάδα (Cucumber, n.d.).

## Καλές πρακτικές για γραφή με Cucumber

Η χρήση του Cucumber για τον αυτοματισμό των ελέγχων και την υιοθέτηση της μεθοδολογίας BDD μπορεί να συμβάλλει σημαντικά στην βελτίωση της ποιότητας του παραγόμενου λογισμικού. Ωστόσο, η επίτευξη αυτού του στόχου εξαρτάται σε μεγάλο βαθμό από τον τρόπο που θα γίνει η γραφή των σεναρίων και η οργάνωσή τους σε feature files. Μερικές καλές πρακτικές για την συγγραφή σεναρίων με Cucumber, όπως αναφέρει και ο Pathak (2022), είναι οι εξής:

1. **Μικρά σενάρια** : Τα σενάρια που γράφονται για ένα feature και τα οποία περιγράφουν μία περίπτωση χρήσης του συστήματος, θα πρέπει να είναι μικρά και να εστιάζουν σε μία μόνο λειτουργία του συστήματος.
2. **Χρήση παραδειγμάτων από τον πραγματικό κόσμο** : Τα σενάρια θα πρέπει να βασίζονται σε πραγματικά παραδείγματα ώστε να διασφαλίζουν ότι είναι ρεαλιστικά και ότι το σύστημα θα λειτουργήσει σωστά σε πραγματικές συνθήκες
3. **Επαναχρησιμοποίηση βημάτων** : Όταν είναι δυνατόν, θα πρέπει να γίνεται επαναχρησιμοποίηση των βημάτων των σεναρίων ώστε να μειωθεί η επανάληψη και να διευκολυνθεί η συντήρηση των σεναρίων.
4. **Οργάνωση των feature files** : Τα feature files θα πρέπει να οργανώνονται με τρόπο που να διευκολύνει την εύρεση και κατανόηση των σεναρίων. Κάθε αρχείο χαρακτηριστικών θα πρέπει να περιέχει σενάρια που σχετίζονται με ένα συγκεκριμένο χαρακτηριστικό ή λειτουργία του συστήματος, χρησιμοποιώντας κατανοητούς τίτλους και περιγραφές.

# ΧΡΗΣΗ ΤΩΝ ΜΕΓΑΛΩΝ ΓΛΩΣΣΙΚΩΝ ΜΟΝΤΕΛΩΝ ΓΙΑ BDD

Το παρόν κεφάλαιο αναλύει την μεθοδολογία, τα κριτήρια αξιολόγησης αλλά και την εμπειρική αξιολόγηση του πειράματος που πραγματοποιήθηκε στο πλαίσιο της εργασίας.

## Μεθοδολογία

Στην παρούσα εργασία, ακολουθήθηκε μια συστηματική προσέγγιση για την αξιολόγηση της απόδοσης διαφορετικών μεγάλων γλωσσικών μοντέλων σε ένα συγκεκριμένο πλαίσιο χρήσης, βασισμένο στο πρόβλημα της διαχείρισης βιβλιοθήκης.

### Το πρόβλημα της βιβλιοθήκης

Ως αρχική αναφορά για το πείραμα που πραγματεύεται η εργασία, επιλέχθηκε το πρόβλημα της διαχείρισης μίας βιβλιοθήκης. Το πρόβλημα αυτό της διαχείρισης της βιβλιοθήκης, αποτελεί μια εφαρμογή η οποία αποτελείται από μία σειρά τυπικών use cases (περιπτώσεων χρήσης) που καλύπτουν τις βασικές λειτουργίες μίας βιβλιοθήκης, όπως για παράδειγμα:

1. **H αναζήτηση αντιγράφων βιβλίων** : Εύρεση αντιγράφων βιβλίων στην συλλογή της βιβλιοθήκης με βάση διάφορα κριτήρια.
2. **Η επιστροφή αντιγράφων βιβλίων** : Διαχείριση της διαδικασίας επιστροφής βιβλίων από τους δανειζόμενους και λειτουργίες σε περίπτωση καθυστέρησης.
3. **Η διαχείριση δανειζόμενων** : Εγγραφή νέου δανειζόμενου, ενημέρωση των στοιχείων του, διαγραφή των στοιχείων του.

Το συγκεκριμένο πρόβλημα επιλέχθηκε ως αναφορά για την εργασία διότι αποτελεί ένα πλήρως τεκμηριωμένο, απλό και σαφές πρόβλημα, το οποίο δημιουργήθηκε από τον επιβλέποντα δάσκαλο της εργασίας, τον κύριο Διαμαντίδη, και με το οποίο μπορούν να δημιουργηθούν με ευκολία σενάρια δοκιμών και αυτοματοποιημένος κώδικας με χρήση της μεθοδολογίας BDD και του εργαλείου Cucumber, για να γίνει στην συνέχεια η σύγκριση με τα αποτελέσματα των μεγάλων γλωσσικών μοντέλων ([σύνδεσμος του συστήματος της βιβλιοθήκης στο GitHub](https://github.com/diamantidakos/Library)).

### Μετατροπή των use cases σε σενάρια Cucumber

Αφού επιλέχθηκε το πρόβλημα της διαχείρισης της βιβλιοθήκης ως αναφορά, το επόμενο βήμα για την βαθιά κατανόηση του συστήματος της βιβλιοθήκης αλλά και της μεθοδολογίας του BDD και της χρήσης του εργαλείου Cucumber, ήταν η μετατροπή των use cases της βιβλιοθήκης σε σενάρια γραμμένα με την γλώσσα Gherkin. Αυτή η διαδικασία επέτρεψε την αυτοματοποίηση των σεναρίων με το εργαλείο Cucumber και την δημιουργία των BDD tests, παρέχοντας ένα μέτρο σύγκρισης για τα αποτελέσματα των γλωσσικών μοντέλων. Για κάθε use case του συστήματος της βιβλιοθήκης, το οποίο ήταν εντός του εύρους της εφαρμογής, δημιουργήθηκε το αντίστοιχο feature αποτελούμενο από τα αντίστοιχα σενάρια σε Gherkin, που περιγράφουν την συμπεριφορά του συστήματος με βάση πραγματικά παραδείγματα. Στην **Εικόνα 16** παρουσιάζεται ένα παράδειγμα ενός use case του συστήματος της βιβλιοθήκης, που περιγράφει την βασική ροή της διαδικασίας του δανεισμού ενός αντιτύπου σε έναν δανειζόμενο αλλά και όλες τις εναλλακτικές ροές που αυτό περιέχει.

Εικόνα 16. Παράδειγμα του use case δανεισμού αντιτύπων του συστήματος της βιβλιοθήκης

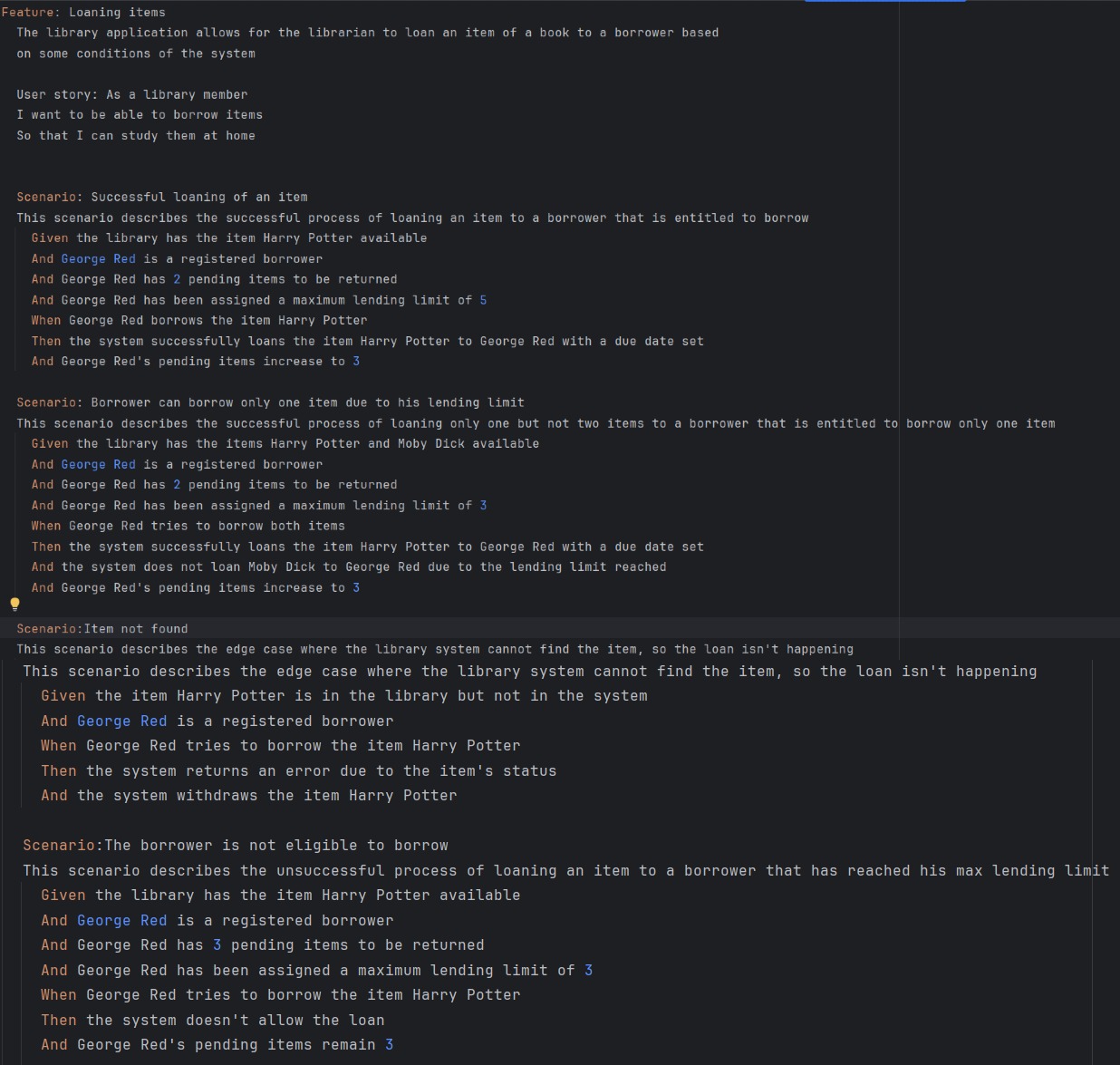
A close-up of a document

Description automatically generated

Σημείωση: Ανάκτηση από «Τεχνολογία Λογισμικού Β’ΕΚΔΟΣΗ», των Ε.Α.Γιακουμάκη και Ν.Α. Διαμαντίδη, σελ. 127-128 (2021)

Στην συνέχεια, το συγκεκριμένο use case μετατράπηκε σε feature με σενάρια που περιγράφουν τις περιπτώσεις χρήσης. Όπως φαίνεται και στην **Εικόνα 17**, το feature «Loaning Items» αποτελείται από διάφορα σενάρια που χρησιμοποιούν χαρακτήρες ή «περσόνες» όπως αναφέρονται σε πολλά άρθρα, για την αληθοφανή παρουσίαση του παραδείγματος και της καλύτερης κατανόησης από τους αναγνώστες.

Εικόνα 17. Παράδειγμα σεναρίων Cucumber για το use case του δανεισμού αντιτύπων



Συγκεκριμένα, δημιουργήθηκαν σενάρια για την περίπτωση του επιτυχή δανεισμού των αντιτύπων, για την περίπτωση της ύπαρξης ορίου για τον δανεισμό από τον δανειζόμενο, για την μη-ύπαρξη ενός αντιτύπου αλλά και σενάρια για την περίπτωση που ο δανειζόμενος δεν πληρεί τα κριτήρια δανεισμού.

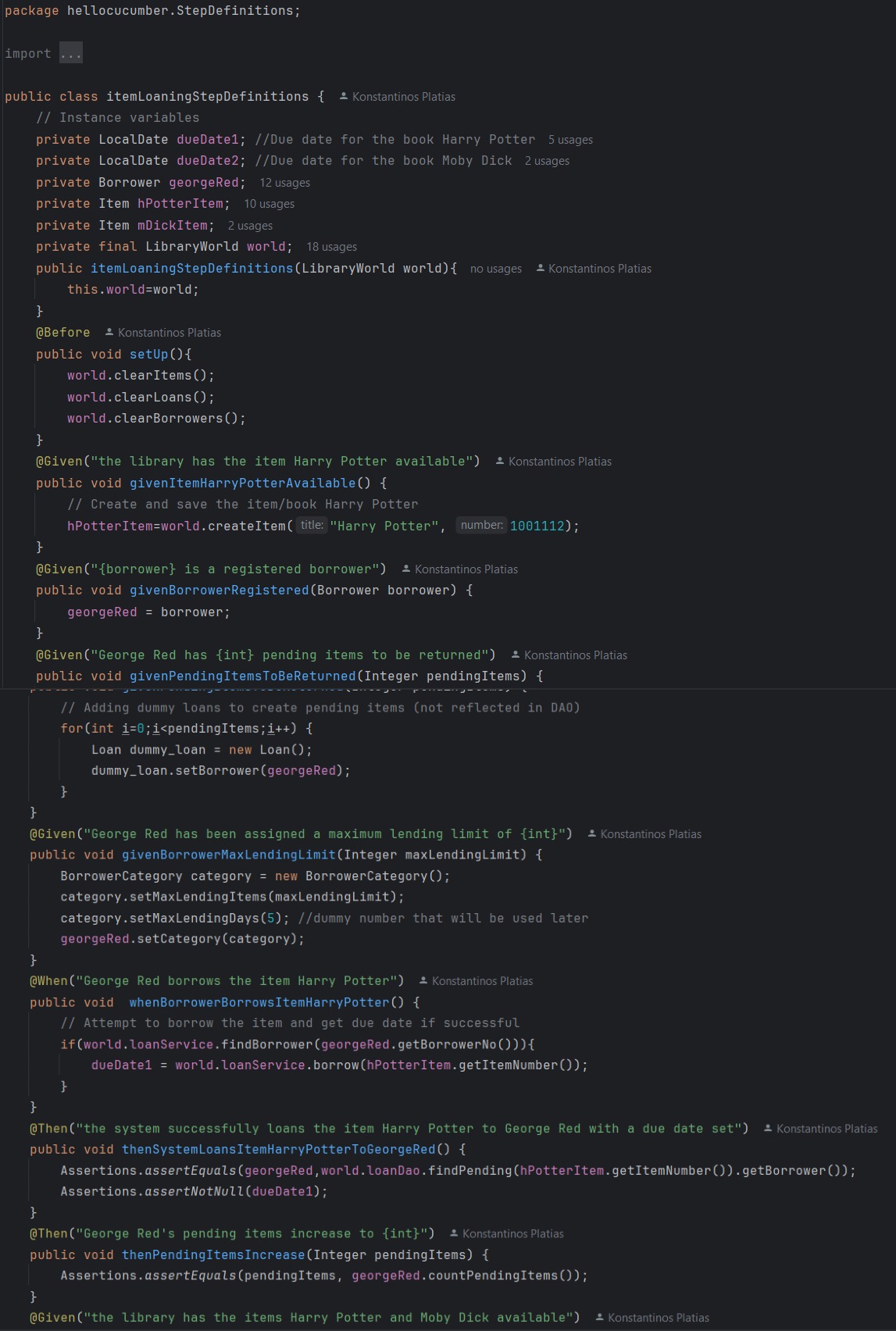
### Ανάπτυξη BDD tests

Η επόμενη φάση μετά την συγγραφή των σεναρίων Cucumber για τα use cases του προβλήματος της βιβλιοθήκης, ήταν η δημιουργία των αυτοματοποιημένων τεστ, γνωστών και ως Step Definitions, τα οποία χρησιμοποιήθηκαν από το Cucumber για την επαλήθευση των σεναρίων.

Για κάθε σενάριο δημιουργήθηκαν αντίστοιχες κλάσεις και μέθοδοι στην γλώσσα προγραμματισμού Java, που έλεγχαν την λειτουργία του συγκεκριμένου βήματος του σεναρίου με το πραγματικό σύστημα βιβλιοθήκης.

Κατά την συγγραφή του κώδικα των Step Definitions, έγινε χρήση αντικειμένων Data Access Objects με σκοπό την προσομοίωση μίας πραγματικής βάσης δεδομένων, αποθηκεύοντας δεδομένα και μεταβλητές προερχόμενα από τα τεστ. Επίσης, χρησιμοποιήθηκε η τεχνική «Dependency Injection» για τη μείωση της πολυπλοκότητας του κώδικα και την αύξηση της επαναχρησιμοποίησής του, η οποία ουσιαστικά δημιουργεί μία κοινή κλάση όπου περιέχει όλες τις βοηθητικές μεθόδους για να χρησιμοποιεί κάθε διαφορετικό feature file. Ωστόσο, η τεχνική αυτή δεν ζητήθηκε από τα μεγάλα γλωσσικά μοντέλα, καθώς αύξανε την πολυπλοκότητα χωρίς να προσφέρει πρόσθετη πληροφορία. Στην **Εικόνα 18** φαίνεται ο κώδικας των Step Definitions που γράφτηκε για το feature «Loaning Items» όπου αναφέρθηκε παραπάνω.

Εικόνα 18. Παράδειγμα κώδικα σε Java για τον έλεγχο των σεναρίων



A screenshot of a computer screen

Description automatically generated

### Εκτέλεση και επαλήθευση των BDD Tests

Μετά την ανάπτυξη των BDD tests, τα σενάρια εκτελέστηκαν για να επαληθευτεί η ορθότητα των σεναρίων σε Gherkin. Η εκτέλεση των BDD tests περιλάμβανε την αυτόματη εκτέλεση των σεναρίων από το εργαλείο Cucumber, το οποίο αυτοματοποιεί αυτή την διαδικασία και παράγει αναφορές σχετικά με τα αποτελέσματα των ελέγχων που έγιναν, όπως φαίνεται στην **Εικόνα 15** του προηγουμένου κεφαλαίου.

Τα BDD tests/Step Definitions επαλήθευσαν αν το σύστημα ανταποκρινόταν σωστά στις δεδομένες συνθήκες και αν τα αποτελέσματα συμφωνούσαν με τις αναμενόμενες συμπεριφορές που περιγράφονται στα σενάρια.

Συνοψίζοντας, η χρήση του συστήματος της βιβλιοθήκης, η μετατροπή των use cases σε σενάρια Cucumber και η ανάπτυξη των BDD tests αποτέλεσε ένα κρίσιμο βήμα στη μεθοδολογία της εργασίας. Η διαδικασία αυτή επέτρεψε τη συστηματική αξιολόγηση της ικανότητας των μεγάλων γλωσσικών μοντέλων να εκτελούν τις προβλεπόμενες λειτουργίες της βιβλιοθήκης και έθεσε τις βάσεις για τη σύγκριση των διαφορετικών μοντέλων με αντικειμενικά κριτήρια.

### Επιλογή συγκεκριμένων μεγάλων γλωσσικών μοντέλων

Για την υλοποίηση του πειράματος, επιλέχθηκαν συγκεκριμένα μεγάλα γλωσσικά μοντέλα που διακρίνονται για την ταχύτητα και την ικανότητά τους να παράγουν ποιοτικά αποτελέσματα. Η επιλογή αυτών των μεγάλων γλωσσικών μοντέλων κρίθηκε απαραίτητη για την εξαγωγή αξιόπιστων συμπερασμάτων σχετικά με την απόδοσή τους στην παραγωγή αυτοματοποιημένου κώδικα.

Το πρώτο μοντέλο που χρησιμοποιήθηκε ήταν το **GitHub Copilot** της Microsoft, το οποίο βασίζεται στο μοντέλο GPT-3.5 Turbo. Το GitHub Copilot χρησιμοποιείται ειδικά μόνο για την παραγωγή κώδικα και λειτουργεί ως ένας παράλληλος βοηθός για την υποβοήθηση και συμπλήρωση των προγραμματιστών. Η εξειδίκευσή του στην παραγωγή κώδικα το διαφοροποιεί από τα υπόλοιπα μοντέλα που χρησιμοποιήθηκαν στην εργασία, καθιστώντας το ένα εξαιρετικά χρήσιμο εργαλείο

Στην συνέχεια εξετάστηκαν όλα τα μεγάλα γλωσσικά μοντέλα της OpenAI, δηλαδή το **Chat GPT-3.5**, το **Chat GPT-4** και το **Chat GPT-4o (omni).** Τα μοντέλα αυτά κατατάσσονται κατά αύξουσα ισχύ και επίπεδο πρόσβασης, από το λιγότερο ισχυρό στο πιο ισχυρό. Η OpenAI, ως πρωτοπόρος στη δημιουργία μεγάλων γλωσσικών μοντέλων, έχει αναπτύξει τα πιο δημοφιλή μοντέλα της αγοράς έως σήμερα. Η επιλογή των μοντέλων της OpenAI στην εργασία βασίζεται στη γενική αναγνώριση της και την δύναμη που κατέχουν τα μοντέλα της, κάτι που δικαιολογεί την χρήση όλων τους στην αξιολόγηση.

Συνολικά, η επιλογή των παραπάνω μεγάλων γλωσσικών μοντέλων παρέχει μια αντικειμενική αξιολόγηση της ικανότητάς τους να υποστηρίξουν την ανάπτυξη λογισμικού και συγκεκριμένα το έργο της παραγωγής αυτοματοποιημένου κώδικα με περιορισμένες πληροφορίες.

### Σχεδιασμός διαφορετικών συνομιλιών βάσει προϋπάρχουσας γνώσης

Ένα κρίσιμο βήμα στην διεξαγωγή του πειράματος και στην αξιολόγηση του αυτοματοποιημένου κώδικα τεστ που παρήγαγαν τα μεγάλα γλωσσικά μοντέλα που αναφέρθηκαν παραπάνω, ήταν ο σχεδιασμός και η διαμόρφωση διαφορετικών συνομιλιών με βάση την προϋπάρχουσα γνώση που δινόταν στο σύστημα κάθε φορά. Στο πλαίσιο της εργασίας, δημιουργήθηκαν ποικίλες συνομιλίες με κάθε ένα από τα μεγάλα γλωσσικά μοντέλα για να δοκιμαστεί η απόδοσή τους σε διαφορετικά επίπεδα προϋπάρχουσας γνώσης.

Η αξιολόγηση χωρίστηκε σε τέσσερις φάσεις, με κάθε φάση να προσθέτει νέες πληροφορίες στις προηγούμενες, διαφοροποιώντας έτσι την ποσότητα και την ποιότητα της γνώσης που διατίθεται στο μοντέλο σε κάθε στάδιο. Συγκεκριμένα :

1. **Πρώτη φάση (Phase 1)** : Σε αυτή τη φάση, το μεγάλο γλωσσικό μοντέλο έλαβε περιορισμένη προϋπάρχουσα γνώση. Συγκεκριμένα, δόθηκε μια βασική περιγραφή της αρχιτεκτονικής του συστήματος της βιβλιοθήκης, γενικές πληροφορίες για τις απαντήσεις που αναμένονται και όλα τα σενάρια των διάφορων λειτουργιών γραμμένα σε Gherkin μέσω του εργαλείου Cucumber.
2. **Δεύτερη φάση (Phase 2)** : Στη δεύτερη φάση, η γνώση που δόθηκε ήταν η ίδια με την προηγούμενη φάση, με την προσθήκη νέας πληροφορίας, τα ονόματα των Domain κλάσεων του συστήματος της βιβλιοθήκης, που θα χρειαζόταν να χρησιμοποιήσει για την δημιουργία των αυτοματοποιημένων τεστ.
3. **Τρίτη φάση (Phase 3)** : Η τρίτη φάση με την σειρά της περιείχε την ίδια γνώση με την φάση 2, αλλά στο σημείο αυτό εισάχθηκε σαν επιπλέον προϋπάρχουσα γνώση οι ιδιότητες που χρησιμοποιεί κάθε μία από τις Domain κλάσεις που δόθηκαν, ώστε να μπορέσει το μοντέλο να χρησιμοποιήσει τις ιδιότητες αυτές στην παραγωγή των τεστ.
4. **Τέταρτη φάση (Phase 4)** : Στην τέταρτη και τελευταία φάση, επεκτάθηκε η γνώση της τρίτης φάσης με την εισαγωγή των υπογραφών μεθόδων που χρησιμοποιούνται από τις Domain κλάσεις. Αυτή η φάση αντιπροσωπεύει την μέγιστη ποσότητα προϋπάρχουσας γνώσης που δόθηκε στα μεγάλα γλωσσικά μοντέλα, επιτρέποντας την πληρέστερη κατανόηση του συστήματος και τη δημιουργία των πιο σύνθετων και ακριβών αυτοματοποιημένων τεστ.

Για κάθε μία από τις παραπάνω κατηγορίες προϋπάρχουσας γνώσης χρησιμοποιήθηκαν διαφορετικές τεχνικές παρουσίασης της γνώσης αυτής στα μεγάλα γλωσσικά μοντέλα, προκειμένου να βρεθεί ο αποδοτικότερος τρόπος επικοινωνίας με τα μεγάλα γλωσσικά μοντέλα. Σκοπός αυτής της προσέγγισης ήταν η ανάπτυξη μιας **μεθοδολογίας** που θα εξασφάλιζε τη βέλτιστη επικοινωνία με τα μοντέλα για την παραγωγή αυτοματοποιημένου κώδικα τεστ.

Ειδικότερα, κατά τη διάρκεια κάθε συζήτησης με τα γλωσσικά μοντέλα, εφαρμόστηκαν τέσσερις διαφορετικές τεχνικές παρουσίασης της γνώσης, οι οποίες ήταν οι εξής:

1. **Παρουσίαση όλων των σεναρίων Cucumber σε μεμονωμένα μηνύματα ή σε διαφορετικά μηνύματα** : Σε αυτήν την τεχνική, η προϋπάρχουσα γνώση για τα σενάρια Gherkin που σχετίζονταν με το σύστημα δινόταν στο μεγάλο γλωσσικό μοντέλο σε ένα ενιαίο μήνυμα (prompt) ή σε ξεχωριστά μηνύματα. Στη συνέχεια, ζητούνταν από το μοντέλο να δημιουργήσει τον κώδικα για τα Step Definitions που αντιστοιχούσαν για τα συγκεκριμένα σενάρια.
2. **Εντολή στο μεγάλο γλωσσικό μοντέλο να παράγει ή να μην παράγει τον κώδικα των Domain κλάσεων πρώτα** : Σε αυτή την τεχνική, αφού επιλέγονταν ένας από τους παραπάνω τρόπους παρουσίασης, ζητούσαμε επιπλέον, μετά την παρουσίαση των σεναρίων σε Cucumber, τη δημιουργία ή την μη δημιουργία του κώδικα για τις Domain κλάσεις που το μοντέλο θεωρούσε απαραίτητο. Ο στόχος ήταν να εξετάσουμε αν η παραγωγή του κώδικα των Domain κλάσεων πρώτα θα βελτίωνε την ποιότητα των αποτελεσμάτων στην παραγωγή των αυτοματοποιημένων τεστ ή εάν μη παρουσιάζοντας την εντολή αυτή, τα αποτελέσματα ήταν χειρότερα.

Η εφαρμογή αυτών των τεχνικών είχε ως σκοπό την κατανόηση του τρόπου με τον οποίο οι διάφορες μορφές παρουσίασης γνώσης επηρεάζουν την απόδοση των γλωσσικών μοντέλων στην παραγωγή αυτοματοποιημένου κώδικα, και συνεισέφερε στην ανάπτυξη στρατηγικών για τη βελτίωση της αποδοτικότητας της επικοινωνίας με τα μοντέλα.

## Κριτήρια αξιολόγησης

Η αξιολόγηση της απόδοσης των μεγάλων γλωσσικών μοντέλων στο πλαίσιο της εργασίας στην παραγωγή του αυτοματοποιημένου κώδικα των σεναρίων BDD, βασίζεται σε ένα σύνολο κριτηρίων τα οποία εξασφαλίζουν μία συστηματική και αντικειμενική προσέγγιση στην ανάλυση των αποτελεσμάτων. Τα κριτήρια αυτά εστιάζουν σε διάφορες πτυχές της απόδοσης των μεγάλων γλωσσικών μοντέλων, από την ακρίβεια των απαντήσεων έως μέχρι την αποδοτικότητα της παραγωγής των μηνυμάτων.

Συγκεκριμένα, τα κριτήρια αξιολόγησης περιλαμβάνουν παράγοντες όπως ο αριθμός των μηνυμάτων που απαιτούνται για να ληφθεί μια πλήρης απάντηση, καθώς και το ποσοστό των απαντήσεων που κρίνονται αποδεκτές βάσει των απαιτήσεων του σεναρίου BDD.

Ο **Πίνακας 1** παραθέτει τα κριτήρια αξιολόγησης, με κάθε σειρά να αναφέρεται σε ένα διαφορετικό αριθμημένο κριτήριο. Στις στήλες του πίνακα παρουσιάζονται οι απαραίτητες πληροφορίες για κάθε κριτήριο, όπως το όνομα, η περιγραφή, η μονάδα μέτρησης, καθώς και ο τύπος που χρησιμοποιείται για τον υπολογισμό του αποτελέσματος που περιγράφει το κριτήριο, όπου αυτό είναι εφικτό.

Πίνακας 1. Κριτήρια αξιολόγησης των μεγάλων γλωσσικών μοντέλων

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Αριθμός  Κριτηρίου | Όνομα Κριτηρίου | Περιγραφή | Μέτρηση | Τύπος |
| 1 | Παροχή των features ταυτόχρονα | Εάν παρέχουμε τα features στο LLM όλα μαζί ή σε περισσότερα από ένα μηνύματα. | Ναι/Όχι | - |
| 2 | Παροχή κώδικα Domain/DAOs/Services από την αρχή | Εάν δώσαμε οδηγίες στον LLM να παράγει τον κώδικα Domain/DAOs/Services πρώτα ή όχι. | Ναι/Όχι | - |
| 3 | Απαιτούμενα μηνύματα για ολοκληρωμένα Step Definitions | Καταμέτρηση του αριθμού των μηνυμάτων που απαιτούνται για τη δημιουργία όλων των Step Definitions με όσο το δυνατόν περισσότερο κώδικα. | Ακέραια τιμή | - |
| 4 | Χρησιμοποίηση και Ακρίβεια Data Access Objects | Αξιολόγηση της ακρίβειας χρήσης και των υπενθυμίσεων για Data Access Objects από το Ai. | Χρήση DAO: 1 αν χρησιμοποιηθεί από την αρχή  / 0.5 αν δεν χρησιμοποιηθεί από την αρχή  Ακέραιες τιμές | Βαθμολογία χρήσης DAO = Χρήση DAO × (Ακριβείς DAO - Απαιτούμενες Υπενθυμίσεις) |
| 5 | Χρησιμοποίηση και ακρίβεια των Services | Αξιολόγηση της αποδοχής και των υπενθυμίσεων για Services από το LLM. | Χρήση Services: 1 αν χρησιμοποιηθεί από την αρχή  / 0.5 αν δεν χρησιμοποιηθεί από την αρχή  Ακέραιες τιμές | Βαθμολογία χρήσης Services = Χρήση Services × (Αποδεκτά Services - Απαιτούμενες Υπενθυμίσεις) |
| 6 | Ακρίβεια κλάσεων Domain που μαντεύτηκαν/χρησιμοποιήθηκαν | Μέτρηση του ποσοστού των κλάσεων Domain που μαντεύτηκαν ή χρησιμοποιήθηκαν σωστά από το LLM. | Ποσοστιαία τιμή | Ποσοστό ακρίβειας κλάσεων Domain = (Αριθμός σωστών μαντευμένων Κλάσεων Domain / Συνολικός Αριθμός Κλάσεων Domain) × 100 |
| 7 | Αποδεκτές λύσεις Step Definition | Μέτρηση του ποσοστού των Step Definitions που είναι αποδεκτές βάσει της ολοκλήρωσης, της ακρίβειας, της λειτουργικότητας και της ενσωμάτωσης. | Ποσοστιαία τιμή | Ποσοστό αποδεκτών ορισμών Step Definitions = (Αριθμός Αποδεκτών Step Definitions / Συνολικός Αριθμός Step Definitions) × 100 |
| 8 | Καλύτερες από αποδεκτές λύσεις Step Definitions | Μέτρηση του ποσοστού των Step Definitions που είχαν καλύτερες από αποδεκτές λύσεις. | Ποσοστιαία τιμή | Ποσοστό καλύτερων από αποδεκτές λύσεων = (Αριθμός Καλύτερων από Αποδεκτές Step Definitions / Συνολικός Αριθμός Step Definitions) × 100 |
| 9 | Αντικατάσταση αντικειμένων σε φυσική γλώσσα με μεταβλητές κώδικα | Περιγραφή εάν το LLM αντικαθιστά περιπτώσεις αντικειμένων που εκφράζονται σε φυσική γλώσσα με μεταβλητές στον κώδικα. | Αλφαριθμητικό | - |
| 10 | Ακρίβεια ιδιοτήτων | Μέτρηση του ποσοστού των σωστών ιδιοτήτων που χρησιμοποιεί το LLM αφότου του έχουν δοθεί (φάση 3,4) | Ποσοστιαία τιμή | Βαθμολογία ακρίβειας ιδιοτήτων= (Σωστές ιδιότητες / (Σωστές ιδιότητες + Αγνοούμενες ιδιότητες)) × 100 |
| 11 | Ακρίβεια Μεθόδων (εξαιρουμένων των Getters και Setters) | Μέτρηση του ποσοστού των σωστών μεθόδων που χρησιμοποιεί το LLM αφότου του έχουν δοθεί (φάση 4) | Ποσοστιαία τιμή | Βαθμολογία ακρίβειας μεθόδων = (Σωστές Μέθοδοι) / (Σωστές Μέθοδοι + Αγνοούμενοι Μέθοδοι) × 100 |
| 12 | Επανάληψη/Πρόσθετη επεξήγηση | Καταμέτρηση των περιπτώσεων όπου το LLM χρειαζόταν πρόσθετη εξήγηση ή επανάληψη μηνυμάτων. | Ακέραια τιμή | - |
| 13 | Κενά Step Definitions | Καταμέτρηση των περιπτώσεων όπου το LLM παρείχε κενά Step Definitions και χωρίς κώδικα | Ακέραια τιμή | - |

Στην συνέχεια, ακολουθεί μία αναλυτική επεξήγηση κάθε κριτηρίου του **Πίνακα 1 :**

1. **Παροχή των features ταυτόχρονα:** Το κριτήριο αυτό εξετάζει αν τα features του συστήματος της βιβλιοθήκης δόθηκαν σε ένα μόνο μήνυμα ή αν δόθηκαν σταδιακά σε περισσότερα από ένα μηνύματα. Αν η παρουσίαση των χαρακτηριστικών έγινε σε ένα μήνυμα, η αξιολόγηση αποδίδει την τιμή «Ναι». Σε αντίθετη περίπτωση, αν τα χαρακτηριστικά δόθηκαν σε πολλά μηνύματα, αποδίδεται η τιμή «Όχι».
2. **Παροχή του κώδικα των Domain/Daos/Services από την αρχή:** Αυτό το κριτήριο εξετάζει αν στο μεγάλο γλωσσικό μοντέλο δόθηκε η εντολή να παράγει εξ αρχής τον κώδικα για τις κλάσεις Domain, τα Data Access Objects (DAOs) και τις υπηρεσίες (Services) που χρησιμοποιεί το σύστημα της βιβλιοθήκης. Αν δόθηκε η εντολή αυτή, η αξιολόγηση αποδίδει την τιμή «Ναι», διαφορετικά, εάν δεν δόθηκε κάποια επιπλέον εντολή, αποδίδει την τιμή «Όχι».
3. **Απαιτούμενα μηνύματα για ολοκληρωμένα Step Definitions:** Tο κριτήριο αυτό μετρά τον αριθμό των μηνυμάτων που απαιτήθηκαν για τη δημιουργία του κώδικα για τα Step Definitions από το μεγάλο γλωσσικό μοντέλο. Εξετάζεται ο συνολικός αριθμός μηνυμάτων που ανταλλάχθηκαν στην συγκεκριμένη συνομιλία, προκειμένου να ολοκληρωθούν τα Step Definitions και ο μικρότερος αριθμός μηνυμάτων δείχνει καλύτερη αποδοτικότητα και κατανόηση από το σύστημα.
4. **Χρησιμοποίηση και ακρίβεια Data Access Objects:** Το κριτήριο αυτό μετρά την ακρίβεια του μεγάλου γλωσσικού μοντέλου να χρησιμοποιεί και να μαντεύει τις σωστές κλάσεις Data Access Objects. Υπολογίζεται από τον τύπο «Βαθμολογία Χρήσης DAO = Χρήση DAO × (Ακριβείς DAO - Απαιτούμενες Υπενθυμίσεις)», όπου η μεταβλητή «χρήση DAO» έχει την τιμή 1 εάν το σύστημα χρησιμοποίησε χωρίς καμία επιπρόσθετη βοήθεια τα Data Access Objects, και 0.5 εάν χρειάστηκε κάποιο επιπλέον μήνυμα και δεν τα χρησιμοποίησε από την αρχή. Ακόμη, η μεταβλητή «Ακριβείς DAO» είναι ο αριθμός τον σωστά μαντευμένων DAO που χρησιμοποίησε το μεγάλο γλωσσικό μοντέλο στην συνομιλία, ενώ οι «απαιτούμενες υπενθυμίσεις» είναι ένας ακέραιος αριθμός ο οποίος μετρά πόσα μηνύματα χρειάστηκαν για να υπενθυμιστούν τα DAOs στο μοντέλο. Ένα υψηλό ποσοστό ακρίβειας σημαίνει καλύτερη κατανόηση και χρήση των DAOs από το μοντέλο.
5. **Χρησιμοποίηση και ακρίβεια των Services:** Το κριτήριο αυτό μετρά την ακρίβεια του μεγάλου γλωσσικού μοντέλου να χρησιμοποιεί και να μαντεύει τα σωστά Service κλάσεις, τα οποία είναι βοηθητικές κλάσεις που λειτουργούν ως υπηρεσίες για το σύστημα της βιβλιοθήκης. Υπολογίζεται από τον τύπο «Βαθμολογία Χρήσης Services = Χρήση Services × (Αποδεκτά Services - Απαιτούμενες Υπενθυμίσεις)», όπου η μεταβλητή «χρήση Services» έχει την τιμή 1 εάν το σύστημα χρησιμοποίησε χωρίς καμία επιπρόσθετη βοήθεια τα Services από την αρχή, και 0.5 εάν χρειάστηκε κάποιο επιπλέον μήνυμα για να τα χρησιμοποιήσει. Ακόμη, η μεταβλητή «Αποδεκτά Services» μετράει τον αριθμό των αποδεκτών service κλάσεων που χρησιμοποίησε το μεγάλο γλωσσικό μοντέλο, ακόμη και αν είχαν κάποιες διαφορές από τα services του πραγματικού συστήματος βιβλιοθήκης, ενώ οι «απαιτούμενες υπενθυμίσεις» είναι ένας ακέραιος αριθμός ο οποίος μετράει το πόσα μηνύματα υπενθύμισης χρειάστηκαν να δοθούν στο σύστημα για την χρήση των Services στην υλοποίησή του. Ένα υψηλό ποσοστό υποδεικνύει σημαίνει καλή κατανόηση και χρήση των Services από το μοντέλο.
6. **Ακρίβεια κλάσεων Domain που μαντεύτηκαν/χρησιμοποιήθηκαν:** Το κριτήριο αυτό μετράει το ποσοστό των Domain κλάσεων του συστήματος της βιβλιοθήκης που το μεγάλο γλωσσικό μοντέλο μάντεψε και χρησιμοποίησε σωστά στην υλοποίηση του κώδικα των Step Definitions. Το ποσοστό αυτό προκύπτει από τον τύπο «Ποσοστό Ακρίβειας Κλάσεων Domain = (Αριθμός σωστών μαντευμένων Κλάσεων Domain / Συνολικός Αριθμός Κλάσεων Domain) × 100» όπου ο συνολικός αριθμός κλάσεων Domain είναι οι συνολικές σωστές κλάσεις που υπάρχουν στο σύστημα βιβλιοθήκης. Ένα υψηλό ποσοστό υποδεικνύει ότι το μοντέλο κατάφερε να μαντέψει και να χρησιμοποιήσει σωστά τις κλάσεις Domain.
7. **Αποδεκτές λύσεις Step Definition:** Το κριτήριο αυτό μετρά το ποσοστό των αποδεκτών λύσεων του κώδικα των Step Definitions που παρήγαγε το μεγάλο γλωσσικό μοντέλο. Αποδεκτή θεωρείται μία λύση που θα μπορούσε να ενσωματωθεί με μικρές τροποποιήσεις στο πραγματικό σύστημα βιβλιοθήκης και γενικά εάν έχει μία ορθή λογική η ο κώδικας των βημάτων. Το ποσοστό αυτό προκύπτει από τον τύπο «Ποσοστό Αποδεκτών Ορισμών Step Definitions = (Αριθμός Αποδεκτών Step Definitions / Συνολικός Αριθμός Step Definitions) × 100» όπου ο συνολικός αριθμός Step Definitions είναι τα συνολικά Step Definitions που δημιουργήθηκαν για τα σενάρια Cucumber. Ένα υψηλό ποσοστό δείχνει ότι το μοντέλο παράγει λύσεις που είναι αποδεκτές και εφαρμόσιμες.
8. **Καλύτερες από αποδεκτές λύσεις Step Definitions:** Το κριτήριο αυτό μετρά το ποσοστό των αποδεκτών λύσεων του κώδικα των Step Definitions που είναι καλύτερες από τις αποδεκτές λύσεις ή ακόμα καλύτερες από τον υπάρχοντα κώδικα. Το ποσοστό αυτό προκύπτει από τον τύπο «Ποσοστό Καλύτερων από Αποδεκτές Λύσεων = (Αριθμός Καλύτερων από Αποδεκτές Step Definitions / Συνολικός Αριθμός Step Definitions) × 100» όπου ο συνολικός αριθμός Step Definitions είναι τα συνολικά Step Definitions που δημιουργήθηκαν για τα σενάρια Cucumber. Το υψηλό ποσοστό δείχνει ότι το μοντέλο δεν παρέχει μόνο αποδεκτές λύσεις, αλλά και βελτιωμένες και αποτελεσματικές λύσεις.
9. **Αντικατάσταση αντικειμένων σε φυσική γλώσσα με μεταβλητές κώδικα:** Το κριτήριο αυτό αξιολογεί την ικανότητα του μεγάλου γλωσσικού μοντέλου να αντιληφθεί και να χρησιμοποιήσει μεταβλητές που περιγράφονται σε φυσική γλώσσα στα σενάρια Cucumber και να τις ενσωματώσει ως μεταβλητές στον κώδικα των Step Definitions. Το μοντέλο αξιολογείται για το πόσο καλά μετατρέπει τις περιγραφές σε φυσική γλώσσα σε κατάλληλες μεταβλητές στον κώδικα. Η σωστή αντικατάσταση υποδεικνύει καλή κατανόηση της γλώσσας και του κώδικα που απαιτείται για την παραγωγή των Step Definitions.
10. **Ακρίβεια ιδιοτήτων:** Το κριτήριο αυτό αξιολογεί την ικανότητα του μεγάλου γλωσσικού μοντέλου να μαντεύει τις σωστές ιδιότητες όλων των Domain κλάσεων, αφότου του έχουν δοθεί ως πληροφορίες. Για τον λόγο αυτό, το κριτήριο αυτό χρησιμοποιείται μόνο την φάση 3 και στην φάση 4 όπου παρέχουμε τις πληροφορίες αυτές στο μεγάλο γλωσσικό μοντέλο. Η ικανότητα αυτή μετριέται σε ποσοστό και δίνεται από τον τύπο «Βαθμολογία Ακρίβειας ιδιοτήτων= (Σωστές ιδιότητες / (Σωστές ιδιότητες + Αγνοούμενες ιδιότητες)) × 100» όπου η μεταβλητή «Σωστές ιδιότητες» είναι ένας ακέραιος αριθμός με τον συνολικό αριθμό των σωστών ιδιοτήτων που μάντεψε το σύστημα, ενώ «Αγνοούμενες ιδιότητες» είναι οι ιδιότητες που δόθηκαν αλλά παραλήφθηκαν να χρησιμοποιηθούν από το σύστημα. Ένα υψηλό ποσοστό υποδεικνύει ότι το μοντέλο κατανόησε τις πληροφορίες που δέχθηκε και μπόρεσε να τις χρησιμοποιήσει στην υλοποίησή του.
11. **Ακρίβεια μεθόδων (εξαιρούμενων των Getters και Setters):** Το κριτήριο αυτό εξετάζει την ικανότητα του μεγάλου γλωσσικού μοντέλου να μαντεύει σωστά τις μεθόδους όλων των Domain κλάσεων, αφότου του έχουν δοθεί ως πληροφορίες. Για τον λόγο αυτό, το κριτήριο αυτό χρησιμοποιείται μόνο στην φάση 4, στην οποία παρέχουμε ως πληροφορία τις μεθόδους που αποτελούν κάθε Domain κλάση. Η ικανότητα αυτή μετριέται σε ποσοστό και δίνεται από τον τύπο «Βαθμολογία Ακρίβειας Μεθόδων = (Σωστές Μέθοδοι) / (Σωστές Μέθοδοι + Αγνοούμενοι Μέθοδοι) × 100» όπου η μεταβλητή «Σωστές Μέθοδοι» είναι ένας ακέραιος αριθμός με τον συνολικό αριθμό σωστών μεθόδων που μάντεψε το σύστημα, ενώ «Αγνοούμενοι μέθοδοι» είναι οι μέθοδοι που δόθηκαν αλλά παραλήφθηκαν να χρησιμοποιηθούν από το σύστημα. Ένα υψηλό ποσοστό υποδεικνύει ότι το μοντέλο κατανόησε τις μεθόδους που δέχθηκε και μπόρεσε να τις χρησιμοποιήσει στην υλοποίησή του.
12. **Επανάληψη/Πρόσθετη επεξήγηση:** Το κριτήριο αυτό μετρά τον αριθμό των μηνυμάτων που απαιτήθηκε να δοθούν στο μεγάλο γλωσσικό μοντέλο ως υπενθύμιση, επανάληψη ή βοήθεια για να συνεχίσει να παράγει. Ένα χαμηλό νούμερο στο κριτήριο αυτό υποδεικνύει ότι σύστημα είχε καλύτερη κατανόηση των πληροφοριών που το δόθηκαν.
13. **Κενά Step Definitions:** Το κριτήριο αυτό μετρά τον αριθμό των φορών που το μεγάλο γλωσσικό μοντέλο παρήγαγε Step Definitions χωρίς κάποιο κώδικα ή με πολύ λίγη πληροφορία για το Step Definition. Ένα χαμηλό νούμερο στο κριτήριο αυτό υποδεικνύει ότι το σύστημα παρήγαγε ευκολότερα κώδικα και με πιο αποτελεσματικό τρόπο.

Τα κριτήρια αξιολόγησης που παρουσιάζονται παραπάνω παρέχουν μια ολοκληρωμένη βάση για την ανάλυση της απόδοσης των μεγάλων γλωσσικών μοντέλων που χρησιμοποιήθηκαν στην εργασία. Καλύπτουν κρίσιμες πτυχές της χρησιμότητας των μοντέλων, προσφέροντας ένα μεθοδικό εργαλείο για τη σύγκριση και την αξιολόγηση των δυνατοτήτων τους στην παραγωγή αυτοματοποιημένου κώδικα για τα σενάρια BDD.

## Εμπειρική αξιολόγηση

Όπως αναφέρθηκε αναλυτικά και στην ενότητα της μεθοδολογίας, η αξιολόγηση των μεγάλων γλωσσικών μοντέλων έγινε σε τέσσερις φάσεις, όπου κάθε φάση αντιπροσωπεύει διαφορετική ποσότητα γνώσης που παρέχεται στα μοντέλα.

### Αξιολόγηση της φάσης 1

Ο **Πίνακας 2** που φαίνεται παρακάτω παρουσιάζει τα αποτελέσματα της **πρώτης φάσης** της αξιολόγησης της απόδοσης των μεγάλων γλωσσικών μοντέλων στην παραγωγή αυτοματοποιημένου κώδικα για τα σενάρια BDD. Στις σειρές του πίνακα παρουσιάζονται τα διαφορετικά αριθμημένα κριτήρια που αναλύθηκαν παραπάνω και στις στήλες του πίνακα οι διαφορετικές συζητήσεις που έγιναν με τα διαφορετικά μεγάλα γλωσσικά μοντέλα.

Πίνακας 2. Φάση 1 αξιολόγησης των μεγάλων γλωσσικών μοντέλων

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Φάση 1 | | | | | | | | | | | | | | | |
|  | **GPT -3.5** | | | | **GitHub COPILOT 3.5 Turbo** | | | | **GPT -4** | | | | **GPT -4o** | | |
| Αριθμός Κριτηρίου | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 4 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 4 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 4 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 |
| 1 | Ναι | Ναι | Όχι | Όχι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Όχι | Όχι | Ναι | Ναι | Όχι |
| 2 | Όχι | Ναι | Ναι | Ναι | Όχι | Όχι | Όχι | Ναι | Όχι | Ναι | Όχι | Ναι | Όχι | Ναι | Όχι |
| 3 | 13 | 10 | 11 | 11 | 8 | 7 | 9 | 8 | 9 | 8 | 8 | 8 | 5 | 7 | 5 |
| 4 | 3 | 1 | 2 | 1 | 0 | 1 | 3 | 3 | 0.5 | 3 | 1 | 2 | 2 | 2 | 2 |
| 5 | 3 | 3 | 2 | 3 | 3 | 3 | 3 | 4 | 3 | 3 | 3 | 4 | 3 | 3 | 4 |
| 6 | 28.57 % | 42.85% | 28.57% | 42.85% | 28.57% | 42.85 % | 42.85 % | 42.85 % | 28.57% | 42.85 % | 28.57% | 28.57% | 28.57% | 28.57% | 28.57% |
| 7 | 22.91% | 31.25% | 20.83% | 20.83% | 20.83% | 33.3% | 18.75% | 22.91% | 18.75% | 33.3% | 8% | 10% | 35.4% | 20,8% | 18% |
| 8 | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 2% | 0% | 0% | 2% | 0% | 0% |
| 9 | Το σύστημα δεν χρησιμοποίησε πλήρως τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα, μόνο σε λίγα Step Definitions | Το σύστημα δεν χρησιμοποίησε πλήρως τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα, μόνο σε λίγα Step Definitions | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το LLM κατάλαβε και χρησιμοποίησε εν μέρει κάποια αντικείμενα που εκφράστηκαν σε φυσική γλώσσα, αλλά όχι σε όλα τα Step Definitions | Το LLM κατανόησε σχεδόν σε όλα τα Step Definitions τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα. | Το LLM κατανόησε σχεδόν σε όλα τα Step Definitions τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα. | Το LLM κατανόησε σχεδόν σε όλα τα Step Definitions τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα. | Το LLM χρησιμοποίησε μόνο λίγα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα. | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το LLM κατάλαβε και χρησιμοποίησε εν μέρει κάποια αντικείμενα που εκφράστηκαν σε φυσική γλώσσα, αλλά όχι σε όλα τα Step Definitions | Το LLM κατάλαβε και χρησιμοποίησε εν μέρει κάποια αντικείμενα που εκφράστηκαν σε φυσική γλώσσα, αλλά όχι σε όλα τα Step Definitions | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα |
| 12 | 4 | 4 | 4 | 4 | 4 | 2 | 3 | 3 | 5 | 2 | 2 | 2 | 0 | 1 | 0 |
| 13 | 28 | 7 | 15 | 11 | 23 | 11 | 19 | 8 | 10 | 4 | 2 | 2 | 0 | 10 | 0 |

Στην αρχική αυτή φάση της αξιολόγησης, παρατηρήθηκε ότι όλα τα μεγάλα γλωσσικά μοντέλα παρουσίασαν περιορισμένα αποτελέσματα σε σχέση με την ποσότητα του παραγόμενου κώδικα, τη λεπτομέρεια των απαντήσεων και την αποδοχή των παραγόμενων Step Definitions. Συγκεκριμένα, στις αρχικές συζητήσεις όπου δεν δόθηκε εντολή δημιουργίας του κώδικα για τα Domain/Data Access Objects/Services, τα μοντέλα αντιμετώπισαν σημαντικές δυσκολίες στην παραγωγή κώδικα. Αυτό είχε ως αποτέλεσμα την εμφάνιση πολλών κενών Step Definitions και την ανάγκη για πολλές υπενθυμίσεις ή εντολές προς το σύστημα για να παραχθεί ο απαιτούμενος κώδικας. Στον πίνακα εμφανίζεται ότι ο αριθμός των κενών Step Definitions έφτασε έως και 28.

Η κατάσταση αυτή βελτιώθηκε σημαντικά όταν αρχίσαμε να δίνουμε την εντολή στα μοντέλα να παράγουν πρώτα τον κώδικα των Domain/Data Access Objects/Services, προτού προχωρήσουν στην παραγωγή των Step Definitions. Αυτή η προσέγγιση βοήθησε τα μοντέλα να εστιάσουν καλύτερα και να χρησιμοποιήσουν αποτελεσματικότερα τις κλάσεις που είχαν ήδη δημιουργήσει.

Ως αποτέλεσμα, παρατηρήθηκε σημαντική μείωση στον αριθμό των κενών Step Definitions και βελτίωση στην ποιότητα των παραγόμενων Step Definitions, με πολλά μοντέλα να επιδεικνύουν αυξημένα ποσοστά αποδεκτών Step Definitions ή να διατηρούν υψηλά ποσοστά αποδοχής. Επιπλέον, η τεχνική αυτή φάνηκε να ενίσχυσε τη χρήση των Data Access Objects από τα μοντέλα, κάτι που είχε παραληφθεί σε πολλές προηγούμενες περιπτώσεις. Ειδικότερα, τα GitHub Copilot και GPT-4o ξεχώρισαν καθώς παρήγαγαν από μόνα τους τον κώδικα των Domain/Data Access Objects/Services χωρίς επιπλέον εντολές, υποδεικνύοντας την ευεργετική επίδραση αυτής της τεχνικής στην απόδοση των μοντέλων. Τα δύο αυτά μοντέλα επίσης αποδείχθηκαν ιδιαίτερα ικανά στην κατανόηση και τη χρήση μεταβλητών σε φυσική γλώσσα, κάτι που συνέβαλε στην παραγωγή πιο ακριβούς κώδικα.

Στη συνέχεια, δοκιμάστηκε η μέθοδος της σταδιακής παρουσίασης των απαιτήσεων, όπου κάθε feature δινόταν σε ξεχωριστό μήνυμα, με σκοπό να επικεντρωθεί το μοντέλο σε κάθε feature ξεχωριστά και να βελτιώσει τα αποτελέσματα. Ωστόσο, η παρατήρηση έδειξε ότι η σταδιακή παρουσίαση δεν βελτίωσε την απόδοση σε σύγκριση με άλλες τεχνικές, καθώς τα αποδεκτά Step Definitions συχνά ήταν λιγότερα και τα μοντέλα έχαναν την ευρύτερη εικόνα του συστήματος που θα μπορούσε να οδηγήσει σε πιο ακριβή αποτελέσματα.

Από την πρώτη φάση, προκύπτει ότι το GitHub Copilot υπερείχε των GPT-3.5 και GPT-4 σε πολλές περιπτώσεις, καθώς παρουσίασε καλύτερη κατανόηση των δεδομένων και ταχύτερη παραγωγή κώδικα με λιγότερα κενά Step Definitions. Το GPT-4o, ως το πιο πρόσφατο και προηγμένο μοντέλο, αποδείχθηκε ανώτερο από όλα τα άλλα μοντέλα, επιδεικνύοντας καλύτερη κατανόηση των πληροφοριών, ελάχιστο αριθμό μηνυμάτων για την παραγωγή των Step Definitions και ταχύτητα παραγωγής κώδικα που ξεπέρασε όλα τα άλλα μοντέλα.

### Αξιολόγηση της φάσης 2

Στη **δεύτερη φάση** του πειράματος, χρησιμοποιήθηκαν οι ίδιες τεχνικές παρουσίασης της πληροφορίας όπως και στην πρώτη φάση, και παρουσιάζονται αναλυτικά στον **Πίνακα 3** παρακάτω. Ωστόσο, τα αποτελέσματα ήταν κατά μεγάλο ποσοστό παρόμοια ή και χειρότερα σε πολλές περιπτώσεις. Ειδικότερα, παρατηρήθηκε ότι τα ποσοστά των αποδεκτών Step Definitions ήταν πολύ χαμηλότερα σε σύγκριση με την πρώτη φάση, παρά το γεγονός ότι τα γλωσσικά μοντέλα είχαν πλέον περισσότερη γνώση. Αυτό υποδηλώνει ότι τα ονόματα των κλάσεων δεν συμβάλλουν ουσιαστικά στη βελτίωση των αποτελεσμάτων.

Στις περισσότερες περιπτώσεις, τα μοντέλα αδυνατούσαν να κατανοήσουν τις συνδέσεις μεταξύ των κλάσεων και περιορίζονταν στη χρήση μόνο των πιο βασικών κλάσεων, παραλείποντας πολλές άλλες. Όπως και στην προηγούμενη φάση, η εντολή για την παραγωγή των Domain/Data Access Objects/Services συνέβαλε σημαντικά στη βελτίωση των απαντήσεων των γλωσσικών μοντέλων. Ειδικότερα, αυτή η τεχνική βοήθησε τα μοντέλα να κατανοήσουν καλύτερα και να παράγουν πιο αποδεκτά Step Definitions, ενώ διευκόλυνε επίσης τη χρήση των Data Access Objects για την αποθήκευση και εύρεση αντικειμένων, όπως φαίνεται και από τα ποσοστά χρήσης ορθών Data Access Objects.

Η τεχνική της παρουσίασης των απαιτήσεων σε φυσική γλώσσα σε ξεχωριστά μηνύματα δημιούργησε παρόμοια προβλήματα με εκείνα της πρώτης φάσης. Αν και υπήρξαν κάποιες περιπτώσεις με θετικά αποτελέσματα, η συνολική απόδοση δεν δικαίωσε την τεχνική αυτή ως την καλύτερη επιλογή. Αντιθέτως, η παρουσίαση των απαιτήσεων σε ένα μόνο μήνυμα και η εντολή για παραγωγή του κώδικα των Domain/Data Access Objects/Services πρώτα παρέμειναν πιο αποτελεσματικές.

Συμπερασματικά, το GitHub Copilot συνεχίζει να ξεχωρίζει σε σχέση με τα GPT-3.5 και GPT-4, αποδεικνύοντας ότι κατανοεί καλύτερα τις μεταβλητές που έχουν δοθεί σε φυσική γλώσσα, όπως 'George Red', 'Moby Dick' και 'Harry Potter', και τις χρησιμοποιεί σωστά σε πολλές περιπτώσεις. Ωστόσο, το GPT-4o επιβεβαιώνεται για άλλη μία φορά ως το ανώτερο μοντέλο, παράγοντας πολύ καλύτερα αποτελέσματα από τα υπόλοιπα γλωσσικά μοντέλα. Το GPT-4o χρησιμοποίησε αποδοτικά την επιπλέον γνώση από τα ονόματα των Domain κλάσεων και παρήγαγε πολλά και υψηλής ποιότητας Step Definitions, όπως φαίνεται από τα αποτελέσματα στο κριτήριο αξιολόγησης 8.

Πίνακας 3. Φάση 2 αξιολόγησης των μεγάλων γλωσσικών μοντέλων

### Αξιολόγηση της φάσης 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Φάση 2 | | | | | | | | | | | | | | | |
|  | **GPT -3.5** | | | **GitHub COPILOT 3.5 Turbo** | | | | | **GPT -4** | | | | **GPT -4o** | | |
| Αριθμός Κριτηρίου | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 4 | Συνομιλία 5 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 4 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 |
| 1 | Ναι | Ναι | Όχι | Ναι | Ναι | Ναι | Ναι | Όχι | Ναι | Ναι | Όχι | Όχι | Ναι | Ναι | Ναι |
| 2 | Ναι | Όχι | Όχι | Όχι | Όχι | Ναι | Ναι | Ναι | Όχι | Ναι | Όχι | Ναι | Όχι | Ναι | Ναι |
| 3 | 10 | 10 | 7 | 9 | 11 | 10 | 8 | 7 | 9 | 9 | 9 | 8 | 6 | 3 | 5 |
| 4 | 3 | 0 | 0 | 0 | 3 | 0 | 3 | 1 | 3 | 2 | 1 | 1 | 3 | 3 | 3 |
| 5 | 3 | 2 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 4 | 3 | 3 | 3 | 1 | 3 |
| 7 | 6.25% | 2.08% | 10.4% | 20.8% | 12.5% | 15% | 33.3% | 20.8% | 25% | 33.3% | 12.5% | 31.2% | 50% | 54.1% | 37.5% |
| 8 | 0% | 0% | 0% | 0% | 0% | 0% | 2% | 2% | 2% | 2% | 2% | 8.33% | 20.83% | 20.83% | 8.33% |
| 9 | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. | Το LLM δεν κατανόησε πολύ καλά τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε μόνο σε σπάνιες περιπτώσεις . | Το LLM δεν κατανόησε πολύ καλά τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε μόνο σε σπάνιες περιπτώσεις. | Το LLM δεν κατανόησε πολύ καλά τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε μόνο σε σπάνιες περιπτώσεις. | Το LLM δεν κατανόησε πολύ καλά τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε μόνο σε σπάνιες περιπτώσεις. | Το LLM κατανόησε μόνο εν μέρει τα αντικείμενα που δόθηκαν σε φυσική γλώσσα, χάνοντας πολλά από αυτά. | Το LLM κατανόησε άψογα τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. | Το LLM δεν κατανόησε ούτε χρησιμοποίησε τα αντικείμενα που δόθηκαν σε φυσική γλώσσα. |
| 12 | 4 | 3 | 1 | 1 | 5 | 2 | 1 | 1 | 2 | 2 | 3 | 2 | 0 | 0 | 0 |
| 13 | 31 | 26 | 26 | 3 | 15 | 0 | 9 | 5 | 1 | 1 | 1 | 3 | 6 | 4 | 0 |

Στην **τρίτη φάση** του πειράματος, επεκτείνεται η γνώση που παρέχουμε στα γλωσσικά μοντέλα, και όπως αναφέρθηκε και στην υποενότητα της μεθοδολογίας, προσθέτουμε νέες πληροφορίες πέρα από την αρχιτεκτονική του συστήματος, τις γενικές πληροφορίες σχετικά με το σύστημα, τις απαιτήσεις σε φυσική γλώσσα και τα ονόματα των Domain κλάσεων. Αυτή τη φορά, συμπληρώνουμε την πληροφορία με τις ιδιότητες κάθε κλάσης, δίνοντας στα μοντέλα τη δυνατότητα να ανακαλύψουν και να κατανοήσουν καλύτερα τις σχέσεις μεταξύ των κλάσεων και τις μεθόδους που πρέπει να εφαρμόσουν.

Η προσθήκη αυτών των ιδιοτήτων, όπως περιγράφεται και από τον **Πίνακα 4** παρακάτω, φαίνεται να έχει σημαντική επίδραση στην απόδοση των γλωσσικών μοντέλων. Τα μοντέλα βελτίωσαν σημαντικά την ανάλυση και τη χρήση των κλάσεων, όπως δείχνουν τα αυξημένα ποσοστά στα αποδεκτά Step Definitions και τα Step Definitions που ξεπέρασαν την αποδεκτή λύση. Η εντολή για την παραγωγή των Domain/Data Access Objects/Services συνεχίζει να αποδεικνύεται χρήσιμη, με τα γλωσσικά μοντέλα να κατανοούν και να παράγουν κώδικα πιο αποτελεσματικά. Αυτό αποτυπώνεται στην αύξηση των ποσοστών στις αποδοτικές συναρτήσεις και τη μείωση των κενών Step Definitions.

Σε αυτή τη φάση, παρατηρείται ότι η τεχνική της παρουσίασης των απαιτήσεων σε φυσική γλώσσα σε μεμονωμένα μηνύματα χρησιμοποιείται όλο και λιγότερο. Τα αποτελέσματα από τις προηγούμενες φάσεις δεν επιβεβαιώνουν ότι αυτή η τεχνική προσφέρει τα καλύτερα αποτελέσματα, με αποτέλεσμα οι συνομιλίες να επικεντρώνονται περισσότερο στις άλλες αναφερθείσες τεχνικές.

Το GitHub Copilot παρουσιάζει σημαντική βελτίωση σε σύγκριση με τα GPT-3.5 και GPT-4, ειδικότερα όσον αφορά τα αποδεκτά Step Definitions, τα καλύτερα από τα αποδεκτά Step Definitions, και την κατανόηση των μεταβλητών που δίνονται σε φυσική γλώσσα. Από την άλλη, το GPT-4o συνεχίζει να ξεχωρίζει, παρέχοντας πλήρη κώδικα με ελάχιστα μηνύματα, χωρίς κενά, και με εξαιρετική κατανόηση των δεδομένων και των ιδιοτήτων που του έχουν παραχωρηθεί.

Επιπλέον, τα τέσσερα γλωσσικά μοντέλα δείχνουν σημαντική πρόοδο στην κατανόηση των ορθών Data Access Objects και Services, σε σύγκριση με τις προηγούμενες φάσεις, όπως αποδεικνύεται από τα κριτήρια αξιολόγησης 4 και 5.

Πίνακας 4. Φάση 3 αξιολόγησης των μεγάλων γλωσσικών μοντέλων

### Αξιολόγηση της φάσης 4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Φάση 3 | | | | | | | | | | | | | | |
|  | **GPT -3.5** | | | | **GitHub COPILOT 3.5 Turbo** | | | | **GPT -4** | | | **GPT -4o** | | |
| Αριθμός Κριτηρίου | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 4 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 4 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 |
| 1 | Ναι | Ναι | Ναι | Όχι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι |
| 2 | Όχι | Όχι | Ναι | Ναι | Όχι | Όχι(το έκανε από μόνο του) | Ναι | Ναι | Όχι | Ναι | Όχι | Όχι | Ναι | Ναι |
| 3 | 8 | 11 | 9 | 11 | 6 | 11 | 9 | 9 | 11 | 7 | 11 | 4 | 4 | 3 |
| 4 | 1 | 0 | 3 | 2 | 2 | 2 | 0 | 3 | 2 | 3 | 1 | 3 | 3 | 3 |
| 5 | 2 | 0.5 | 4 | 3 | 3 | 3 | 1 | 3 | 3 | 4 | 4 | 3 | 3 | 2 |
| 7 | 6.25% | 18.75% | 37.5% | 35.41%% | 54.1% | 56.25% | 22.91% | 20.8% | 22.91% | 37.5% | 20.8% | 68.75% | 60.41% | 56.25% |
| 8 | 0% | 0% | 4.1% | 4.1% | 4.1% | 18.75% | 14% | 2% | 4.1% | 10.41% | 4.1% | 22.91% | 20.83% | 14.58% |
| 9 | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το LLM κατανόησε εν μέρει τα αντικείμενα που δόθηκαν σε φυσική γλώσσα.  4o | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το LLM κατανόησε σε κάποιο βαθμό τα αντικείμενα που δόθηκαν σε φυσική γλώσσα, αλλά δεν τα χρησιμοποίησε σε όλα τα Step Definitions | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. | Το LLM κατανόησε σε κάποιο βαθμό τα αντικείμενα που δόθηκαν σε φυσική γλώσσα, αλλά δεν τα χρησιμοποίησε σε όλους τους ορισμούς βημάτων. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το LLM κατανόησε σε κάποιο βαθμό τα αντικείμενα που δόθηκαν σε φυσική γλώσσα, αλλά δεν τα χρησιμοποίησε σε όλους τους ορισμούς βημάτων.. | Το LLM κατανόησε σε κάποιο βαθμό τα αντικείμενα που δόθηκαν σε φυσική γλώσσα, αλλά δεν τα χρησιμοποίησε σε όλους τους ορισμούς βημάτων. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε σωστά. |
| 10 | 25% | 15.6% | 31.25% | 43.75% | 42.85% | 59.37% | 62,5% | 31.25% |  |  |  | 71.87% | 78.1% | 68.75% |
| 12 | 1 | 3 | 2 | 4 | 0 | 4 | 1 | 2 | 2 | 2 | 2 | 0 | 0 | 0 |
| 13 | 24 | 4 | 7 | 38 | 6 | 12 | 0 | 3 | 1 | 1 | 2 | 0 | 2 | 9 |

Στην τελευταία φάση του πειράματος, τη **φάση 4**, όπως αναφέρθηκε προηγουμένως, επεκτείνουμε τη γνώση που παρέχουμε στα γλωσσικά μοντέλα με την εισαγωγή επιπλέον πληροφοριών. Εκτός από την αρχιτεκτονική του συστήματος, τις γενικές πληροφορίες, τις απαιτήσεις σε φυσική γλώσσα, τα ονόματα των Domain κλάσεων και τις ιδιότητες κάθε κλάσης, προσθέτουμε και τα ονόματα όλων των μεθόδων κάθε κλάσης, συμπεριλαμβανομένων των τύπων επιστροφής και των παραμέτρων τους. Αυτό επιτρέπει στα μοντέλα να έχουν την καλύτερη δυνατή γνώση για να παράγουν αποδοτικά και αποδεκτά αποτελέσματα στα αυτοματοποιημένα τεστ.

Η προσθήκη αυτών των πληροφοριών, όπως φαίνεται και στον **Πίνακα 5** παρακάτω, έχει θετική επίδραση στην απόδοση των γλωσσικών μοντέλων. Παρατηρήθηκε ότι, σε σχεδόν κάθε συζήτηση με κάθε γλωσσικό μοντέλο, οι κλάσεις Data Access Objects και Services κατανοήθηκαν πλήρως, κάτι που είχε αρχίσει να παρατηρείται στην προηγούμενη φάση αλλά σε μικρότερο βαθμό. Τα ποσοστά των αποδεκτών Step Definitions και των καλύτερων από τα αποδεκτά Step Definitions αυξήθηκαν σημαντικά, ανεξαρτήτως του τρόπου παρουσίασης των πληροφοριών, δείχνοντας ότι η προσθήκη αυτών των στοιχείων ήταν αναγκαία για τη βελτίωση της απόδοσης του συστήματος.

Η εντολή για την παραγωγή του κώδικα των Domain/Data Access Objects/Services πρώτα συνέχισε να βελτιώνει την απόδοση σε ορισμένες περιπτώσεις, αν και δεν παρατηρήθηκε σημαντική διαφορά σε σχέση με τις συνομιλίες όπου η εντολή αυτή παραλείφθηκε, καθώς το σύστημα συχνά παρήγαγε αυτόν τον κώδικα αυτόματα. Η τάση του συστήματος να παράγει πρώτα τον κώδικα των Domain/Data Access Objects/Services επιβεβαιώθηκε ξανά, συμβάλλοντας στην αποτελεσματική παραγωγή του αυτοματοποιημένου κώδικα.

Η τεχνική της παρουσίασης των απαιτήσεων σε ξεχωριστά μηνύματα παραλείφθηκε και σε αυτή τη φάση, καθώς αποδείχθηκε ότι έχει περισσότερα μειονεκτήματα από τα οφέλη της.

Το GitHub Copilot ξεχώρισε για την ικανότητά του να κατανοεί καλύτερα από τα GPT-3.5 και GPT-4 κάθε κριτήριο, αν και, αν και είχε μικρό ποσοστό καλύτερων από τα αποδεκτά Step Definitions, δεν έφτασε στα σχεδόν τέλεια αποτελέσματα του GPT-4o. Το GPT-4o, με τις νέες πληροφορίες, παρουσίασε εξαιρετικά αποτελέσματα με ελάχιστα μηνύματα, καταδεικνύοντας μεγάλη ταχύτητα και αποδοτικότητα, και σχεδόν τέλεια ποσοστά στην χρήση των ιδιοτήτων και συναρτήσεων που του δόθηκαν (κριτήρια 10 και 11).

Συνολικά, τα γλωσσικά μοντέλα GitHub Copilot και GPT-4o είναι τα μόνα που κατανοούν σε μεγάλο βαθμό τη χρήση μεταβλητών σε φυσική γλώσσα και τις εφαρμόζουν σωστά στα αυτοματοποιημένα τεστ.

Πίνακας 5. Φάση 4 αξιολόγησης των μεγάλων γλωσσικών μοντέλων

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Φάση 4 | | | | | | | | | | | | | |
|  | **GPT -3.5** | | | | **GitHub COPILOT 3.5 Turbo** | | | **GPT -4** | | | **GPT -4o** | | |
| Αριθμός Κριτηρίου | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 4 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 | Συνομιλία 1 | Συνομιλία 2 | Συνομιλία 3 |
| 1 | Ναι | Ναι | Ναι | Όχι | Ναι | Ναι | Ναι | Ναι | Ναι | Όχι | Ναι | Ναι | Ναι |
| 2 | Ναι | Ναι | Ναι | Ναι | Ναι | Ναι | Όχι | Ναι | Όχι | Όχι | Ναι | Όχι(το έκανε μόνο του) | Όχι(το έκανε μόνο του) |
| 3 | 14 | 10 | 14 | 14 | 11 | 8 | 8 | 10 | 12 | 8 | 4 | 4 | 3 |
| 4 | 3 | 3 | 3 | 1 | 3 | 3 | 1 | 3 | 2 | 2 | 3 | 3 | 3 |
| 5 | 3 | 2 | 3 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 4 | 2 |
| 7 | 10.41% | 37.5% | 37.5% | 33.3% | 33.3% | 52.08% | 43.75% | 43.75% | 27.07% | 31.25% | 58.33% | 72.91% | 64.58% |
| 8 | 0% | 16.6% | 12.5% | 10.41% | 14.5% | 12.51% | 14.5% | 14.5% | 4.1% | 6.2% | 20.83% | 31.25% | 27.08% |
| 9 | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το LLM κατανόησε σε κάποιο βαθμό τα αντικείμενα που παρέχονται σε φυσική γλώσσα, αλλά δεν τα χρησιμοποίησε σωστά σε όλα τα Step Definitions | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το LLM χρησιμοποίησε τα αντικείμενα που δόθηκαν σε φυσική γλώσσα μόνο μερικές φορές, αλλά όχι τόσο καλά ώστε να αξίζει να σημειωθεί. | Το LLM κατανόησε τα αντικείμενα που δόθηκαν σε φυσική γλώσσα σε πολλές περιπτώσεις, αλλά όχι σε όλες. | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το LLM κατανόησε σε κάποιο βαθμό τα αντικείμενα που παρέχονται σε φυσική γλώσσα, αλλά δεν τα χρησιμοποίησε σωστά σε όλα τα Step Definitions. | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | Το σύστημα δεν χρησιμοποίησε καθόλου τα αντικείμενα που εκφράστηκαν σε φυσική γλώσσα | TΤο LLM κατανόησε σε κάποιο βαθμό τα αντικείμενα που παρέχονται σε φυσική γλώσσα, αλλά δεν τα χρησιμοποίησε σωστά σε όλους τους ορισμούς βημάτων. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε πολύ καλά. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε πολύ καλά. | Το LLM κατανόησε τέλεια τα αντικείμενα που δόθηκαν σε φυσική γλώσσα και τα χρησιμοποίησε πολύ καλά. |
| 10 | 18.75% | 78.12% | 52.12% | 46.87% | 25% | 46.8% | 46.8% | 46.8% | 46.8% | 43.75% | 90.6% | 90.6% | 90.6% |
| 11 | 11% | 46.1% | 34.6% | 34.6% | 30.7% | 42.3% | 30.7% | 26.9% | 34.6% | 34.6% | 50% | 34.6% | 50% |
| 12 | 5 | 4 | 5 | 6 | 4 | 1 | 2 | 3 | 2 | 1 | 0 | 0 | 0 |
| 13 | 14 | 1 | 10 | 9 | 20 | 0 | 1 | 2 | 0 | 1 | 6 | 4 | 3 |

# ΣΥΜΠΕΡΑΣΜΑΤΑ

Ολοκληρώνοντας, η ανάλυση της χρήσης των μεγάλων γλωσσικών μοντέλων στα πλαίσια του BDD αποδεικνύει ότι τα εργαλεία αυτά μπορούν να προσφέρουν πληθώρα πλεονεκτήματα στον τομέα της ανάπτυξης λογισμικού και ειδικότερα στην δημιουργία αυτοματοποιημένου ελέγχου κώδικα. Συγκεκριμένα, όπως ερευνήθηκε και στην εργασία, τα μεγάλα γλωσσικά μοντέλα έχουν αποδειχθεί εξαιρετικά χρήσιμα στην αυτοματοποίηση της δημιουργίας σεναρίων BDD, στη βελτίωση της ποιότητας του κώδικα, και στη μείωση των σφαλμάτων. Ωστόσο, η έρευνα έχει επίσης αναδείξει περιορισμούς στην δημιουργία του αυτοματοποιημένου κώδικα, όπως η ανάγκη για περαιτέρω κατανόηση των δεδομένων.

Τα αποτελέσματα της μελέτης και του πειράματος που διεξήχθη, δείχνουν καθαρά ότι η εφαρμογή των μεγάλων γλωσσικών μοντέλων μπορεί να βελτιώσει σημαντικά την διαδικασία της δημιουργίας αυτοματοποιημένου κώδικα μέσω BDD, αλλά και να παρουσιάσει ορισμένα περιορισμένα αποτελέσματα που απαιτούν βελτίωση.

Στην πρώτη φάση, παρατηρήθηκε ότι τα γλωσσικά μοντέλα, παρά τη δυνατότητά τους να παράγουν κώδικα, παρουσίασαν περιορισμένα αποτελέσματα όσον αφορά την ποσότητα, την λεπτομέρεια και την αποδοχή των παραγόμενων Step Definitions. Ειδικότερα, παρατηρήθηκαν πολλά κενά Step Definitions, **ιδιαίτερα** όταν δεν δόθηκε αρχική εντολή για την παραγωγή του κώδικα των Domain/Data Access Objects/Services. Η εφαρμογή της εντολής αυτής βοήθησε στην καλύτερη εστίαση των μοντέλων και στη βελτίωση των αποτελεσμάτων, ενώ το GitHub Copilot και το GPT-4o ξεχώρισαν για την ικανότητά τους στην κατανόηση και χρήση μεταβλητών σε φυσική γλώσσα.

Στη δεύτερη φάση, αν και χρησιμοποιήθηκαν οι ίδιες τεχνικές παρουσίασης της πληροφορίας, τα αποτελέσματα δεν παρουσίασαν σημαντική βελτίωση σε σχέση με την πρώτη φάση. Τα ποσοστά αποδεκτών Step Definitions ήταν χαμηλότερα, και οι τεχνικές όπως η σταδιακή παρουσίαση δεν αποδείχθηκαν πιο αποτελεσματικές. Ωστόσο, η εντολή για παραγωγή του κώδικα των Domain/Data Access Objects/Services συνέχισε να **βελτιώνει** τα αποτελέσματα.

Στην τρίτη φάση, η προσθήκη ιδιοτήτων κλάσεων αποδείχθηκε ότι είχε θετική επίδραση στην απόδοση των μοντέλων. Τα μεγάλα γλωσσικά μοντέλα παρουσίασαν βελτιωμένη κατανόηση και χρήση των κλάσεων, οδηγώντας σε υψηλότερα ποσοστά αποδεκτών Step Definitions. Εντούτοις, η τεχνική της παρουσίασης απαιτήσεων σε ξεχωριστά μηνύματα παρέμεινε λιγότερο αποτελεσματική.

Στην τελευταία φάση, η προσθήκη ονομάτων μεθόδων, τύπων επιστροφής και παραμέτρων κλάσεων είχε θετική επίδραση στην απόδοση των μοντέλων. Τα GitHub Copilot και GPT-4o ξεχώρισαν και πάλι για την ικανότητά τους να κατανοούν και να χρησιμοποιούν τις μεταβλητές σε φυσική γλώσσα, ενώ το GPT-4o παρουσίασε σχεδόν τέλεια αποτελέσματα με υψηλή ταχύτητα και αποδοτικότητα.

Συνολικά, τα γλωσσικά μοντέλα GitHub Copilot και GPT-4o ξεχώρισαν ως τα πιο αποτελεσματικά στην παραγωγή κώδικα για σενάρια BDD, με το GPT-4o να παρουσιάζει τα καλύτερα αποτελέσματα. Οι προσθήκες επιπλέον πληροφοριών, όπως οι ιδιότητες κλάσεων και τα ονόματα μεθόδων, βελτίωσαν την απόδοση, ενώ η τεχνική της δημιουργίας πρώτα του κώδικα των domain /Data Access Objects / Services φαίνεται να είναι η αποτελεσματικότερη τεχνική επικοινωνίας με τα μεγάλα γλωσσικά μοντέλα για παραγωγή κώδικα.

# ΒΙΒΛΙΟΓΡΑΦΙΑ

Cucumber. (χ.χ.). *Gherkin Reference*. Ανάκτηση από Cucumber documentation: https://cucumber.io/docs/gherkin/reference/

Fitzgibbons, L. (2021, Οκτώβριος). *behavior-driven development (BDD).* Ανάκτηση από TechTarget: https://www.techtarget.com/searchsoftwarequality/definition/Behavior-driven-development-BDD

Lee, T. B. (2023, Ιούλιος 31). Ανάκτηση από arstechnica.com: https://arstechnica.com/science/2023/07/a-jargon-free-explanation-of-how-ai-large-language-models-work/6/

Lutkevich, B. (2024, Ιούνιος 21). *TechTarget.* Ανάκτηση από https://www.techtarget.com/whatis/feature/12-of-the-best-large-language-models

Pathak, K. (2022, Σεπτέμβριος 12). *Cucumber Best Practices to follow for efficient BDD Testing*. Ανάκτηση από Medium: https://kailash-pathak.medium.com/cucumber-best-practices-to-follow-for-efficient-bdd-testing-b3eb1c7e9757

Pi, W. (2024, Μάϊος 7). *Research Graph.* Ανάκτηση από Medium: https://medium.com/@researchgraph/brief-introduction-to-the-history-of-large-language-models-llms-3c2efa517112

Sanderson, G. (2017, Οκτώβριος 16). *3blue1brown.* Ανάκτηση από 3blue1brown: https://www.3blue1brown.com/lessons/gradient-descent#another-way-to-think-about-the-gradient

Sumrak, J. (2024, Μάρτιος 11). *7 LLM use cases and applications in 2024.* Ανάκτηση από AssemblyAI: https://www.assemblyai.com/blog/llm-use-cases/

Xinyi Hou, Y. Z. (2023). *Large Language Models for Software Engineering: A Systematic Literature Review.* Ανάκτηση από https://ar5iv.labs.arxiv.org/html/2308.10620

Ανδρουτσόπουλος, Ί. (2024, Φεβρουάριος). Τεχνητή Νοημοσύνη και Μεγάλα Γλωσσικά Μοντέλα. *ΟΠΑ News Εφημερίδα Οικονομικού Πανεπιστημίου Αθηνών Τεύχος 51*, σσ. 8-9.

# ΠΑΡΑΡΤΗΜΑΤΑ