



ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

Διεθνές Πανεπιστήμιο Ελλάδος
Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη Εφαρμογής Ιστού Ερωταποκρίσεων σε Επιστημονικά Άρθρα με χρήση RAG και Vector Embeddings



Φοιτητής: Κωνσταντίνος Ζοσιμίδης
Αριθμός Μητρώου: 185379

Επιβλέπων: Στέφανος Ουγιάρογλου

6 Σεπτεμβρίου 2026

Title of Dissertation Development of a Question-Answering Web Application for Scientific
Articles using RAG and Vector Embeddings

Code of Dissertation 23101

Student's full name Konstantinos Zosimidis

Supervisor's full name Stefanos Ougiaroglou

Date of undertaking 31-03-2026

Date of completion 06-09-2026

We hereby affirm the authorship of this paper as well as the acknowledgement and credit of whichever assistance We received in its composition. We have, furthermore, noted the various sources from which We extracted data, ideas, visual or written material, in paraphrase or exact quotation. Moreover, we affirm the exclusive composition of this paper by myself only, for the purpose of it being a dissertation, in the Department of Information and Electronic Engineering of the I.H.U.

This paper constitutes the intellectual property of Konstantinos Zosimidis, the student that composed it. According to the open-access policy, the author/composer offers the International Hellenic University authorisation to use the right to reproduce, borrow, publicly present and digitally distribute the paper globally, in electronic form and media of all kinds, for teaching or research purposes, voluntarily. Open access to the full text, by no means grants the right to trespass the intellectual property of the author/composer, nor does it authorise the reproduction, republication, duplication, selling, commercial use, distribution, publication, downloading, uploading, translation, modification of any kind, in part or summary of the paper, without the explicit written consent of the authors.

The approval of this dissertation by the Department of Information and Electronic Engineering of the International Hellenic University, does not necessarily entail the adoption of the author's views, on behalf of the Department.

Αφιέρωση

Στους γονείς μου.

Πρόλογος

Η ιδέα για αυτήν την εργασία δεν ήρθε από κάποιο μάθημα της σχολής αλλά από ένα πρόβλημα που ο ίδιος αντιμετώπισα στις εργασίες τις οποίες έκανα κατά την διάρκεια της φοίτησής μου. Ψάχνοντας άρθρα κατάλαβα ότι ο χρόνος δεν φεύγει στο να τα βρεις αλλά στο να βρεις μέσα τους το σημείο που σε ενδιαφέρει να πάρεις για την εργασία σου και ότι η περίληψη στην αρχή κάθε επιστημονικού άρθρου σχεδόν ποτέ δεν μπορεί να σε βοηθήσει σε αυτό που ρωτάς επειδή δεν γράφτηκε για σένα. Το πρώτο πράγμα που δοκίμασα τότε είναι να δώσω δύο τρία επιστημονικά άρθρα σε ένα γλωσσικό μοντέλο και έπιασε για αυτά τα άρθρα αλλά αντιμετώπισα δύο προβλήματα. Το πρώτο είναι ότι σε κάθε καινούργια συνομιλία έπρεπε να ξανανεβάζω τα αρχεία και δεύτερον που είναι το κύριο πρόβλημα είναι ότι δεν ήξερα και δεν είχα κάποιο τρόπο να ελέγχω αν αυτό που μου απαντάει υπάρχει όντως στα επιστημονικά άρθρα που του έδωσα ή τα είχε συμπληρώσει από τις γνώσεις που έχει το ίδιο.

Από εκεί βγήκε και ο κανόνας πάνω στο οποίο χτίστηκε ολόκληρο το σύστημα, ότι η απάντηση επιτρέπεται να λέει μόνο ό,τι υπάρχει στο άρθρο το οποίο έχει ανεβάσει ο χρήστης και ότι πρέπει πάντα να δείχνει από πού το πήρε και από ποια ακριβώς σελίδα, ακόμα και όταν αυτό σημαίνει ότι θα αρνηθεί να απαντήσει άμα δεν υπάρχει. Ένα σύστημα το οποίο απαντά δεν ξέρω όταν δεν ξέρει είναι λιγότερο εντυπωσιακό αλλά είναι το μόνο που είναι πραγματικό και μπορεί να χρησιμοποιηθεί για πραγματική δουλειά.

Το κομμάτι που δεν το περίμενα ήταν πόσο δύσκολο είναι να μετρήσεις ένα τέτοιο σύστημα από το να το φτιάξεις και αυτό γιατί η υλοποίηση έχει σαφή πράγματα να κάνεις για να φτάσεις από την αρχή στο τέλος ενώ η μέτρηση σε αναγκάζει να ελέγξεις όλες τις τεχνολογίες που χρησιμοποιείς ώστε να διατυπώσεις ποιό είναι το καλύτερο για το σύστημα και να ξεχωρίσεις τι είναι πραγματική βελτίωση και τι είναι απλά θόρυβος, και αρκετές φορές χρειάστηκε να δεχτώ ότι μια αλλαγή η οποία φαινόταν ότι δούλευε στην πραγματικότητα δεν άλλαζε τίποτα. Τα περισσότερα από όσα έμαθα για το σύστημα σε αυτούς τους μήνες βγήκαν από τα πειράματα τα οποία έκανα όχι από τον κώδικα.

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Στέφανο Ουγιάρογλου για την καθοδήγηση και για τις παρατηρήσεις του σε κάθε στάδιο της εργασίας. Ευχαριστώ επίσης τους είκοσι εννέα ανθρώπους οι οποίοι μπήκαν στον κόπο να αφιερώσουν χρόνο στην δοκιμή της εφαρμογής και έπειτα στην συμπλήρωση του ερωτηματολογίου καθώς χωρίς αυτούς το έβδομο κεφάλαιο δεν θα υπήρχε στην διπλωματική. Τέλος να ευχαριστήσω την οικογένειά μου για την στήριξη σε όλη την διάρκεια των σπουδών μου.

Abstract

Anyone working through a collection of scientific articles wants an answer to a specific question rather than a summary of a whole text, yet a language model given that question answers just as fluently whether the answer is in the articles or not. This thesis designs and implements Z-AI Platform, a web application in which a user uploads their own scientific articles once and then asks questions over them in Greek or in English. Every answer is drawn only from those articles, the file and the page of each passage are shown next to it, and when the material does not cover the question the system says so instead of filling the gap from what the model already knows.

The retrieval path combines dense vector search with BM25, merges the two rankings with reciprocal rank fusion, rescores the result with a cross-encoder and blocks anything that falls below a relevance threshold, rewriting the question and retrying once before it refuses. Greek questions are translated into English with the help of a domain description and a vocabulary generated from the corpus itself.

The system was evaluated on two independent corpora at three levels, namely retrieval, an LLM judge and RAGAS, the last one scored by two different model families. Retrieval reaches an MRR of 0.793 with keyword coverage between 97.78% and 98.52%, faithfulness is 0.982 and 0.978 under the two judges while context recall is 1.000 under both, and the relevance gate refuses all five questions that the corpus cannot answer. Twenty-nine participants rated the interface at 84.74 on the System Usability Scale.

Keywords: retrieval-augmented generation, hybrid retrieval, cross-encoder reranking, grounded question answering, cross-lingual retrieval, scientific literature

Περίληψη

Όποιος δουλεύει πάνω σε μια συλλογή από επιστημονικά άρθρα θέλει απάντηση σε μια συγκεκριμένη ερώτηση και όχι σύνοψη από ολόκληρο το κείμενο για αυτό το γλωσσικό μοντέλο δεν είναι καλύτερη λύση γιατί θα του έδινε μια εξίσου πειστική απάντηση είτε η απάντηση υπάρχει στο υλικό που θέλει να ψάξει ο χρήστης είτε όχι. Στην εργασία αυτή σχεδιάστηκε και υλοποιήθηκε η Z-AI Platform μία εφαρμογή στην οποία ο χρήστης ανεβάζει μια φορά τα δικά του επιστημονικά άρθρα και ρωτάει στα ελληνικά ή στα αγγλικά επιλέγοντας ο ίδιος. Η κάθε απάντηση βγαίνει μόνο από τα άρθρα τα οποία έχει ανεβάσει ο ίδιος και από κάτω υπάρχουν οι παραπομπές μαζί με τα αρχεία και την σελίδα του κάθε αποσπάσματος και σε περίπτωση που το υλικό δεν καλύπτει την ερώτηση απαντάει με δεν ξέρω από το να συμπληρώσει το κενό με γνώσεις που κατέχει το ίδιο το μοντέλο.

Η διαδρομή της ανάκτησης ενώνει την διανυσματική αναζήτηση με το BM25, συγχωνεί τις δύο κατατάξεις με RRF κρατώντας τις πιο σημαντικές, τις οποίες ξαναβαθμολογεί με το cross-encoder και στη συνέχεια περνάει από το κατώφλι συνάφειας, και άμα δεν περάσει του δίνει πρώτα μια δεύτερη ευκαιρία με ξαναγραμμένη την ερώτηση ενώ άμα περάσει το κατώφλι φτάνει στο γλωσσικό μοντέλο ώστε να δημιουργήσει την απάντηση. Οι ελληνικές ερωτήσεις μεταφράζονται στην αρχή στα αγγλικά με την βοήθεια μιας περιγραφής του πεδίου και ενός λεξιλογίου τα οποία είναι από τα ίδια τα κείμενα που θα ψάξει για την ερώτηση.

Η αξιολόγηση έγινε σε δύο ανεξάρτητα σώματα κειμένων και σε τρία επίπεδα, την ανάκτηση, έναν κριτή LLM και το RAGAS με δύο διαφορετικές οικογένειες μοντέλων. Η ανάκτηση φτάνει MRR 0,793 με κάλυψη λέξεων κλειδιών από 97,78% έως 98,52%, η πιστότητα βγαίνει 0,982 και 0,978 στους δύο κριτές με ανάκληση συμφραζομένων 1,000 και στους δύο, ενώ η πύλη συνάφειας αρνείται και τις πέντε ερωτήσεις τις οποίες δεν καλύπτει το υλικό. Είκοσι εννέα συμμετέχοντες βαθμολόγησαν την εμπειρία τους με την εφαρμογή με 84,74 στην κλίμακα SUS.

Λέξεις κλειδιά: Ανάκτηση-Επαυξημένη Παραγωγή, υβριδική αναζήτηση, επανακατάταξη με cross-encoder, θεμελίωση της απάντησης, διαγλωσσική ανάκτηση, επιστημονικά άρθρα

Περιεχόμενα

1	Εισαγωγή	1
1.1	Επιστημονικά άρθρα και οργάνωση βιβλιογραφίας	1
1.2	Σύνοψη επιστημονικών άρθρων	1
1.3	Κίνητρο	2
1.4	Συνεισφορά	2
1.5	Οργάνωση της εργασίας	3
2	Ανάκτηση-Επαυξημένη Παραγωγή (RAG)	5
2.1	Τι είναι το RAG	5
2.2	Retriever και Generator	6
2.3	Κομμάτια (chunks), embeddings, διανυσματική βάση	6
2.4	Υβριδική αναζήτηση και επανακατάταξη	7
2.5	Θεμελίωση της απάντησης και παραπομπές	7
3	Γλώσσες και Τεχνολογίες	9
3.1	Γλώσσα και περιβάλλον ανάπτυξης	9
3.2	Frontend	10
3.3	Backend και API	10
3.4	Αποθήκευση δεδομένων	11
3.4.1	Σχεσιακή βάση	12
3.4.2	Διανυσματική βάση	12
3.5	Μοντέλα τεχνητής νοημοσύνης	12
3.6	Ανάκτηση και επεξεργασία κειμένου	13
3.7	Αξιολόγηση και έλεγχος	15
4	Σχεδίαση και υλοποίηση της εφαρμογής Z-AI Platform	17
4.1	Ανάλυση απαιτήσεων χρηστών (user stories)	17
4.2	Λειτουργικές απαιτήσεις (use cases/flow chart)	19
4.3	Αρχιτεκτονική του συστήματος	20
4.4	Υλοποίηση του backend	26
4.4.1	Η ανατομία ενός endpoint	26
4.4.2	Επεξεργασία	27
4.4.3	Η διαδρομή μιας ερώτησης	29
4.4.4	Συγχώνευση δύο κατατάξεων	30
4.4.5	Πύλη συνάφειας και δεύτερη προσπάθεια	32
4.4.6	Από τα κείμενα στην απάντηση	33
4.5	Υλοποίηση του frontend	35
4.5.1	Ένα σημείο επικοινωνίας με τον server	35

4.5.2	Επανεκτέλεση σελίδας και ροή απάντησης	36
4.5.3	Ο μετρητής που δεν παγώνει	38
4.6	Εργαλεία	39
4.6.1	Τι περιέχει το αποθετήριο	39
4.6.2	Έλεγχοι που τρέχουν σε κάθε ανέβασμα	40
4.7	Εγκατάσταση και εκτέλεση σε άλλον υπολογιστή	41
5	Παρουσίαση της εφαρμογής	43
5.1	Είσοδος στην εφαρμογή	43
5.2	Η οθόνη εργασίας και τα έγγραφα	43
5.3	Η απάντηση	45
5.3.1	Όταν υπάρχει απάντηση	45
5.3.2	Όταν δεν υπάρχει απάντηση	46
5.4	Οι πηγές και η αξιολόγηση	47
6	Αξιολόγηση	48
6.1	Τι μετρήθηκε και με τι	48
6.2	Ανάκτηση	50
6.2.1	Τι προσθέτει κάθε στάδιο	50
6.2.2	Το μέγεθος του τμήματος	50
6.2.3	Πού αποτυγχάνει	51
6.3	Το ερώτημα πριν την αναζήτηση	51
6.3.1	Το κόστος: διαρροή στις ερωτήσεις χωρίς υλικό	52
6.4	Ποιότητα των απαντήσεων	52
6.4.1	Ο πρώτος κριτής	52
6.5	Επιδόσεις	54
7	Αξιολόγηση ευχρηστίας από χρήστες	56
7.1	Το εργαλείο μέτρησης	56
7.2	Η διαδικασία	57
7.3	Ο υπολογισμός του σκορ	57
7.4	Αποτελέσματα	57
7.5	Τι δείχνουν οι επιμέρους ερωτήσεις	58
7.6	Περιορισμοί της μέτρησης	59
8	Συμπεράσματα	60
8.1	Ανακεφαλαίωση	60
8.2	Τι έδειξαν οι μετρήσεις	60
8.3	Τα όρια του συστήματος	61
8.4	Μελλοντικές επεκτάσεις	62
	Βιβλιογραφία	63

Κατάλογος εικόνων

4.1	Διάγραμμα περιπτώσεων χρήσης.	19
4.2	Αρχιτεκτονική του συστήματος.	23
4.3	Διάγραμμα οντοτήτων-συσχετίσεων της σχεσιακής βάσης.	25
5.1	Η οθόνη εισόδου, με τις δύο καρτέλες για σύνδεση και εγγραφή.	44
5.2	Ολόκληρη η οθόνη εργασίας. Αριστερά η στήλη διαχείρισης με τα ύφη απάντησης, το ιστορικό και τη βιβλιοθήκη των εγγράφων, δεξιά η συζήτηση και το πεδίο της ερώτησης.	45
5.3	Δύο ερωτήσεις στην ίδια συζήτηση. Επάνω, απάντηση στα ελληνικά με βάση αγγλικά έγγραφα, με τις παραπομπές μέσα στο κείμενο. Κάτω, ερώτηση που δεν καλύπτεται από τα έγγραφα: το σύστημα δηλώνει ότι δεν βρήκε την πληροφορία αντί να συνθέσει κείμενο χωρίς στήριγμα.	46
5.4	Το πλαίσιο των πηγών ανοιχτό. Κάθε παραπομπή δίνει αρχείο, σελίδα και απόσπασμα του πρωτοτύπου. Εικονίζονται οι πέντε πρώτες από τις οκτώ.	47
6.1	MRR ανά κατηγορία ερώτησης, στο πλήρες pipeline.	51
6.2	Μετρικές RAGAS στις 45 ερωτήσεις εντός corpus.	53

Κατάλογος πινάκων

4.1	Οι απαιτήσεις χρήστη.	17
4.2	Μη λειτουργικές απαιτήσεις.	18
4.3	Συνοπτική παρουσίαση των περιπτώσεων χρήσης.	20
4.4	Περίπτωση χρήσης UC-02: Ανέβασμα εγγράφου.	21
4.5	Περίπτωση χρήσης UC-05: Υποβολή ερώτησης.	22
4.6	Το σχεσιακό σχήμα της βάσης.	23
4.7	Τι αποθηκεύεται για κάθε κομμάτι κειμένου στη βάση διανυσμάτων.	25
6.1	Τα σετ αξιολόγησης που χρησιμοποιούνται στο κεφάλαιο αυτό.	49
6.2	Θετική σκάλα: τι προσθέτει κάθε στάδιο (45 ερωτήσεις εντός corpus, 5 εκτός).	50
6.3	Μέγεθος τμήματος στα δύο σετ ερωτήσεων. Το πρόσημο αντιστρέφεται.	50
6.4	Τρεις διατυπώσεις της ίδιας πληροφορίας (20 ερωτήσεις εντός corpus, 5 εκτός).	52
6.5	Οι ίδιες 45 ερωτήσεις, όπως τις βαθμολόγησαν δύο διαφορετικές οικογένειες μοντέλων. Αλλάζει μόνο ο κριτής· το dataset είναι το ίδιο.	53
6.6	Ανάκτηση υπό ταυτόχρονο φορτίο. Μετριέται μόνο το σκέλος που δεσμεύει CPU, χωρίς τη γέννηση.	54
7.1	Συγκεντρωτικά αποτελέσματα του SUS, $n = 29$	57
7.2	Μέση συνεισφορά ανά ερώτηση, μετά την αντιστροφή των αρνητικών. Κλίμακα 0–4, όπου 4 είναι το καλύτερο.	58

Κεφάλαιο 1

Εισαγωγή

1.1 Επιστημονικά άρθρα και οργάνωση βιβλιογραφίας

Η επιστήμη προοδεύει με τις δημοσιεύσεις, δηλαδή με άρθρα στα οποία γράφονται οι ανακαλύψεις, τα οποία κρίνονται από άλλους ερευνητές και μπαίνουν στην βιβλιογραφία ώστε η γνώση να γίνει κοινή για όλους. Τα πιο πολλά επιστημονικά άρθρα είναι σε μορφή PDF και ο λόγος είναι ότι η μορφή αυτή κρατάει σταθερή την σελιδοποίηση με στήλες, σχήματα, λεζάντες και βιβλιογραφία στο τέλος, φτιαγμένη για να την διαβάσει άνθρωπος και όχι κάποια μηχανή.

Είναι ένα πρόβλημα που επηρεάζει και εμάς γιατί ένας άνθρωπος που γράφει την διπλωματική του ή κάνει μια έρευνα για ένα συγκεκριμένο θέμα πρέπει να χρησιμοποιήσει και να διαβάσει πολλά PDF που προέρχονται από επιστημονικά άρθρα ώστε να στηρίξει σε αυτά όσα γράφει. Υπάρχει ένας τρόπος αλλά αντιμετωπίζει και αυτός ένα πρόβλημα. Ο τρόπος αυτός είναι να δώσεις σε κάθε αρχείο ένα όνομα, ή να χρησιμοποιήσεις προγράμματα όπως το Zotero που σου κρατάνε τίτλο, συγγραφέα και την χρονιά που δημοσιεύτηκε. Το πρόβλημα με αυτό όμως είναι ότι ξέρουν μόνο στοιχεία γύρω από το άρθρο όπως ο τίτλος και όχι το περιεχόμενό του οπότε πολλές φορές δεν βρίσκεις αυτό που ψάχνεις επειδή ο συγγραφέας χρησιμοποίησε άλλη λέξη για το ίδιο ακριβώς πράγμα [1] και ακόμα και αν βρεθεί, η αναζήτηση με λέξη είναι αδύναμος τρόπος γιατί βρίσκει την λέξη και όχι αναγκαστικά την απάντηση σε αυτό που ψάχνεις. Και αυτό γιατί μια λέξη μπορεί να εμφανίζεται σε όλες τις σελίδες του άρθρου και έτσι φτάνεις στο σημείο που ήθελες να αποφύγεις, να τα διαβάσεις όλα για να την βρεις. Είναι σημαντικό γιατί όταν χρησιμοποιείς ένα άρθρο πρέπει να ξέρεις πού βρίσκεται η απάντηση και σε ποια σελίδα. Και με αυτό φτάνουμε στο σημείο που χρειαζόμαστε κάτι που να καταλαβαίνει το περιεχόμενο και όχι μόνο τις λέξεις.

1.2 Σύνοψη επιστημονικών άρθρων

Το θέμα γενικότερα με τα επιστημονικά άρθρα δεν είναι μόνο να τα βρεις αλλά υπάρχει και μια δεύτερη ανάγκη η οποία είναι να πάρεις όσο πιο γρήγορα γίνεται αυτό που γράφει μέσα και ο λόγος είναι ότι δεν προλαβαίνεις να διαβάσεις κάθε άρθρο ολόκληρο, οπότε θέλεις την σύνοψή του η οποία άλλοτε είναι μία δύο προτάσεις για να καταλάβεις λίγο περί τίνος πρόκειται και άλλοτε πιο αναλυτική για μια ολόκληρη ενότητα. Σήμερα χρησιμοποιείται η περίληψη η οποία υπάρχει στην αρχή κάθε άρθρου κάτι το οποίο μας βοηθάει να καταλάβουμε με τι ασχολείται αλλά είναι μικρή, περιορισμένη και δεν δείχνει όλη την εικόνα γιατί την γράφει ο συγγραφέας για να δείξει την καλή πλευρά αφήνοντας έξω τους

περιορισμούς και τις λεπτομέρειες που σε ενδιαφέρουν ενώ πολλές φορές δεν περιέχει καθόλου την απάντηση που θέλεις. Τις περισσότερες φορές αναζητείς μια σύνοψη η οποία θα απαντάει στην ερώτησή σου είτε είναι κάποια μέθοδος είτε κάποιος πίνακας αποτελεσμάτων είτε κάποια ανάλυση για να κατανοήσεις πώς συνδέεται με το θέμα που έχεις εσύ. Είναι πράγματα τα οποία δεν μπορεί να στα δώσει η περίληψη του άρθρου γιατί είναι ίδια για όλους και δεν προσαρμόζεται σε αυτό που εσύ ψάχνεις. Όταν γράφεις μια διπλωματική δεν σε ενδιαφέρει μόνο ένα άρθρο αλλά χρειάζεσαι την γνώση και το περιεχόμενο από πολλά που σχετίζονται με το ίδιο θέμα δηλαδή θέλεις την σύνοψη που αναφέρθηκε πιο πριν να ενώνει πληροφορίες από πολλά άρθρα μαζί κάτι που μια περίληψη πάλι δεν μπορεί να στη δώσει.

Όλα αυτά που αναφέρθηκαν μπορεί να στα δώσει μια αυτόματη σύνοψη γλιτώνοντας πολύ χρόνο αλλά υπάρχει ένας κανόνας που δεν πρέπει να σπάσει γιατί το λάθος της περνάει στην δική σου δουλειά και την κάνει αναξιόπιστη. Πρέπει να σου δίνει την γνώση η οποία προέρχεται μόνο από τα άρθρα που θέλεις και όχι να βγάζει δικά της συμπεράσματα και αποτελέσματα και αυτό γιατί όσα προσθέτει από μόνη της μπορεί να ακούγονται σωστά αλλά είναι επικίνδυνα αφού δεν υπάρχουν πουθενά στα άρθρα.

1.3 Κίνητρο

Όπως αναφέρθηκε και προηγουμένως χρειαζόμαστε ένα εργαλείο που μπορεί να καταλάβει το περιεχόμενο των άρθρων, να απαντά σε αυτό που το ρωτάμε και η απάντησή του να είναι βασισμένη σε ό,τι υπάρχει στα άρθρα και όχι σε δικά του συμπεράσματα.

Τα τελευταία χρόνια δοκιμάζεται μια νέα λύση η οποία είναι τα μεγάλα γλωσσικά μοντέλα όπως το ChatGPT και το Gemini, στα οποία στέλνεις τα PDF μαζί με την ερώτησή σου και σου απαντούν με απλά λόγια κάτι που τα κάνει να φαίνονται ιδανικά.

Έχουν όμως προβλήματα που δεν επιτρέπουν στον χρήστη να τα εμπιστευτεί, διότι δίνουν απαντήσεις οι οποίες ακούγονται σωστές αλλά δεν είναι αληθινές επειδή απαντούν με βάση και όσα έμαθαν στην εκπαίδευσή τους και όχι μόνο με βάση το άρθρο που τους δίνεις. Αυτό λέγεται ψευδαίσθηση [2] και δεν φαίνεται με μια πρώτη ματιά και για να γίνει αντιληπτό θα πρέπει ο χρήστης να διαβάσει το άρθρο ώστε να διαπιστώσει αν όσα του απάντησε υπάρχουν πράγματι μέσα σε αυτό. Ένα δεύτερο θέμα είναι ότι ενώ το να ανεβάσεις ένα δύο αρχεία δουλεύει, το ίδιο δεν ισχύει για μια ολόκληρη βιβλιοθήκη με δεκάδες άρθρα [3] και αναγκάζεσαι κάθε φορά που ανοίγεις καινούργια συνομιλία να τα ξαναανεβάσεις. Και πάλι δεν σου λέει με σιγουριά από πού μέσα στο άρθρο βρίσκει την πληροφορία ώστε να μπορείς να την επαληθεύσεις.

Για τους λόγους που προαναφέρθηκαν χρειάζεται ένα εργαλείο φτιαγμένο ειδικά για αυτήν την δουλειά δηλαδή ένα εργαλείο χτισμένο πάνω στην τεχνική της ανάκτησης-επαυξημένης παραγωγής (RAG) [4] το οποίο κρατάει όλη την βιβλιοθήκη μαζί, απαντά στηριγμένο μόνο σε αυτήν και δείχνει από πού πήρε την κάθε πληροφορία ώστε να μπορεί ο χρήστης να την επαληθεύσει, δουλεύοντας παράλληλα και στα ελληνικά αφού τα άρθρα είναι κατά κύριο λόγο στα αγγλικά.

1.4 Συνεισφορά

Με βάση τα παραπάνω στην εργασία αυτή σχεδιάστηκε και υλοποιήθηκε ένα εργαλείο το οποίο δίνει απαντήσεις στα προβλήματα αυτά, δηλαδή πρόκειται για την Z-AI Platform στην οποία ο χρήστης εγγράφεται, κάνει την σύνδεσή του και ανεβάζει τα άρθρα του τα

οποία είναι ιδιωτικά και τα βλέπει μόνο αυτός, ρωτάει την ερώτησή του στα ελληνικά ή στα αγγλικά και το σύστημα απαντάει στηριζόμενο μόνο σε αυτά τα άρθρα.

Πιο συγκεκριμένα οι συνεισφορές της εργασίας είναι οι εξής:

- Μια εφαρμογή η οποία κρατάει όλη την βιβλιοθήκη του χρήστη μαζί χωρίς να χρειάζεται να την ανεβάζει ξανά σε κάθε ερώτηση και με την αναζήτηση να γίνεται πολύ γρήγορα σε κάθε άρθρο επειδή το κείμενο έχει ήδη επεξεργαστεί και είναι αποθηκευμένο.
- Απάντηση στηριγμένη αυστηρά στο κείμενο το οποίο έχει επιλεγθεί χωρίς προσθήκες οι οποίες δεν υπάρχουν εκεί και προέρχονται από τις γνώσεις του μοντέλου.
- Όταν η πληροφορία δεν υπάρχει στα άρθρα που έχουν επιλεγεί επιλέγει να απαντήσει με δεν ξέρω αντί να επινοήσει την απάντηση.
- Παραπομπές σε κάθε απάντηση μαζί με την σελίδα στην οποία βρίσκεται το κείμενο ώστε ο χρήστης να μπορεί να την επαληθεύσει.
- Η υποστήριξη της ελληνικής γλώσσας που δίνει την δυνατότητα στον χρήστη να ρωτάει και να παίρνει απάντηση στα ελληνικά ενώ τα άρθρα είναι στα αγγλικά.
- Η πειραματική αξιολόγηση του συστήματος σε δύο ανεξάρτητα σώματα κειμένων, τόσο ως προς την ανάκτηση όσο και ως προς την ποιότητα των απαντήσεων με δύο διαφορετικούς κριτές.

1.5 Οργάνωση της εργασίας

Η εργασία είναι οργανωμένη σε οκτώ κεφάλαια. Το πρώτο κεφάλαιο είναι η εισαγωγή στο θέμα στην οποία αναφέρεται ο προβληματισμός για το θέμα των επιστημονικών εγγράφων, το κίνητρο που χρειάστηκε ώστε να υλοποιηθεί το σύστημα και η συνεισφορά της εργασίας στην λύση του προβλήματος.

Στο δεύτερο κεφάλαιο περιγράφεται η τεχνολογία RAG θεωρητικά και ποιες λειτουργίες διαθέτει.

Στο τρίτο κεφάλαιο παρουσιάζονται οι γλώσσες προγραμματισμού, τα βασικά εργαλεία και βιβλιοθήκες που χρησιμοποιήθηκαν για να δημιουργηθεί η εφαρμογή αφιερώνοντας μια ενότητα για το καθένα.

Στο τέταρτο κεφάλαιο αναλύεται η σχεδίαση και υλοποίηση της εφαρμογής, από τις απαιτήσεις των χρηστών και την λειτουργία του συστήματος μέχρι την αρχιτεκτονική του και τον κώδικα του backend και του frontend.

Στο πέμπτο κεφάλαιο γίνεται η παρουσίαση της διεπαφής της εφαρμογής Z-AI Platform μέσα από εικόνες και παραδείγματα χρήσης της ώστε να μάθει ένας απλός χρήστης πώς μπορεί να την χρησιμοποιήσει.

Στο έκτο κεφάλαιο γίνεται η πειραματική μελέτη ώστε να αποδειχθεί για ποιο λόγο χρησιμοποιήθηκε η κάθε λειτουργία αλλά και να διερευνηθεί το πόσο αξιόπιστες είναι οι απαντήσεις και πόσο καλά βρίσκει την σωστή απάντηση.

Στο έβδομο κεφάλαιο μέσα από ένα ερωτηματολόγιο το οποίο έγινε και δόθηκε σε είκοσι εννέα άτομα να το δοκιμάσουν εξετάζεται η εμπειρία του χρήστη από την χρήση της εφαρμογής.

Τέλος στο όγδοο κεφάλαιο γίνεται η ανακεφαλαίωση της εργασίας και συγκεντρώνονται τα συμπεράσματα από τις μετρήσεις, δηλαδή τι κέρδισε το σύστημα από το κάθε στάδιο της ανάκτησης και τι από την υποστήριξη των ελληνικών. Στη συνέχεια καταγράφονται τα όρια του συστήματος, ο κορεσμός ο οποίος έρχεται στους τέσσερις ταυτόχρονους χρήστες και οι ερωτήσεις οι οποίες ζητάνε ολόκληρη λίστα από ένα άρθρο. Το κεφάλαιο κλείνει με τις μελλοντικές επεκτάσεις, τους ελεύθερους γύρους ανάκτησης, ένα σχήμα αναδιάταξης το οποίο δεν χρειάζεται να κόψει το τμήμα στα 512 tokens και την κλίμακα πέρα από τους τέσσερις χρήστες με κάρτα γραφικών ή με ξεχωριστή υπηρεσία.

Κεφάλαιο 2

Ανάκτηση-Επαυξημένη Παραγωγή (RAG)

2.1 Τι είναι το RAG

Το RAG (Retrieval-Augmented Generation) είναι μια τεχνική την οποία πρότειναν οι Lewis κ.ά. [4] και συνδυάζει δύο πράγματα που υπήρχαν ήδη αλλά δούλευαν ξεχωριστά, την αναζήτηση πληροφορίας μέσα στα έγγραφα που ανεβάζει ο χρήστης και ένα γλωσσικό μοντέλο που παίρνει τις πληροφορίες και δημιουργεί την απάντηση σε φυσική γλώσσα.

Το πρόβλημα που αντιμετωπίζει ένα γλωσσικό μοντέλο είναι ότι ξέρει μόνο όσα έχει δει κατά την εκπαίδευσή του, με αποτέλεσμα ακόμη και αν του δοθεί ένα άρθρο και ρωτηθεί για αυτό, στην απάντηση να προσθέσει εκτός από τα κείμενα του άρθρου και δικές του γνώσεις. Έτσι δεν μπορεί να ξέρει κάποιος εάν η απάντηση που του δόθηκε βασίζεται όντως στα κείμενα που του έχει δώσει ούτε από πού προήλθε. Επίσης όταν δεν υπάρχει η απάντηση στο άρθρο το οποίο του δόθηκε αντί να πει ότι δεν υπάρχει κάτι επινοεί την απάντηση και την συμπληρώνει πάλι με δικές του γενικές γνώσεις.

Η ιδέα του RAG έρχεται να λύσει αυτό ακριβώς το πρόβλημα και είναι σχετικά απλή, χωρίζοντας την διαδικασία σε δύο βήματα. Το σύστημα στην αρχή ψάχνει μέσα σε όλη την συλλογή των εγγράφων και επιλέγει μόνο τα αποσπάσματα που έχουν κάποια σχέση με την ερώτηση που έχει γίνει χωρίς να υπάρχει περιορισμός όσα και να είναι τα άρθρα. Έπειτα δίνει τα αποσπάσματα που έχουν ανακτηθεί μαζί με την ερώτηση που έχει γίνει για να δημιουργήσει την απάντηση βασιζόμενο πάνω σε αυτά μόνο και έτσι δεν αφήνεται το μοντέλο να ανακατέψει την γνώση αλλά κλειδώνεται η απάντηση στα κείμενα που του δόθηκαν.

Με αυτήν την τεχνική η απάντηση δεν προέρχεται από την εκπαίδευση που μπορεί να έχει ένα γλωσσικό μοντέλο αλλά από το πραγματικό περιεχόμενο που έχουν τα επιλεγμένα κείμενα που ανακτήθηκαν για την απάντηση. Αυτό εξασφαλίζει δύο από τα πράγματα που αναφέρθηκαν στις ανάγκες του πρώτου κεφαλαίου, πρώτον ότι οι απαντήσεις στηρίζονται σε πηγές οι οποίες υπάρχουν ώστε ο χρήστης να δει από ποιο άρθρο και από ποια σελίδα προέρχεται κάθε πληροφορία, και δεύτερον την δυνατότητα του γλωσσικού μοντέλου να απαντήσει «δεν ξέρω» όταν η απάντηση δεν υπάρχει στα σχετικά κείμενα.

Το πώς ακριβώς γίνεται κάθε βήμα, δηλαδή η αποθήκευση των άρθρων και η διαδρομή από την ερώτηση μέχρι την απάντηση, αναλύεται στις επόμενες ενότητες του κεφαλαίου.

2.2 Retriever και Generator

Το RAG αποτελείται από δύο βασικά μέρη που συνεργάζονται μεταξύ τους, τον Retriever (ανακτητή) και τον Generator (παραγωγό), οι οποίοι έχουν δικό τους ξεχωριστό ρόλο στην υλοποίηση του RAG.

Ο Retriever είναι το μέρος το οποίο παίρνει την ερώτηση του χρήστη και ψάχνει σε όλη την συλλογή ώστε να βρει τα άρθρα που σχετίζονται με αυτήν και να γυρίσει πίσω μόνο τα πιο σχετικά αποσπάσματα. Ο ρόλος του Retriever δεν είναι να γράψει την απάντηση αλλά είναι να βρίσκει το σχετικό κείμενο για την απάντηση και να το κατατάσσει από το πιο σχετικό στο λιγότερο σχετικό.

Ο Generator είναι το γλωσσικό μοντέλο βασισμένο στην αρχιτεκτονική των Transformers [5] το οποίο παίρνει την ερώτηση μαζί με τα αποσπάσματα τα οποία έχει εντοπίσει ο Retriever και συνθέτει την απάντηση. Το γλωσσικό μοντέλο σε αυτό το σύστημα έχει έναν πιο περιορισμένο ρόλο γιατί δεν απαντάει με τις γνώσεις τις οποίες διαθέτει αλλά μόνο με το υλικό το οποίο του έχει δοθεί.

Έτσι ενώ οι δύο ρόλοι συνεργάζονται είναι και ξεκάθαρα χωρισμένοι, και αυτό οφείλεται στο ότι ο Retriever επιλέγει ποια πληροφορία θα ανακτηθεί ενώ ο Generator αποφασίζει το πώς θα διατυπωθεί η απάντηση πάνω σε αυτήν. Αυτός ο διαχωρισμός είναι που δίνει την δυνατότητα στο σύστημα να έχει τις παραπομπές από τα σχετικά κείμενα γιατί γνωρίζει ποια αποσπάσματα μπήκαν στην απάντηση, αλλά και το «δεν ξέρω» του Generator που δεν έχει να στηριχθεί κάπου αφού ο Retriever δεν του έχει φέρει κάτι σχετικό.

Η ροή από το κομμάτι της ερώτησης μέχρι το κομμάτι της απάντησης είναι η εξής: η ερώτηση πηγαίνει πρώτα στον Retriever, ο οποίος επιστρέφει αποσπάσματα από τα επιλεγμένα αρχεία και έπειτα τα συγκεκριμένα αποσπάσματα μαζί με την ερώτηση πάνε στον Generator ο οποίος παράγει την τελική απάντηση.

2.3 Κομμάτια (chunks), embeddings, διανυσματική βάση

Το πρώτο πράγμα που κάνει ο χρήστης είναι να ανεβάσει τα άρθρα πριν κάνει την ερώτησή του. Ένα ολόκληρο άρθρο είναι πολύ μεγάλο ώστε να το ψάξει ο χρήστης ή να το δώσει ολόκληρο στο γλωσσικό μοντέλο για αυτό και σπάει σε μικρότερα κομμάτια (Chunks) όπου το καθένα είναι ένα απόσπασμα κειμένου που αναζητείται ξεχωριστά.

Στην συνέχεια κάθε κομμάτι μετατρέπεται σε ένα διάνυσμα (Embedding) το οποίο είναι μια λίστα από αριθμούς που περιγράφει το νόημά του και όχι τις ίδιες τις λέξεις [6] και η ιδέα γύρω από αυτό είναι ότι τα κομμάτια τα οποία έχουν παρόμοιο νόημα έχουν διανύσματα που βρίσκονται κοντά το ένα στο άλλο ενώ εάν είναι άσχετα πέφτουν μακριά.

Αυτή την μετατροπή του κειμένου σε διανύσματα την κάνει ένα ειδικό μοντέλο το οποίο ονομάζεται το μοντέλο ενσωματώσεων (Embedding model). Η ερώτηση και τα αποσπάσματα των άρθρων τοποθετούνται στον ίδιο διανυσματικό χώρο ώστε να μπορούν να συγκριθούν μεταξύ τους [7].

Όλα αυτά τα διανύσματα που δημιουργούνται αποθηκεύονται σε μια ειδική βάση δεδομένων, την διανυσματική βάση (Vector Database), της οποίας η δουλειά είναι να βρίσκει πολύ γρήγορα ποια από τα αποθηκευμένα διανύσματα είναι πιο κοντά σε αυτό της ερώτησης [8], κάτι το οποίο δίνει την δυνατότητα στο σύστημα να ψάχνει πολύ γρήγορα όλη την συλλογή ακόμα και αν τα άρθρα είναι πολλά και μεγάλα σε μέγεθος.

Έτσι δημιουργούνται δύο φάσεις από τις οποίες η πρώτη είναι όταν ανεβαίνει το άρθρο το οποίο στην αρχή χωρίζεται σε κομμάτια από τα οποία κάθε κομμάτι μετά μετατρέπεται σε διάνυσμα και τέλος αποθηκεύονται στην διανυσματική βάση. Η δεύτερη φάση είναι

όταν ο χρήστης κάνει μια ερώτηση η οποία και αυτή μετατρέπεται σε διάνυσμα και με την βοήθεια της βάσης βρίσκει τα πιο κοντινά διανύσματα και επιστρέφει τα αντίστοιχα αποτελέσματα στον Retriever.

2.4 Υβριδική αναζήτηση και επανακατάταξη

Στην προηγούμενη ενότητα περιγράφηκε η διαδικασία με την οποία το σύστημα βρίσκει τα αποσπάσματα με βάση το νόημά τους, κάτι που όμως δεν είναι αρκετό από μόνο του όταν έρθει η ερώτηση. Η αναζήτηση με βάση το νόημα είναι καλή ώστε να πιάνει γενικές έννοιες με βάση την ερώτηση που γίνεται αλλά χάνει πολύ στο κομμάτι να βρει συγκεκριμένους όρους, όπως ένα ακριβές όνομα, έναν αριθμό, έναν τεχνικό ή σπάνιο όρο ή ακόμα και έναν κωδικό και ο λόγος που συμβαίνει αυτό είναι γιατί αυτή η μέθοδος βασίζεται στο νόημα και όχι στην ίδια την λέξη.

Για αυτό και στο κομμάτι της ερώτησης χρησιμοποιείται ένας δεύτερος τρόπος αναζήτησης, η λεξική αναζήτηση, η οποία ψάχνει για τις λέξεις που υπάρχουν στην ερώτηση μέσα στα αντίστοιχα κείμενα προσπαθώντας να βρει συγκεκριμένους όρους [9]. Είναι πολύ δυνατή σε ένα κομμάτι στο οποίο υστερεί η σημασιολογική αναζήτηση δηλαδή σε ονόματα και σπάνιους όρους ενώ από την άλλη δυσκολεύεται όταν η ερώτηση χρησιμοποιεί διαφορετικές λέξεις από το κείμενο όπως τα συνώνυμα.

Επειδή η κάθε αναζήτηση έχει τα δυνατά και τα αδύνατα σημεία δεν επιλέγεται ένας από τους δύο τρόπους αλλά ο συνδυασμός αυτών των δύο ο οποίος ονομάζεται υβριδική αναζήτηση όπου τρέχουν και οι δύο και η καθεμία από αυτές δίνει μια ξεχωριστή κατάταξη των αποσπασμάτων και έτσι η μία καλύπτει τις αδυναμίες της άλλης.

Στην συνέχεια πρέπει να ενωθούν αυτές οι δύο διαφορετικές κατατάξεις με τρόπο ο οποίος θα μετράει την θέση και όχι το σκορ που έχει το καθένα και ο λόγος είναι ότι το σκορ της κάθε μεθόδου είναι διαφορετικό. Η σημασιολογική αναζήτηση το βγάζει με βάση το πόσο κοντά στο νόημα είναι το απόσπασμα με την ερώτηση ενώ η λεξική με βάση το πόσο συχνά εμφανίζονται οι λέξεις της ερώτησης μέσα στο απόσπασμα δίνοντας μεγαλύτερο βαθμό σε αυτές οι οποίες είναι πιο σπάνιες. Αυτό γίνεται με μια απλή μέθοδο η οποία ονομάζεται RRF (Reciprocal Rank Fusion) [10] η οποία τα επιλέγει βάσει θέσης και όχι σκορ δηλαδή όποια αποσπάσματα εμφανίζονται ψηλά και στις δύο λίστες τους δίνει μεγαλύτερη βαρύτητα και με αυτόν τον τρόπο φτιάχνει μια τελική ενιαία κατάταξη.

Τέλος αφού έχουν μαζευτεί τα υποψήφια αποσπάσματα γίνεται και ένα ακόμη βήμα το οποίο ονομάζεται επανακατάταξη (Reranking) και χρησιμοποιεί ένα πιο ακριβές μοντέλο το οποίο δεν διαλέγει βάσει θέσης αλλά διαβάσει το ίδιο το περιεχόμενο των αποσπασμάτων. Ο cross-encoder [11] παίρνει την ερώτηση του χρήστη και το κάθε απόσπασμα που έχει έρθει από το RRF και τα ξαναβαθμολογεί πιο προσεκτικά και αναλυτικά αυτήν την φορά. Έπειτα αφού βαθμολογήσει το καθένα, τα ανακατατάσσει με βάση αυτήν την βαθμολογία και κρατάει τα καλύτερα τα οποία μετέπειτα δίνονται στον Generator ώστε να δημιουργήσει την απάντηση και με αυτόν τον τρόπο διασφαλίζεται ότι η τελική λίστα που φτάνει στο γλωσσικό μοντέλο έχει όλα τα στοιχεία και είναι όσο πιο σχετική γίνεται.

2.5 Θεμελίωση της απάντησης και παραπομπές

Το πιο σημαντικό κομμάτι του RAG είναι η θεμελίωση της απάντησης, που σημαίνει ότι ο Generator απαντά με βάση μόνο τα αποσπάσματα που του έχει δώσει ο Retriever και όχι με βάση τις δικές του γνώσεις και με αυτόν τον τρόπο η απάντηση βασίζεται πάντα στο

περιεχόμενο το οποίο υπάρχει στα άρθρα και όχι στις γνώσεις που έχει μάθει το γλωσσικό μοντέλο κατά την εκπαίδευσή του.

Αυτό το αποτέλεσμα δεν γίνεται από μόνο του αλλά μέσα από τις οδηγίες που δίνονται στο μοντέλο μαζί με τα αποσπάσματα, όπου ζητείται ρητά από αυτό να απαντάει με βάση το υλικό που του έχει δοθεί και να μην προσθέτει δικές του πληροφορίες στην απάντηση και έτσι κλειδώνεται ότι η απάντηση θα είναι βασισμένη στα κείμενα που διαθέτουν τα άρθρα.

Επειδή το σύστημα γνωρίζει ποια ακριβώς αποσπάσματα βρίσκονται στην απάντηση μπορεί να δείξει στον χρήστη την πηγή της πληροφορίας και με αυτόν τον τρόπο ο χρήστης βλέπει από ποιο άρθρο και από ποια ακριβώς σελίδα προέρχεται κάθε στοιχείο της απάντησης, ώστε να μπορεί και ο ίδιος να την επαληθεύσει χωρίς να την πιστεύει τυφλά.

Υπάρχει όμως και η περίπτωση να μην βρίσκεται η απάντηση στα κείμενα που έχουν δοθεί ή ο Retriever να μην έχει βρει τα σωστά αποσπάσματα. Σε αυτήν την περίπτωση το σύστημα έχει την οδηγία να μην επινοήσει την απάντηση βάσει των δικών του γνώσεων και να απαντήσει «δεν ξέρω», κάτι το οποίο είναι σημαντικό γιατί έτσι αποφεύγεται μια απάντηση η οποία μπορεί να ακούγεται σωστή αλλά δεν στηρίζεται στα κείμενα που του έχουν δοθεί παρά σε δικές του γνώσεις [2].

Και τέλος αυτά τα τρία μαζί που αναφέρθηκαν πιο πριν, η θεμελίωση της απάντησης, οι παραπομπές στην πηγή και η δυνατότητα να απαντήσει ότι δεν γνωρίζει σε περίπτωση που δεν βρει κάτι στα κείμενα είναι που κάνουν το κομμάτι των ερωτήσεων του συστήματος αξιόπιστο. Έτσι καλύπτονται οι ανάγκες που περιγράφηκαν στο πρώτο κεφάλαιο και δίνεται λύση για το βασικό πρόβλημα που προσπαθεί να λύσει αυτό το σύστημα και είναι ο λόγος που ξεκίνησε αυτή η εργασία, δηλαδή το κομμάτι της ψευδαίσθησης.

Κεφάλαιο 3

Γλώσσες και Τεχνολογίες

3.1 Γλώσσα και περιβάλλον ανάπτυξης

Η γλώσσα που χρησιμοποιήθηκε σε όλη την εφαρμογή είναι η Python η οποία είναι και η βασική γλώσσα της τεχνητής νοημοσύνης και ο λόγος είναι ότι όλες οι βιβλιοθήκες που χρειάστηκαν για τα embeddings, για την διανυσματική βάση, για το σπάσιμο του κειμένου σε κομμάτια και την επικοινωνία με το Gemini είναι φτιαγμένες σε Python. Μια άλλη γλώσσα θα σήμαινε ότι τα εργαλεία αυτά θα έπρεπε να κληθούν έμμεσα ή να αντικατασταθούν και επιπλέον με την Python γράφτηκε το backend και ο πυρήνας του RAG και τα scripts που χρειάστηκαν ώστε να γίνει η αξιολόγηση δηλαδή βοήθησε να φτιαχτεί ολόκληρο το σύστημα με μόνο μία γλώσσα.

Οι ρυθμίσεις της εφαρμογής δεν γράφονται μέσα στον κώδικα αλλά σε ένα ειδικό και ξεχωριστό αρχείο .env και αυτό γίνεται για δύο ξεχωριστούς λόγους. Ο πρώτος είναι η ασφάλεια, γιατί στο .env μπαίνουν πληροφορίες που δεν θα έπρεπε να διαρρεύσουν, δηλαδή τα μυστικά της εφαρμογής. Το GEMINI_API_KEY είναι το κλειδί για την επαφή με το Gemini, το DATABASE_URL περιέχει τα στοιχεία σύνδεσης με την βάση δεδομένων και το SECRET_KEY είναι αυτό με το οποίο υπογράφονται τα tokens των χρηστών, τα οποία αποδεικνύουν ότι ο κάθε χρήστης είναι συνδεδεμένος στην εφαρμογή. Αν κάτι από όλα αυτά ήταν γραμμένο μέσα στον κώδικα και ανέβαινε στο GitHub ώστε να δουν και άλλοι τον τρόπο που δημιουργήθηκε ή και να το εγκαταστήσουν στην δική τους συσκευή θα μπορούσε να τα δει ο οποιοσδήποτε και να τα χρησιμοποιήσει και αυτός.

Ο δεύτερος λόγος είναι πιο απλός και είναι η ευελιξία που προσθέτει στο project, δηλαδή ότι στο ίδιο αρχείο μπαίνουν και ρυθμίσεις που στην διάρκεια του project μπορούν να αλλάξουν όπως το γλωσσικό μοντέλο που έχει χρησιμοποιηθεί και έτσι μπορεί αλλάζοντας μία γραμμή στις ρυθμίσεις να αλλάξει σε μεγαλύτερο ή μικρότερο μοντέλο ανάλογα με τις απαιτήσεις χωρίς να πειραχτεί ο κώδικας που υπάρχει μέσα στο project.

Η εφαρμογή τρέχει μέσα σε containers του Docker, κάτι που σημαίνει ότι όλες οι βιβλιοθήκες οι οποίες έχουν χρησιμοποιηθεί είναι κλειδωμένες σε συγκεκριμένες εκδόσεις ώστε το περιβάλλον να παραμένει ίδιο κάθε φορά που τρέχει το σύστημα. Το κέρδος είναι διπλό, από την μια όποιος κατεβάσει την εφαρμογή στο δικό του μηχάνημα δεν θα έχει πρόβλημα με τις διαφορετικές βιβλιοθήκες που μπορεί να έχει γιατί θα πάρει ακριβώς τις ίδιες εκδόσεις με αυτές που έχει το σύστημα και από την άλλη έχει σημασία για το κομμάτι της διπλωματικής γιατί τα πειράματα και οι μετρήσεις πρέπει να τρέξουν στις ίδιες εκδόσεις ώστε να μπορούν να επαναληφθούν και να δώσουν τα ίδια αποτελέσματα.

Τέλος ο κώδικας κρατιέται σε σύστημα ελέγχου εκδόσεων, το git. Αντί να αποθηκεύεται μόνο η τελευταία μορφή του κάθε αρχείου, καταγράφεται κάθε αλλαγή ξεχωριστά μαζί

με τον λόγο για τον οποίο έγινε, οπότε φαίνεται πώς έφτασε το σύστημα εδώ και μπορεί να γυρίσει πίσω σε προηγούμενη κατάσταση αν κάτι χαλάσει. Το GitHub είναι η υπηρεσία που φιλοξενεί αυτό το ιστορικό στο διαδίκτυο και δίνει δύο πράγματα παραπάνω: ο κώδικας γίνεται προσβάσιμος σε όποιον θέλει να τον δει, και μπορούν να τρέχουν αυτόματα έλεγχοι κάθε φορά που ανεβαίνει μια αλλαγή. Και τα δύο χρησιμοποιήθηκαν εδώ και περιγράφονται στην ενότητα του κεφαλαίου 4 που μιλάει για το αποθετήριο.

3.2 Frontend

Για την διεπαφή της εφαρμογής χρησιμοποιήθηκε μια βιβλιοθήκη της Python η οποία είναι ειδική στο κομμάτι του να φτιαχτεί μια ιστοσελίδα χωρίς να γράφεται κώδικας HTML, CSS ή JavaScript και ονομάζεται Streamlit. Ο πρώτος και πιο σημαντικός λόγος για τον οποίο επιλέχθηκε η συγκεκριμένη βιβλιοθήκη είναι γιατί είναι γραμμένη στην ίδια γλώσσα με το backend, οπότε ολόκληρο το σύστημα βασίζεται μόνο σε μία γλώσσα και μένει πιο απλό στην συντήρησή του. Αν αντίθετα είχε επιλεγεί ένα framework όπως η React θα χρειαζόταν παράλληλα και JavaScript, κάτι που θα αύξανε τον χρόνο ανάπτυξης χωρίς να προσφέρει κάτι στο κομμάτι του RAG.

Ο δεύτερος λόγος είναι ότι το Streamlit είναι φτιαγμένο ακριβώς για εφαρμογές συνομιλίας, γιατί έχει έτοιμο το πεδίο στο οποίο ο χρήστης θα κάνει τις ερωτήσεις του, τον διαχωρισμό ανάμεσα στο μήνυμα του χρήστη και σε αυτό που στέλνει η εφαρμογή, το κουμπί για το ανέβασμα των αρχείων και κυρίως την δυνατότητα να δίνεται σταδιακά η απάντηση. Αυτό είναι απαραίτητο γιατί το Gemini δεν στέλνει την απάντηση μονομιάς όταν τελειώσει αλλά κομμάτι κομμάτι, έτσι ώστε ο χρήστης να την βλέπει να σχηματίζεται στην οθόνη του.

Μαζί με τα θετικά που διαθέτει υπάρχει και ένα μειονέκτημα και αυτό είναι ότι δίνει λιγότερο έλεγχο στην εμφάνιση από ό,τι ένα κανονικό framework αλλά για τα δεδομένα της εργασίας δεν είναι πρόβλημα γιατί ο στόχος της διπλωματικής είναι μεν να δημιουργηθεί μια λειτουργική διεπαφή αλλά η έμφαση δίνεται περισσότερο στο κομμάτι του RAG.

Το frontend είναι το κομμάτι που επικοινωνεί με το backend μόνο μέσω HTTP με μια βιβλιοθήκη η οποία ονομάζεται requests, κάτι που σημαίνει ότι η διεπαφή δεν έχει καμία πρόσβαση στην βάση δεδομένων, δεν φορτώνει κανένα μοντέλο και δεν γνωρίζει κανένα από τα μυστικά κλειδιά που υπάρχουν, κερδίζοντας έτσι δύο πράγματα με αυτό. Από την μια είναι θέμα ασφάλειας γιατί όλα τα κλειδιά μένουν στην πλευρά του backend και δεν βγαίνουν προς τα έξω και από την άλλη είναι και θέμα μεγέθους, γιατί παρατηρήθηκε ότι όταν αφαιρέθηκαν από το frontend οι βιβλιοθήκες που δεν χρειαζόνταν το μέγεθος του frontend μίκρυνε κατά περίπου 2 GB.

3.3 Backend και API

Το backend της εφαρμογής είναι γραμμένο με το FastAPI, ένα framework της Python που χρησιμοποιείται στο κομμάτι της δημιουργίας API, δηλαδή των σημείων στα οποία απευθύνεται το frontend ώστε να ζητήσει να γίνει κάποια διεργασία. Ο βασικός λόγος που επιλέχθηκε είναι ότι δουλεύει ασύγχρονα κάτι το οποίο έχει μεγάλη σημασία για την συγκεκριμένη εφαρμογή γιατί ο server δεν μένει κολλημένος όσο περιμένει αλλά μπορεί να ασχοληθεί με άλλες δουλειές, αφού κάθε φορά που κάνει ο χρήστης μια ερώτηση ο server στέλνει το ερώτημα στο Gemini και περιμένει την απάντηση η οποία μπορεί να αργήσει κάποια δευτερόλεπτα. Στο διάστημα αυτό ο server δεν μένει στάσιμος αλλά εξυπηρετεί άλλους

χρήστες που μπορεί να υπάρχουν, βοηθάει με το ανέβασμα των αρχείων και μπορεί να ετοιμάζει την επόμενη αναζήτηση κάτι το οποίο δεν θα ήταν δυνατό αν το framework δούλευε σύγχρονα και όλος αυτός ο χρόνος θα πήγαινε χαμένος, γιατί θα ήταν δεσμευμένος μόνο με τον έναν χρήστη και οι υπόλοιποι θα έπρεπε να περιμένουν να τελειώσει πρώτα αυτός ώστε να ξεκινήσει τις δικές του διεργασίες.

Ένα ακόμα πλεονέκτημα το οποίο έχει το FastAPI είναι ότι ελέγχει μόνο του αν τα δεδομένα που έρχονται από το frontend είναι σωστά, χρησιμοποιώντας μια ακόμα βιβλιοθήκη της Python την Pydantic. Για κάθε αίτημα έχουν οριστεί από πριν οδηγίες για το τι πρέπει να περιέχει και αν λείπει κάτι ή έχει λάθος μορφή απορρίπτεται πριν φτάσει στην εφαρμογή, ενώ χωρίς αυτό θα έπρεπε να γίνονται χειροκίνητα όλοι αυτοί οι έλεγχοι σε κάθε σημείο που υπάρχει στον κώδικα.

Ένα πολύ σημαντικό κομμάτι στο οποίο δόθηκε πολλή προσοχή και σημασία είναι η ασφάλεια των λογαριασμών και ο τρόπος που έγινε είναι ότι οι κωδικοί δεν αποθηκεύονται ποτέ όπως τους γράφει ο χρήστης αλλά περνάνε από τον αλγόριθμο bcrypt [12] μέσω της βιβλιοθήκης passlib. Ο αλγόριθμος αυτός είναι μονόδρομος, που σημαίνει ότι από το αποτέλεσμα που αποθηκεύεται στην βάση δεν μπορεί να βρεθεί ο αρχικός κωδικός ούτε από όποιον έχει πρόσβαση σε αυτήν. Στο κομμάτι που συνδέεται ο χρήστης δεν συγκρίνεται ο κωδικός που έγραψε με κάποιον αποθηκευμένο κωδικό αλλά με το αποτέλεσμα που βγάζει ο ίδιος αλγόριθμος, ώστε ακόμα και αν διαρρεύσει η βάση δεδομένων οι κωδικοί των χρηστών να μην αποκαλύπτονται.

Αφού κάνει την σύνδεσή του ο χρήστης η εφαρμογή του παρέχει ένα token τύπου JWT το οποίο το δημιουργεί η βιβλιοθήκη python-jose και το υπογράφει με το μυστικό κλειδί της εφαρμογής. Το token αυτό στέλνεται μαζί με το κάθε αίτημα που κάνει ο χρήστης είτε για να γράψει κάποια ερώτηση είτε για να ανεβάσει ένα αρχείο ώστε να αποδεικνύεται η ταυτότητα του χρήστη χωρίς να χρειάζεται ο server να κρατάει πληροφορίες για το ποιος είναι συνδεδεμένος. Για λόγους ασφαλείας το token του κάθε χρήστη δεν ισχύει για πάντα αλλά παύει να ισχύει μετά από μία μέρα ώστε αν κλαπεί από κάποιον να μην έχει μεγάλο διάστημα στο οποίο θα είναι συνδεδεμένος.

Τέλος το FastAPI είναι αυτό που δίνει την δυνατότητα να σταλθεί σταδιακά η απάντηση αντί για μονομιάς και αυτό είναι σημαντικό ώστε να λειτουργήσει η σταδιακή εμφάνιση της απάντησης όπως αναφέρθηκε στην ενότητα 3.2 αλλά για να γίνει αυτό θα πρέπει να τα στείλει κομμάτι κομμάτι και το backend.

3.4 Αποθήκευση δεδομένων

Το σύστημα χρησιμοποιεί δύο διαφορετικές βάσεις δεδομένων και ο λόγος που γίνεται αυτό είναι ότι η καθεμία κάνει διαφορετικά πράγματα στο σύστημα, τα οποία δεν έχουν σχέση μεταξύ τους. Η πρώτη ασχολείται με το κομμάτι της ακρίβειας, όπως το ποιος είναι ο χρήστης με ένα συγκεκριμένο όνομα ή ποια από τα αρχεία ανήκουν σε αυτόν, δηλαδή ασχολείται περισσότερο με το κομμάτι της ταυτότητας του χρήστη και τι είναι δικό του. Η δεύτερη ασχολείται με το κομμάτι της ομοιότητας, όπως το ποια κομμάτια από αυτά που υπάρχουν μοιάζουν περισσότερο με το νόημα της ερώτησης του χρήστη δηλαδή υπάρχει μια κατάταξη με το πόσο κοντά ή μακριά βρίσκονται από την ερώτηση. Όλα αυτά δεν θα μπορούσε να τα καλύψει στο έπακρο μία βάση για αυτό και χρησιμοποιήθηκε για το πρώτο κομμάτι μια σχεσιακή βάση δεδομένων ενώ για το δεύτερο μια διανυσματική βάση δεδομένων.

3.4.1 Σχεσιακή βάση

Για τα δεδομένα του συστήματος χρησιμοποιήθηκε μια σχεσιακή βάση PostgreSQL, στην οποία αποθηκεύονται οι χρήστες, τα έγγραφα που έχει ανεβάσει ο καθένας, οι συνομιλίες, τα μηνύματα της κάθε συνομιλίας, η αξιολόγηση την οποία μπορεί να αφήσει ο χρήστης στις απαντήσεις και οι μετρητές που περιορίζουν τον αριθμό των αιτημάτων. Ο λόγος που επιλέχθηκε η συγκεκριμένη είναι επειδή τα δεδομένα αυτά έχουν κάποια σύνδεση μεταξύ τους, αφού κάθε έγγραφο ανήκει σε έναν χρήστη και κάθε μήνυμα ανήκει σε μια συνομιλία και η σχεσιακή βάση είναι φτιαγμένη για ακριβώς τέτοιες σχέσεις.

Η επικοινωνία με την βάση δεν γίνεται με απευθείας εντολές SQL αλλά μέσω της βιβλιοθήκης SQLAlchemy. Η βιβλιοθήκη δίνει την δυνατότητα οι πίνακες να θεωρούνται σαν κανονικές κλάσεις Python έτσι ώστε ο κώδικας να μένει σε μία γλώσσα για να αποφευχθούν τα πολλά λάθη αλλά υπάρχει και ακόμα ένας λόγος ο οποίος είναι ασφαλείας γιατί η βάση δέχεται εντολές γραμμένες σαν κείμενο. Αν μέσα στην εντολή μπει έτσι αυτούσιο αυτό που θα γράψει ο χρήστης τότε κάποιος που θέλει να επιτεθεί στο σύστημα μπορεί να γράψει στην φόρμα όχι το όνομά του αλλά μια δική του εντολή, και η βάση θα την εκτελέσει χωρίς να το σκεφτεί και με αυτόν τον τρόπο θα μπορούσε να μπει στην εφαρμογή χωρίς κωδικό ή και να σβήσει δεδομένα από την βάση. Για αυτό και χρησιμοποιείται η SQLAlchemy ώστε να στέλνει την εντολή και τα δεδομένα ξεχωριστά οπότε ό,τι και να γράψει ο χρήστης το βλέπει η βάση σαν απλό κείμενο και όχι σαν εντολή.

3.4.2 Διανυσματική βάση

Για την αποθήκευση των κειμένων χρησιμοποιήθηκε η ChromaDB στην οποία δεν αποθηκεύεται ολόκληρο το άρθρο αλλά τα κομμάτια στα οποία έχει σπάσει, μαζί με το διάνυσμα που δημιουργείται για το καθένα, δηλαδή τους αριθμούς που περιγράφουν το νόημά του και όταν της έρχεται μια ερώτηση μετατρέπεται και αυτή σε διάνυσμα και η βάση επιστρέφει μόνο αυτά που βρίσκονται πιο κοντά σε νόημα με αυτό.

Μαζί με το κάθε κομμάτι κειμένου αποθηκεύονται και κάποιες επιπλέον πληροφορίες που είναι εξίσου σημαντικές με το ίδιο το κείμενο. Οι πιο βασικές είναι το όνομα του αρχείου από το οποίο προέρχεται το κάθε απόσπασμα αλλά και η σελίδα στην οποία βρίσκεται κάτι το οποίο βοηθάει το σύστημα να δίνει τις παραπομπές στον χρήστη γιατί χωρίς αυτές θα ήξερε τι βρήκε αλλά όχι από πού. Τελευταίο αποθηκεύεται και ο χρήστης που ανέβασε το κάθε αρχείο έτσι ώστε όταν ρωτάει κάτι η αναζήτηση να γίνεται μόνο μέσα στα αρχεία που έχει δώσει ο ίδιος και να διασφαλίζεται επίσης ότι δεν μπορεί να δει κανείς το υλικό κάποιου άλλου.

3.5 Μοντέλα τεχνητής νοημοσύνης

Η εφαρμογή χρησιμοποιεί τρία διαφορετικά μοντέλα, το καθένα έχει έναν ξεχωριστό ρόλο και κάνει μια ξεχωριστή δουλειά στο project. Το πρώτο διατυπώνει την τελική απάντηση στον χρήστη, το δεύτερο μετατρέπει το κείμενο σε διανύσματα 1024 διαστάσεων και το τρίτο ξαναβαθμολογεί τα αποτελέσματα που του δίνει η αναζήτηση.

Για την δημιουργία της απάντησης χρησιμοποιήθηκε το Gemini 2.5 Flash και επιλέχθηκε το Flash αντί για το Pro και ο λόγος είναι ότι είναι πιο γρήγορο και πιο φθινό, ενώ ταυτόχρονα δέχεται περισσότερο κείμενο μαζί με την ερώτηση κάτι το οποίο είναι πολύ σημαντικό

όταν του έρχονται πολλά κομμάτια από τα άρθρα. Το ποιο μοντέλο θα χρησιμοποιηθεί ορίζεται στις ρυθμίσεις δίνοντας έτσι την δυνατότητα να αλλαχθεί σε μεγαλύτερο ή μικρότερο ανά πάσα στιγμή με μια αλλαγή χωρίς να αλλάξει κάτι στον κώδικα.

Μια σημαντική ρύθμιση που έχει γίνει στο μοντέλο είναι η θερμοκρασία η οποία έχει οριστεί στο 0,1. Αυτό που κάνει η θερμοκρασία είναι να ρυθμίζει την ελευθερία που δίνει στο μοντέλο να διαλέγει διαφορετικές λέξεις κάθε φορά που γίνεται η ίδια ερώτηση δηλαδή όσο πιο χαμηλά είναι τόσο πιο σταθερή θα είναι η απάντηση κάτι το οποίο χρειάζεται για τρεις λόγους. Ο πρώτος και πιο σημαντικός είναι ότι ο χρήστης θα πρέπει να παίρνει σταθερές απαντήσεις και όχι κάθε φορά διαφορετική, ο δεύτερος είναι ότι όσο μικρότερη ελευθερία διαθέτει το μοντέλο τόσο πιο απίθανο είναι να ξεφύγει από το κείμενο που του δόθηκε και να αρχίσει να σκέφτεται με τις δικές του γνώσεις και τρίτον είναι ότι έτσι τα πειράματα της αξιολόγησης που γίνονται μπορούν να επαναληφθούν δίνοντας τα ίδια αποτελέσματα και έχοντας σταθερά νούμερα στις μετρικές.

Η μετατροπή των κειμένων σε διανύσματα γίνεται με το μοντέλο bge-m3 [13] και ο βασικός λόγος που επιλέχθηκε είναι ότι είναι πολύγλωσσο το οποίο είναι απαραίτητο για το συγκεκριμένο σύστημα γιατί αν ο χρήστης ρωτήσει στα ελληνικά ενώ τα άρθρα είναι στα αγγλικά θα πρέπει το μοντέλο να καταλαβαίνει ότι η ερώτηση στα ελληνικά και το αγγλικό κείμενο μιλάνε για το ίδιο πράγμα. Ένα μοντέλο μίας γλώσσας δεν θα μπορέσει να κάνει αυτήν την σύνδεση ενώ με το bge-m3 διασφαλίζεται ότι τα τοποθετεί στον ίδιο χώρο, οπότε αρκεί ένα ευρετήριο και όχι δύο ώστε να καλυφθεί το κομμάτι των δύο γλωσσών.

Το τρίτο μοντέλο που χρησιμοποιήθηκε είναι ένας cross-encoder [11], ο οποίος παίρνει τα αποτελέσματα της αναζήτησης και τα ξαναβαθμολογεί διαβάζοντας την ερώτηση μαζί με το κάθε κομμάτι κειμένου που ήρθε. Εδώ επιλέχθηκε το ms-marco-MiniLM-L-12-v2 ένα μοντέλο που καταλαβαίνει μόνο αγγλικά και αυτό διότι το σύστημα όταν παρατηρήσει ελληνική ερώτηση την μεταφράζει στα αγγλικά πριν φτάσει στην αναζήτηση. Οπότε το μοντέλο βλέπει πάντα αγγλική ερώτηση σε αγγλικό κείμενο για αυτό και δεν χρειάστηκε ένα πολύγλωσσο μοντέλο που στην συγκεκριμένη περίπτωση θα ήταν πολύ αργό και μεγαλύτερο χωρίς να ωφελεί πουθενά. Δοκιμάστηκε και ένα μεγαλύτερο μοντέλο στην θέση αυτή, το bge-reranker-v2-m3, αλλά απορρίφθηκε και ο λόγος είναι ότι παρόλο που ήταν δεκαεπτά φορές μεγαλύτερο και χρειαζόταν είκοσι φορές περισσότερο χρόνο για κάθε ερώτηση, στα αποτελέσματα έδωσε μία ερώτηση παραπάνω σωστή και μία παραπάνω λανθασμένη οπότε το όφελός του ήταν μηδενικό. Δοκιμάστηκε και ένα μικρότερο μοντέλο στην θέση αυτή, το ms-marco-MiniLM-L-6-v2, αλλά και αυτό απορρίφθηκε και ο λόγος είναι ότι με την ίδια εγγύηση ότι το σύστημα δεν απαντάει σε ερωτήσεις εκτός corpus περνούσαν επτά από τις δεκαέξι δύσκολες ερωτήσεις ενώ με το L-12 περνάνε δέκα και το σωστό κομμάτι ερχόταν μία θέση πιο κάτω.

3.6 Ανάκτηση και επεξεργασία κειμένου

Πέρα από τα μοντέλα που υπάρχουν το σύστημα χρειάστηκε και μια σειρά από βιβλιοθήκες οι οποίες αναλαμβάνουν την διαδρομή την οποία ακολουθεί το κείμενο από την στιγμή την οποία ο χρήστης θα ανεβάσει ένα PDF μέχρι την στιγμή που κάποια από αυτά τα κομμάτια θα βρεθούν και θα σταλούν στο μοντέλο για να απαντήσει μια ερώτηση.

Το πρώτο βήμα είναι η εξαγωγή του κειμένου. Για αυτό και χρησιμοποιήθηκε η pymupdf. Στην αρχή δοκιμάστηκε η pypdf η οποία όμως έβαζε κενά μέσα στις λέξεις και μπορεί να έφτιαχνε λέξεις οι οποίες δεν υπάρχουν πουθενά όπως «A WS» αντί για «AWS» κάτι το οποίο είναι σοβαρό γιατί μια λέξη κομμένη στην μέση δεν θα βρεθεί ποτέ από την αναζήτηση. Για αυτό συγκρίθηκαν αυτές οι δύο βιβλιοθήκες σε ολόκληρο το υλικό που υπήρχε

δηλαδή επτά άρθρα και τα αποτελέσματα έδειξαν ότι τα σπασμένα κείμενα έπεσαν από 3,7% σε 2,5% κάνοντας επίσης το κομμάτι της εξαγωγής κειμένου τέσσερις φορές πιο γρήγορο.

Ακόμα και έτσι όμως το κείμενο που βγαίνει από ένα PDF χρειάζεται να καθαριστεί, κάτι που έγινε με κώδικα γραμμένο ειδικά για την εφαρμογή. Το πρώτο πρόβλημα είναι ότι τα PDF αποθηκεύουν κάποιους συνδυασμούς γραμμμάτων σαν έναν χαρακτήρα, για παράδειγμα τα τρία γράμματα f, f, i αποθηκεύονται ως ένας χαρακτήρας αντί για τρεις οπότε μια λέξη μέσα στο άρθρο δεν θα αντιστοιχεί ποτέ με την ίδια λέξη που γράφει ο χρήστης στην ερώτηση ενώ είναι ακριβώς η ίδια, κάτι που εμφανίστηκε πολύ μέσα στα άρθρα και μετρήθηκαν 721 περιπτώσεις. Ο τρόπος με τον οποίο έγινε η διόρθωση είναι μια τυποποιημένη μετατροπή της Unicode, η οποία έχει την δυνατότητα να ξαναγράψει αυτούς τους ειδικούς χαρακτήρες ως κανονικά γράμματα και όχι σαν συνδυασμό γραμμμάτων και η μορφή που επιλέχτηκε για την μετατροπή αυτή είναι NFKC [14] και όχι NFKD γιατί η δεύτερη χωρίζει τους ελληνικούς τόνους από τα γράμματα χαλώντας έτσι το ελληνικό κομμάτι της εφαρμογής. Το δεύτερο πρόβλημα είναι ότι οι λέξεις που βρίσκονται στο τέλος της γραμμής κόβονται με παύλα και με αυτόν τον τρόπο δεν υπήρχε η δυνατότητα να καταλάβει η αναζήτηση σε ποια λέξη αναφέρεται, και μετρήθηκαν 941 τέτοιες περιπτώσεις οι οποίες διορθώθηκαν με μια κανονική έκφραση η οποία εντοπίζει την παύλα που βρίσκεται ανάμεσα σε δύο γράμματα και έχει την αλλαγή γραμμής στην μέση και τις ενώνει ξανά δημιουργώντας την λέξη που ήταν πριν. Και οι δύο διορθώσεις γίνονται πριν το κείμενο μπει στην βάση ώστε τα δύο αυτά προβλήματα να μην περνούν στα αποθηκευμένα κείμενα.

Το καθάρισμα αυτό δεν λύνει όλα τα προβλήματα που δημιουργούνται στο σύστημα, καθώς μένουν τα κενά που μπαίνουν μέσα στις λέξεις όπως «distributed», τα οποία δεν μπορούν να διορθωθούν με τον ίδιο τρόπο, γιατί είναι δύσκολο να βρεθεί ποιο είναι πραγματικό κενό και ποιο δεν ήταν. Αυτά αποτελούν μέρος του 2,5% που αναφέρθηκε προηγουμένως, ποσοστό που υπήρχε σε όλο το υλικό όταν έγιναν οι μετρήσεις δηλαδή τα αποτελέσματα του κεφαλαίου 6 προέκυψαν και με αυτό το πρόβλημα και όχι σε τέλειο κείμενο.

Στην συνέχεια το κείμενο κόβεται σε κομμάτια και για αυτό χρησιμοποιήθηκε η βιβλιοθήκη langchain-text-splitters η οποία ψάχνει πρώτα να κόψει σε κενή γραμμή, μετά σε αλλαγή γραμμής, μετά σε τελεία και μόνο στο τέλος σε κενό, έτσι ώστε να σέβεται τις παραγράφους και τις προτάσεις. Επίσης το κάθε κομμάτι επικαλύπτεται με το προηγούμενο κατά 300 χαρακτήρες ώστε στην περίπτωση που η ιδέα βρίσκεται ακριβώς στο σημείο στο οποίο κόβεται να παίρνει και αυτό το κομμάτι αντί να χάνεται. Το μέγεθος των 1500 χαρακτήρων δεν επιλέχθηκε τυχαία αλλά μετά από πολλά πειράματα τα οποία έγιναν με διάφορο αριθμό χαρακτήρων και θα παρουσιαστούν πιο αναλυτικά στο κεφάλαιο 6.

Για την αναζήτηση με λέξεις κλειδιά χρησιμοποιήθηκε μια βιβλιοθήκη η οποία είναι για ακριβώς αυτό το πράγμα και ονομάζεται rank-bm25 η οποία με την σειρά της υλοποιεί τον αλγόριθμο BM25 [9] που περιγράφηκε στο δεύτερο κεφάλαιο και αποτελεί το δεύτερο σκέλος της υβριδικής αναζήτησης γιατί το πρώτο σκέλος το οποίο στηρίζεται στο νόημα χάνει ακριβείς όρους όπως ονόματα, συντομογραφίες, παραμέτρους και κωδικούς. Το BM25 έρχεται να καλύψει αυτήν την αδυναμία επειδή δεν κοιτάει να καταλάβει το νόημα αλλά κοιτάει την ίδια την λέξη σαν λέξη και επιλέχθηκε γιατί η συγκεκριμένη βιβλιοθήκη είναι μικρή και δεν χρειάζεται να τρέχει ξεχωριστά σε διαφορετικό πρόγραμμα καθώς το ευρετήριο χτίζεται και μένει στην μνήμη.

Και έπειτα έρχεται το κομμάτι της ένωσης των δύο λιστών με τον αλγόριθμο RRF [10], για το οποίο δεν χρειάστηκε κάποια βιβλιοθήκη και γράφτηκε στην εφαρμογή καθώς είναι λίγες γραμμές κώδικα που συνδυάζουν τις θέσεις των αποτελεσμάτων.

Χρειάζεται όμως πολλή προσοχή το συγκεκριμένο κομμάτι γιατί ο αλγόριθμος δεν χρησιμοποιεί τις βαθμολογίες των δύο αναζητήσεων, αφού όπως αναφέρθηκε στο δεύτερο κεφάλαιο βαθμολογούν διαφορετικά πράγματα και δεν μπορούν να συγκριθούν, αλλά μόνο την θέση της κατάταξης τους και από τις θέσεις στις οποίες βρίσκονται υπολογίζεται μια νέα βαθμολογία δική του με την οποία γίνεται και η τελική κατάταξη. Επειδή ο τύπος που χρησιμοποιείται για την τελική κατάταξη βασίζεται μόνο στην θέση που βρίσκεται το κάθε κομμάτι, δηλαδή η τρίτη θέση του ενός μετράει ακριβώς όσο και η τρίτη θέση του άλλου, δύο κομμάτια κειμένου μπορούν να καταλήξουν ίσα στην τελική βαθμολογία όταν το ένα κρατάει στην μια λίστα την θέση που κρατάει το άλλο στην άλλη. Αυτό είχε ως αποτέλεσμα η σειρά τους να αλλάζει από την μια εκτέλεση στην άλλη. Επειδή από την λίστα αυτή έμεναν μόνο τα καλύτερα 15 κομμάτια αυτό σήμαινε ότι κάθε φορά περνούσαν διαφορετικά κομμάτια και η ίδια ερώτηση μπορεί να είχε διαφορετική απάντηση, για αυτό όταν δύο κομμάτια έχουν ίδια τελική βαθμολογία η σειρά με την οποία μπαίνουν καθορίζεται από το αναγνωριστικό τους ώστε να είναι πάντα η ίδια.

3.7 Αξιολόγηση και έλεγχος

Ένα σύστημα σαν αυτό που δημιουργήθηκε μπορεί να τρέχει χωρίς κανένα μήνυμα λάθους και να δίνει λάθος απαντήσεις για αυτό πάρθηκαν κάποια μέτρα ώστε να λυθεί αυτό το πρόβλημα. Το `pytest` είναι το εργαλείο που ελέγχει αν ο κώδικας κάνει αυτό που πρέπει να κάνει, το `ruff` βλέπει τον τρόπο που είναι γραμμένος ο κώδικας και το `RAGAS` είναι ένα πλαίσιο αξιολόγησης που βαθμολογεί τις απαντήσεις που βγαίνουν στο τέλος.

Για τους ελέγχους του κώδικα χρησιμοποιήθηκε το `pytest` με το οποίο γράφτηκαν 91 έλεγχοι και έχουν μοιραστεί σε 10 αρχεία, οι οποίοι χωρίζονται σε δύο ομάδες. Η πρώτη αφορά ό,τι έρχεται από έξω, δηλαδή τα σημεία του API, τα σχήματα των δεδομένων, το όριο αιτημάτων ανά χρήστη και την ασφάλεια την οποία διαθέτουν και η δεύτερη αφορά τον πυρήνα, δηλαδή την ανάκτηση, την διορθωτική ανάκτηση, τις κλήσεις προς το Gemini και τις παραπομπές. Είναι κάτι το οποίο χρειάζεται πολύ στο σύστημα γιατί έχει πολλά μέρη που εξαρτώνται μεταξύ τους οπότε μια αλλαγή σε κάποιο σημείο μπορεί να χαλάσει κάποιο άλλο χωρίς να γίνει αντιληπτό και το πρόβλημα να φανεί αργότερα και να μην υπάρχει ιδέα από πού έχει προέλθει.

Το `ruff` δεν κοιτάει αν δουλεύει ο κώδικας αλλά πώς είναι γραμμένος και ελέγχει αυτώματα αν ακολουθεί κανόνες που πιάνουν συνηθισμένα λάθη. Από αυτούς τους κανόνες δεν κρατήθηκαν όλοι γιατί οι προεπιλεγμένοι έβγαλαν 136 ευρήματα από τα οποία τα 55 ήταν λάθος για την συγκεκριμένη εφαρμογή. Για παράδειγμα το εργαλείο θεωρούσε σαν σφάλμα τον τρόπο που το FastAPI δηλώνει τις εξαρτήσεις του, ενώ αυτός ο τρόπος είναι ο κανονικός που το ίδιο το FastAPI χρησιμοποιεί. Ένα δεύτερο είναι ότι πρότεινε μια αλλαγή στα φίλτρα βάσης η οποία αν υλοποιούνταν θα χαλούσε το ερώτημα, γιατί εκεί ο κώδικας δεν κάνει έλεγχο αλλά παράγει SQL. Το τελευταίο και πιο μεγάλο είναι ότι οι κανόνες που προστέθηκαν έβγαλαν 1092 ευρήματα επειδή τα σχόλια του συστήματος είναι στα ελληνικά και ήθελε να τα αλλάξει σε λατινικά, κάτι για το οποίο δεν υπάρχει κάποιος λόγος. Κάθε κανόνας που απενεργοποιήθηκε είναι γραμμένος στις ρυθμίσεις μαζί με τον λόγο για τον οποίο κρίθηκε ότι δεν ταιριάζει εδώ.

Ένας κανόνας ο οποίος κρατήθηκε και αξίζει να αναφερθεί είναι αυτός που βρίσκει κλήσεις που μπλοκάρουν, δηλαδή κλήσεις οι οποίες σταματάνε ολόκληρο τον server μέχρι να τελειώσει η διεργασία τους όταν αυτές οι ίδιες βρίσκονται μέσα σε ασύγχρονες συναρτήσεις. Όπως περιγράφηκε στην ενότητα 3.3, ο server έχει την δυνατότητα να ασχοληθεί

με πολλούς χρήστες μαζί επειδή όσο περιμένει ένα αίτημα να υλοποιηθεί μπορεί να ασχοληθεί με κάτι άλλο. Μια τέτοια κλήση το ακυρώνει αυτό γιατί όσο τρέχει δεν εξυπηρετείται κανένας, με το χειρότερο να είναι ότι δεν εμφανίζεται πουθενά ότι είναι σφάλμα γιατί ο κώδικας δίνει σωστό αποτέλεσμα, οι έλεγχοι περνάνε και το πρόβλημα φαίνεται μόνο όταν μπουν πολλοί χρήστες όπου αρχίζει και γίνεται ένα πολύ αργό σύστημα. Είναι δηλαδή ένα λάθος το οποίο δεν πιάνεται ούτε με δοκιμές ούτε με πειράματα παρά μόνο διαβάζοντας τον κώδικα.

Για την ποιότητα των απαντήσεων χρησιμοποιήθηκε το RAGAS [15], ένα εργαλείο ανεξάρτητο φτιαγμένο ειδικά για την αντικειμενική αξιολόγηση συστημάτων RAG, το οποίο μετράει τέσσερα πράγματα, αν η απάντηση στηρίζεται μόνο στα κείμενα που δόθηκαν και δεν επηρεάστηκε ώστε να απαντήσει με τις δικές του γνώσεις, αν απαντάει όντως στην ερώτηση και δεν λέει λάθος πράγματα, αν τα σχετικά κείμενα που ανακτήθηκαν κατατάχθηκαν ψηλά και αν τα κείμενα τα οποία βρέθηκαν για την τελική απάντηση καλύπτουν την σωστή απάντηση. Χρησιμοποιήθηκε παρόλο που υπάρχει και στην εφαρμογή ένα σύστημα αξιολόγησης, γιατί ένα σύστημα το οποίο αξιολογεί τον εαυτό του με δικά του κριτήρια δεν πείθει και δείχνει μεροληψία ενώ το RAGAS δίνει μια δεύτερη πιο αντικειμενική γνώμη από εργαλείο που δεν έχει γραφτεί για το συγκεκριμένο σύστημα.

Το RAGAS δεν τρέχει μέσα στο container του backend αλλά σε ένα ξεχωριστό περιβάλλον γιατί χρειάζεται μια νεότερη έκδοση από αυτήν την οποία έχει η εφαρμογή και αν έμπαιναν μαζί θα άλλαζαν τις εκδόσεις που δουλεύουν οπότε προτιμήθηκε να μπει σε ξεχωριστό και να διαβάζει τα αποτελέσματα που παράγει το σύστημα. Τα νούμερα από όλες αυτές τις μετρήσεις παρουσιάζονται αναλυτικά στο κεφάλαιο 6.

Κεφάλαιο 4

Σχεδίαση και υλοποίηση της εφαρμογής Z-AI Platform

4.1 Ανάλυση απαιτήσεων χρηστών (user stories)

Οι απαιτήσεις τις οποίες είχε το σύστημα καταγράφονται σε αυτό το κεφάλαιο με την μορφή user stories, δηλαδή σύντομες προτάσεις που λένε ποιος θέλει κάτι, τι είναι αυτό που θέλει και για ποιο λόγο το θέλει. Το τρίτο κομμάτι είναι το πιο σημαντικό, γιατί αν μια λειτουργία δεν μπορεί να απαντήσει στο για ποιο λόγο τότε δεν λύνει τίποτα και δεν έχει λόγο ύπαρξης. Ο κατάλογος του πίνακα 4.1 δεν προέκυψε ολόκληρος από την αρχή επειδή ένα μέρος του ήταν προφανές πριν γραφτεί ο κώδικας οπότε είχε σχεδιαστεί αλλά κάποιες απαιτήσεις φάνηκαν καλύτερα κατά την διάρκεια υλοποίησης του συστήματος.

Το σύστημα διαθέτει μόνο τον εγγεγραμμένο χρήστη, δεν υπάρχει ρόλος διαχειριστή ούτε κάποιος που τον επιβλέπει, γιατί ο κάθε χρήστης βλέπει και διαχειρίζεται μόνος του τα δικά του έγγραφα και τις δικές του συνομιλίες.

Πίνακας 4.1: Οι απαιτήσεις χρήστη.

Κωδ.	Ως χρήστης θέλω	ώστε
Λογαριασμός		
US-01	να φτιάχνω λογαριασμό	τα έγγραφά μου να μην μπορεί να τα δει κανείς άλλος
US-02	να συνδέομαι μία φορά και να μένω συνδεδεμένος	να μη δίνω κωδικό κάθε φορά που κάνω κάτι
Διαχείριση εγγράφων		
US-03	να ανεβάζω αρχεία PDF	να ρωτάω πάνω στα δικά μου κείμενα και όχι σε γενική γνώση που έχει εκπαιδευτεί το μοντέλο
US-04	να βλέπω πότε ένα αρχείο είναι έτοιμο	να μη ρωτάω ενώ γίνεται ακόμα η επεξεργασία του
US-05	να μην μπαίνει δύο φορές το ίδιο αρχείο	να μη μου γυρίζει η ίδια σελίδα δύο φορές
US-06	να διαγράφω ένα έγγραφο	να σταματήσει να εμφανίζεται στις απαντήσεις
Ερωτήσεις και απαντήσεις		
US-07	να ρωτάω στα ελληνικά ή στα αγγλικά	να μη χρειάζεται να μεταφράζω εγώ την ερώτησή μου
US-08	να βλέπω αρχείο και σελίδα για κάθε απάντηση	να ελέγχω μόνος μου αν ισχύει

συνεχίζεται στην επόμενη σελίδα

Πίνακας 4.1 — συνέχεια από την προηγούμενη σελίδα

Κωδ.	Ως χρήστης θέλω	ώστε
US-09	να περιορίζω την ερώτηση σε συγκεκριμένα αρχεία	όταν ξέρω ήδη πού είναι η απάντηση
US-10	να μου λέει ότι δεν βρήκε	να μη μου δώσει κάτι που ακούγεται σωστό χωρίς να είναι χρησιμοποιώντας δικές του γνώσεις
US-11	να ρωτάω κάτι επόμενο χωρίς να ξαναπώ το θέμα	να συνεχίζω τη σκέψη μου βάσει του ιστορικού
US-12	να διαλέγω ύφος απάντησης	να ταιριάζει σε αυτό που κάνω
US-13	να βλέπω την απάντηση να γράφεται	να ξέρω ότι δουλεύει και δεν κόλλησε
Ιστορικό συζητήσεων		
US-14	να κρατάω ξεχωριστές συζητήσεις	να μη μπερδεύονται τα θέματα μεταξύ τους
US-15	να ξαναβρίσκω μια παλιά συζήτηση	να συνεχίζω από εκεί που έμεινα
US-16	να διαγράφω μια συζήτηση	να μη μαζεύονται άχρηστα
Ανατροφοδότηση		
US-17	να δηλώνω αν μια απάντηση ήταν καλή	να καταγράφεται πού αστοχεί το σύστημα
US-18	να βλέπω πόσο χρόνο πήρε κάθε στάδιο	να καταλαβαίνω πού πάει ο χρόνος και όχι μόνο πόσος

Τα περισσότερα από τα παραπάνω είναι αναμενόμενα, δηλαδή όποιος ανεβάζει αρχεία θέλει να μπορεί να τα διαγράφει και όποιος ρωτάει θέλει να έχει και το ιστορικό του για να βλέπει παλιές συνομιλίες. Δύο όμως από αυτά φάνηκαν μόνο κατά την χρήση της εφαρμογής. Το US-05 μπήκε όταν ανέβηκε δεύτερη φορά το ίδιο αρχείο και η ίδια σελίδα εμφανίστηκε δύο φορές στην παραπομπή, κάτι που έπρεπε να διορθωθεί γιατί το σύστημα ανακτούσε δύο φορές το ίδιο υλικό. Το US-10 μπήκε όταν το σύστημα απάντησε σε ερώτηση με δικές του γνώσεις ενώ δεν υπήρχε τίποτα στα έγγραφα. Και τα δύο καθόρισαν μεγάλο μέρος από όσα ακολουθούν.

Πέρα από το τι κάνει το σύστημα, καταγράφηκε και το πώς πρέπει να το κάνει (πίνακας 4.2).

Πίνακας 4.2: Μη λειτουργικές απαιτήσεις.

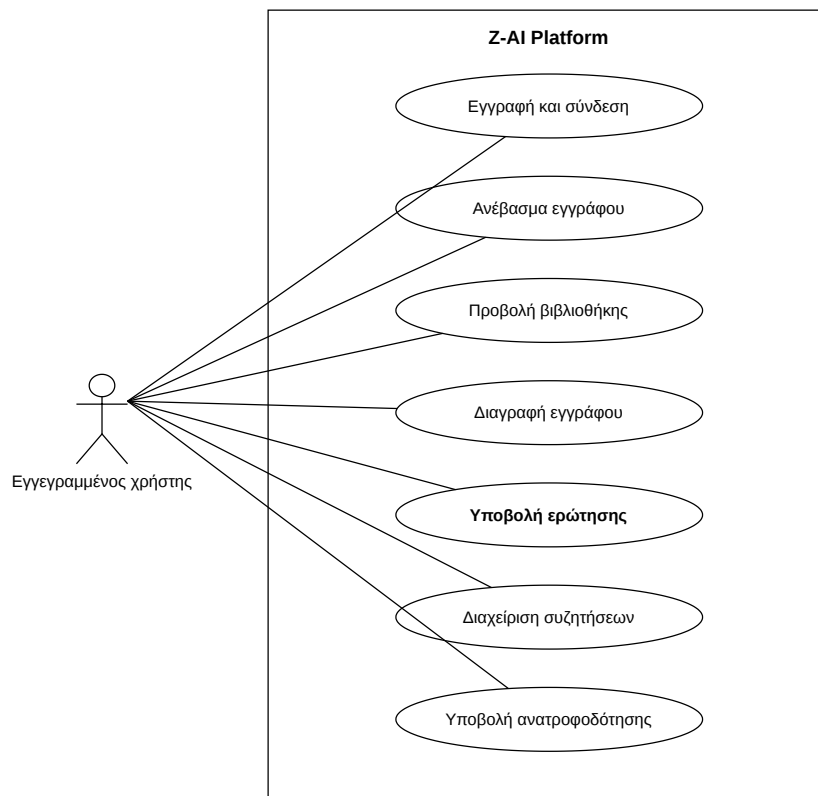
Κωδ.	Απαίτηση	Καθώς
ΜΛ-01	η ενσωμάτωση και η κατάταξη να γίνονται τοπικά	στο εξωτερικό μοντέλο να φτάνουν μόνο οι σελίδες που χρειάζονται για την απάντηση
ΜΛ-02	το σύστημα να τρέχει σε ένα μηχάνημα	να μη χρειάζεται εξωτερική υπηρεσία για να δοκιμαστεί
ΜΛ-03	το κόστος λειτουργίας να μένει χαμηλό	η ανάκτηση να μη χρεώνεται ανά ερώτηση
ΜΛ-04	οι εκδόσεις των βιβλιοθηκών να είναι καθορισμένες	οι μετρήσεις να μπορούν να επαναληφθούν και να δώσουν τα ίδια αποτελέσματα

Οι απαιτήσεις αυτές καθόρισαν τρεις επιλογές που περιγράφονται στην συνέχεια, δηλαδή το ΜΛ-01 μαζί με το ΜΛ-03 είναι ο λόγος που όλες οι διεργασίες τρέχουν τοπικά αντί να χρησιμοποιούν κάποια εξωτερική υπηρεσία, το ΜΛ-02 είναι ο λόγος που η αποθήκη διανυσμάτων είναι ενσωματωμένη στην εφαρμογή και όχι σε ξεχωριστή υπηρεσία και το

ΜΛ-04 είναι ο λόγος που οι εκδόσεις των βιβλιοθηκών έμειναν σταθερές σε όλη την διάρκεια των μετρήσεων ώστε να στέκουν οι αριθμοί.

4.2 Λειτουργικές απαιτήσεις (use cases/flow chart)

Στην προηγούμενη ενότητα αναφέρθηκε τι ζητάει ο χρήστης από το σύστημα, ενώ σε αυτήν την ενότητα περιγράφεται το τι κάνει το σύστημα για να καλύψει όλες αυτές τις ανάγκες του ώστε να κάνει πιο ομαλή την εμπειρία του βήμα προς βήμα. Κάθε λειτουργία της εφαρμογής είναι διαθέσιμη στους χρήστες οι οποίοι έχουν συνδεθεί, και το τι βλέπει ο καθένας καθορίζεται από τα έγγραφα που έχει ανεβάσει ο ίδιος και όχι από το δικαίωμα κάποιου ρόλου, αφού ρόλοι δεν υπάρχουν. Οι περιπτώσεις χρήσης που προέκυψαν είναι



Εικόνα 4.1: Διάγραμμα περιπτώσεων χρήσης.

επτά και φαίνονται συγκεντρωτικά στην εικόνα 4.1, ενώ στον πίνακα 4.3 παρουσιάζονται μαζί με τις ιστορίες χρήστη που καλύπτει η καθεμία και αυτές οι περιπτώσεις χρήσης δεν συνδέονται μεταξύ τους με σχέσεις συμπερίληψης ή επέκτασης. Επίσης στάδια όπως η αναζήτηση των σχετικών αποσπασμάτων δεν εμφανίζονται ως ξεχωριστές περιπτώσεις χρήσης καθώς δεν είναι λειτουργίες που ζητάει ο χρήστης να γίνουν μέσα στο σύστημα αλλά η συμπεριφορά του συστήματος για την οποία θα αναφερθούν περισσότερα πράγματα μετέπειτα στην ενότητα 4.4.

Από αυτές αναλύονται πλήρως μόνο οι δύο και αυτές είναι το ανέβασμα του εγγράφου και η υποβολή της ερώτησης και έχουν διαλεχτεί αυτές καθώς είναι οι μόνες με εναλλακτικές ροές που προκύπτουν λόγω της φύσης του συγκεκριμένου συστήματος και όχι από κάποιους τυπικούς ελέγχους που βλέπουν την εγκυρότητα και τα δικαιώματα (πίνακες 4.4 και 4.5).

Πίνακας 4.3: Συνοπτική παρουσίαση των περιπτώσεων χρήσης.

Κωδ.	Όνομα	Περιγραφή	Καλύπτει
UC-01	Εγγραφή και σύνδεση	Ο χρήστης δημιουργεί λογαριασμό και συνδέεται στην εφαρμογή.	US-01, US-02
UC-02	Ανέβασμα εγγράφου	Ο χρήστης ανεβάζει αρχείο PDF, το οποίο επεξεργάζεται στο παρασκήνιο.	US-03, US-05
UC-03	Προβολή βιβλιοθήκης	Ο χρήστης βλέπει τα διαθέσιμα έγγραφα και την κατάσταση επεξεργασίας τους.	US-04
UC-04	Διαγραφή εγγράφου	Ο χρήστης διαγράφει έγγραφο που του ανήκει.	US-06
UC-05	Υποβολή ερώτησης	Ο χρήστης θέτει ερώτηση και λαμβάνει απάντηση με παραπομπές.	US-07 έως US-13
UC-06	Διαχείριση συζητήσεων	Ο χρήστης δημιουργεί, ανοίγει και διαγράφει συζητήσεις.	US-14 έως US-16
UC-07	Υποβολή ανατροφοδότησης	Ο χρήστης αξιολογεί μια απάντηση θετικά ή αρνητικά.	US-17

Οι δύο περιπτώσεις που αναλύθηκαν έδειξαν ότι το σύστημα ασχολείται το ίδιο με το τι κάνει όταν όλα πάνε καλά και με το τι κάνει όταν κάτι πάει στραβά. Έδειξαν επίσης δύο πράγματα που έπρεπε να λυθούν στην δομή. Το πρώτο είναι ότι ο χρήστης παίρνει απάντηση για το ανέβασμα πριν το έγγραφο γίνει έτοιμο, κάτι που είναι βοηθητικό αλλά υποχρεώνει το σύστημα να κρατάει και να δείχνει την κατάσταση κάθε εγγράφου. Το δεύτερο είναι ότι για να απαντήσει σε μια ερώτηση δεν του αρκεί να ξέρει ποια έγγραφα υπάρχουν, χρειάζεται και το κείμενό τους σε μορφή στην οποία να μπορεί να το ψάξει και η δομή που προέκυψε περιγράφεται στην ενότητα 4.3.

4.3 Αρχιτεκτονική του συστήματος

Σε αυτήν την ενότητα περιγράφεται η δομή που προέκυψε από αυτήν την ανάγκη, δηλαδή από ποια μέρη αποτελείται το σύστημα, ποιο μέρος επικοινωνεί με ποιο και πού αποθηκεύεται τι (εικόνα 4.2). Το πώς γίνεται η ίδια η αναζήτηση δεν περιγράφεται εδώ αλλά στην ενότητα 4.4. Το σύστημα δεν αποθηκεύει τα δεδομένα του σε ένα μέρος αλλά σε δύο βάσεις οι οποίες κάνουν διαφορετική δουλειά, δηλαδή η σχεσιακή βάση απαντάει σε ερωτήσεις τύπου ποιος χρήστης έχει ποιο έγγραφο και τι ειπώθηκε σε ποια συζήτηση ενώ η βάση διανυσμάτων απαντάει σε τελείως άλλη ερώτηση δηλαδή ποια κομμάτια μοιάζουν με την ερώτηση του χρήστη. Η μία συμπληρώνει την άλλη διότι καμία από τις δύο δεν κάνει καλά την δουλειά της άλλης και για αυτόν τον λόγο χρησιμοποιήθηκαν και οι δύο.

Στην PostgreSQL είναι οι λογαριασμοί των χρηστών, η καταχώρηση του κάθε εγγράφου μαζί με το όνομα το οποίο έχει, την κατάστασή του, τον χρήστη που το έχει και το αποτύπωμα του περιεχομένου του, οι συζητήσεις που έχει κάνει, τα μηνύματα που έχει στείλει, οι αξιολογήσεις και οι μετρητές ορίων που έχουν γίνει. Το ίδιο το αρχείο PDF έχει μια ιδιαιτερότητα, γιατί η βάση κρατάει μόνο την διαδρομή του ενώ το αρχείο μένει στον δίσκο. Στην ChromaDB κρατούνται τα κομμάτια στα οποία σπάει το κείμενο, το διάνυσμα του κάθε

Πίνακας 4.4: Περίπτωση χρήσης UC-02: Ανέβασμα εγγράφου.

Κωδικός	UC-02
Όνομα	Ανέβασμα εγγράφου
Δρων	Εγγεγραμμένος χρήστης
Προϋποθέσεις	Ο χρήστης είναι συνδεδεμένος στην εφαρμογή
Βασική ροή	1. Ο χρήστης επιλέγει ένα αρχείο από τον υπολογιστή του και το ανεβάζει. 2. Το σύστημα ελέγχει εάν το αρχείο είναι PDF. 3. Το σύστημα αποθηκεύει το αρχείο ελέγχοντας παράλληλα ότι δεν ξεπερνά το όριο που υπάρχει στο μέγεθος. 4. Ελέγχει αν το ίδιο ακριβώς περιεχόμενο υπάρχει ήδη ανάμεσα στα έγγραφα στα οποία έχει πρόσβαση ο χρήστης. 5. Καταχωρεί το έγγραφο με κατάσταση «σε επεξεργασία» και ενημερώνει αμέσως τον χρήστη ότι το ανέβασμα ολοκληρώθηκε. 6. Η επεξεργασία του κειμένου συνεχίζεται στο παρασκήνιο, χωρίς ο χρήστης να περιμένει δίνοντάς του την δυνατότητα να ασχοληθεί με κάτι άλλο μέχρι να γίνει το ανέβασμα. 7. Με την ολοκλήρωσή της η κατάσταση γίνεται «έτοιμο» και το έγγραφο μπορεί να χρησιμοποιηθεί σε ερωτήσεις.
Εναλλακτικές ροές	2α. Αν το αρχείο δεν είναι PDF τότε απορρίπτεται και ο χρήστης ενημερώνεται. 3α. Αν το αρχείο ξεπερνά το όριο μεγέθους που υπάρχει τότε η αποθήκευση διακόπτεται, το μερικώς αποθηκευμένο αρχείο διαγράφεται και ο χρήστης ενημερώνεται για το όριο. 4α. Αν το ίδιο περιεχόμενο υπάρχει τότε το αρχείο διαγράφεται αυτόματα και ο χρήστης ενημερώνεται με μήνυμα. 7α. Αν δεν εξήχθη κείμενο από το αρχείο τότε η κατάσταση γίνεται «κενό», ώστε ο χρήστης να γνωρίζει ότι το έγγραφο δεν πρόκειται να απαντήσει σε ερωτήσεις. 7β. Αν η επεξεργασία απέτυχε τότε η κατάσταση γίνεται «απέτυχε» και ο χρήστης μπορεί να επαναλάβει το ανέβασμα. 7γ. Αν ο διακομιστής επανεκκινήθηκε ενώ η επεξεργασία ήταν σε εξέλιξη, τότε κατά την εκκίνηση όσα έγγραφα είχαν μείνει σε επεξεργασία σημειώνονται ως αποτυχημένα.
Αποτέλεσμα	Το έγγραφο είναι διαθέσιμο για ερωτήσεις, ή ο χρήστης έχει ενημερωθεί για τον λόγο που δεν έγινε δεκτό.

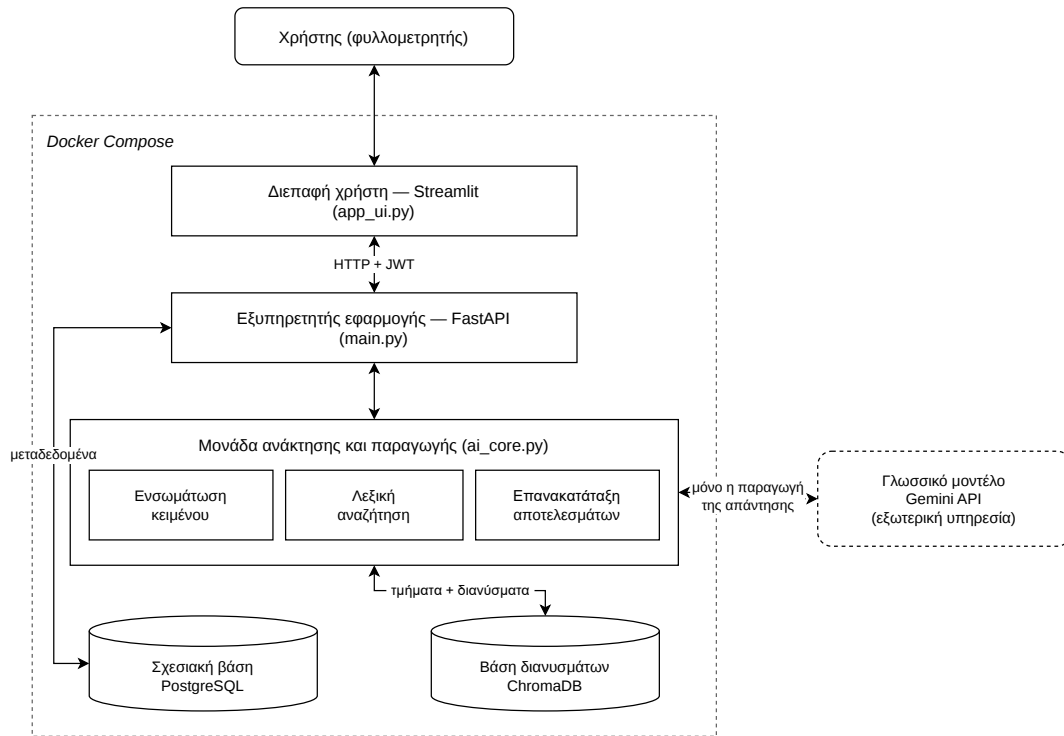
κομματιού που υπάρχει και μερικά μεταδεδομένα, δηλαδή από ποιο αρχείο και ποια σελίδα προέρχεται το κάθε κομμάτι και σε ποιον ανήκει από τα οποία το αρχείο και η σελίδα χρησιμοποιούνται για την παραπομπή στην απάντηση.

Τα δύο συστήματα συνδέονται μεταξύ τους με το αναγνωριστικό του εγγράφου δηλαδή το ίδιο αναγνωριστικό που έχει το έγγραφο που υπάρχει στην PostgreSQL γράφεται και μέσα σε κάθε κομμάτι του στην ChromaDB στο οποίο γράφεται και ο ιδιοκτήτης του εγγράφου κάτι το οποίο το γνωρίζει ήδη η PostgreSQL.

Η επανάληψη αυτή μπορεί να φαίνεται περιττή αλλά δεν είναι, γιατί οι δύο βάσεις δεν είναι δύο πίνακες της ίδιας βάσης ώστε να ξέρουν πράγματα ο ένας για τον άλλον. Είναι δύο ξεχωριστά προγράμματα και δεν υπάρχει ερώτημα που να τα ενώνει. Την αναζήτηση την κάνει η ChromaDB η οποία θα πρέπει να μπορεί μόνη της να ξεχωρίσει ποιο κομμάτι ανήκει σε ποιον χρήστη χωρίς να χρειάζεται να ρωτήσει την PostgreSQL και για αυτόν τον λόγο ο έλεγχος πρόσβασης γράφτηκε μέσα στα ίδια τα κομμάτια και έτσι η αναζήτηση περιορίζεται από την αρχή στα έγγραφα του χρήστη. Το πώς εντάσσεται αυτό στην σειρά την οποία γίνονται τα βήματα της αναζήτησης περιγράφεται αναλυτικά στην ενότητα 4.4. Η εικόνα 4.3 δείχνει ποιες οντότητες υπάρχουν και πώς συνδέονται μεταξύ τους αλλά δεν

Πίνακας 4.5: Περίπτωση χρήσης UC-05: Υποβολή ερώτησης.

Κωδικός	UC-05
Όνομα	Υποβολή ερώτησης
Δρων	Εγγεγραμμένος χρήστης
Προϋποθέσεις	Ο χρήστης είναι συνδεδεμένος, υπάρχει ανοιχτή συζήτηση και τουλάχιστον ένα έγγραφο σε κατάσταση «έτοιμο»
Βασική ροή	1. Ο χρήστης γράφει την ερώτηση που θέλει να κάνει μέσα σε μια συζήτηση. 2. Προαιρετικά περιορίζει την αναζήτηση σε συγκεκριμένα άρθρα που επιλέγει ο ίδιος και επιλέγει το ύφος που θέλει να έχει η απάντηση. 3. Το σύστημα παράλληλα καταγράφει την ερώτηση στο ιστορικό της συζήτησης. 4. Αν η ερώτηση είναι στα ελληνικά μεταφράζεται στα αγγλικά αποκλειστικά για τις ανάγκες της αναζήτησης. 5. Το σύστημα βλέπει τα πιο σχετικά αποσπάσματα μόνο μέσα στα έγγραφα στα οποία έχει πρόσβαση ο χρήστης. 6. Ελέγχει αν το καλύτερο απόσπασμα είναι αρκετά σχετικό ώστε να στηρίξει απάντηση. 7. Στέλνει στο γλωσσικό μοντέλο την ερώτηση μαζί με τις σελίδες που επιλέχθηκαν. 8. Η απάντηση εμφανίζεται σταδιακά μαζί με τις παραπομπές που πήρε από τα έγγραφα και τις σελίδες από τις οποίες προέκυψε. 9. Η ολοκληρωμένη απάντηση αποθηκεύεται στο ιστορικό της συζήτησης.
Εναλλακτικές ροές	1α. Ο χρήστης έχει κάνει πολλές ερωτήσεις σε σύντομο χρονικό διάστημα. Το σύστημα τον ενημερώνει να περιμένει και δεν εκτελεί την ερώτηση. 6α. Αν το καλύτερο απόσπασμα δεν θεωρείται αρκετά σχετικό τότε το σύστημα αναδιατυπώνει την ερώτηση και επαναλαμβάνει την αναζήτηση μια φορά. 6β. Αν ούτε η δεύτερη προσπάθεια εντοπίζει αρκετά σχετικό υλικό τότε το σύστημα δηλώνει ότι δεν βρέθηκε απάντηση στα έγγραφα και δεν παράγει κείμενο. 6γ. Αν η ερώτηση αναφέρεται σε κάτι χωρίς να το ονομάζει τότε το σύστημα ζητά από τον χρήστη να διευκρινίσει σε ποιο έγγραφο ή σύστημα αναφέρεται. 8α. Αν ο χρήστης κλείνει τη σελίδα ενώ γράφεται η απάντηση τότε ό,τι έχει παραχθεί μέχρι εκείνη τη στιγμή αποθηκεύεται στο ιστορικό.
Αποτέλεσμα	Η ερώτηση και η απάντηση έχουν καταγραφεί στη συζήτηση, ή ο χρήστης έχει ενημερωθεί ότι δεν υπάρχει σχετικό υλικό στα έγγραφα στα οποία έχει πρόσβαση.



Εικόνα 4.2: Αρχιτεκτονική του συστήματος.

δείχνει τι ακριβώς κρατάει η καθεμία. Στον πίνακα 4.6 παρουσιάζεται το σχεσιακό σχήμα δηλαδή όλα τα πεδία του κάθε πίνακα με τον τύπο τους και τα κλειδιά τα οποία διαθέτουν.

Πίνακας 4.6: Το σχεσιακό σχήμα της βάσης.

Πεδίο	Τύπος	Κλειδί	Περιγραφή
users — λογαριασμοί χρηστών			
id	INTEGER	PK	Μοναδική και υποχρεωτική Μοναδικό και υποχρεωτικό Σύνοψη του κωδικού, όχι ο κωδικός
email	VARCHAR	UQ	
username	VARCHAR	UQ	
hashed_password	VARCHAR		
documents — καταχώρηση εγγράφων			
id	INTEGER	PK	Όπως το ανέβασε ο χρήστης Η διαδρομή στον δίσκο Σύνοψη SHA-256, μπορεί να είναι κενό processing, ready, empty, failed
file_name	VARCHAR		
file_path	VARCHAR		
file_hash	VARCHAR		
status	VARCHAR		Ορατό σε όλους ή μόνο στον κάτοχο Ο κάτοχος, προς users.id
is_public	BOOLEAN		
user_id	INTEGER	FK	
conversations — συζητήσεις			
id	INTEGER	PK	Ο τίτλος της συζήτησης Την ορίζει η βάση, μπορεί να είναι κενή Ο κάτοχος, προς users.id
title	VARCHAR		
created_at	TIMESTAMPTZ		
user_id	INTEGER	FK	
messages — μηνύματα συζήτησης			
id	INTEGER	PK	user ή assistant
role	VARCHAR		

συνεχίζεται στην επόμενη σελίδα

Πίνακας 4.6 — συνέχεια από την προηγούμενη σελίδα

Πεδίο	Τύπος	Κλειδί	Περιγραφή
content	VARCHAR		Το κείμενο του μηνύματος
sources	VARCHAR		Οι παραπομπές, μπορεί να είναι κενές
conversation_id	INTEGER	FK	Προς conversations.id
feedbacks — αξιολογήσεις απαντήσεων			
id	INTEGER	PK	
is_positive	BOOLEAN		Θετική ή αρνητική
comment	VARCHAR		Προαιρετικό σχόλιο του χρήστη
message_id	INTEGER	FK	Η απάντηση, προς messages.id
user_id	INTEGER	FK	Ο χρήστης, προς users.id
rate_limits — μετρητές ορίων αιτημάτων			
bucket	VARCHAR	PK	login ή chat
key	VARCHAR	PK	Διεύθυνση IP ή αναγνωριστικό χρήστη
window_start	INTEGER	PK	Η αρχή του χρονικού παραθύρου
hits	INTEGER		Πλήθος αιτημάτων στο παράθυρο

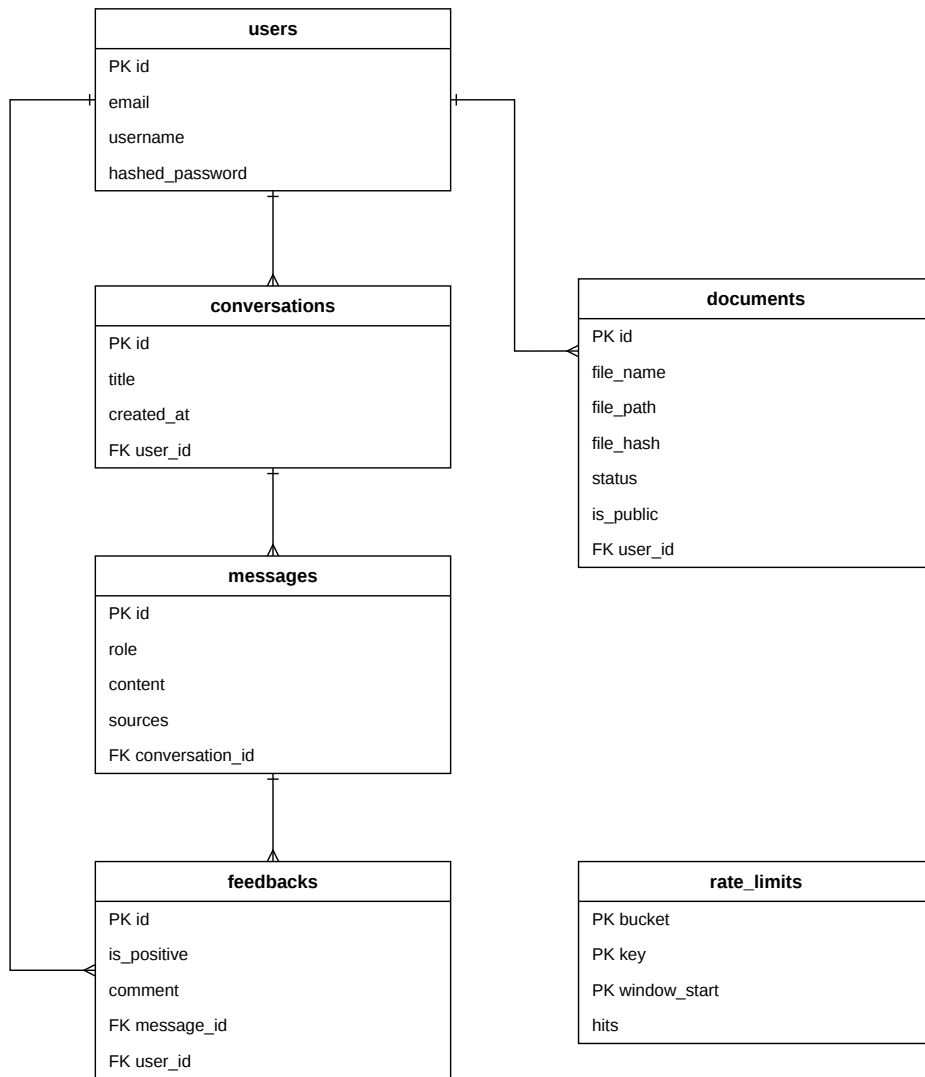
Τέσσερα από τα παραπάνω πεδία δεν είναι αυτονόητα και αξίζει να εξηγηθούν. Το πρώτο είναι ο κωδικός του χρήστη, ο οποίος δεν αποθηκεύεται όπως τον γράφει ο χρήστης αλλά ως σύνοψη και αυτό γίνεται ώστε ακόμα και αν κάποιος το διαβάσει στην βάση να μην μπορεί να καταλάβει ποιος είναι.

Το αποτύπωμα του περιεχομένου υπάρχει για να αναγνωρίζεται ένα έγγραφο από αυτό που περιέχει και όχι από το όνομά του, καθώς το ίδιο αρχείο μπορεί να ανέβει δεύτερη φορά με διαφορετικό όνομα και χωρίς αυτό το σύστημα δεν θα καταλάβαινε ότι το έχει ήδη.

Η ώρα δημιουργίας μιας συζήτησης δεν γράφεται από την εφαρμογή αλλά από την βάση και ο λόγος είναι ότι η εφαρμογή μπορεί να τρέχει διαφορετικές εργασίες και τα ρολόγια τους δεν είναι σίγουρο ότι θα δείχνουν την ίδια ώρα ενώ η βάση είναι μία και δείχνει την ίδια ώρα σε όλους. Επειδή η στήλη προστέθηκε αργότερα υπήρχαν ήδη συζητήσεις οι οποίες έμειναν κενές επειδή η ώρα που φτιάχτηκαν δεν ήταν γνωστή και δεν θα ήταν σωστό να γραφόταν μια ώρα η οποία δεν ισχύει.

Τέλος ο πίνακας των ορίων δεν έχει δικό του αναγνωριστικό αλλά κύριο κλειδί από τρία πεδία μαζί δηλαδή το είδος του ορίου το ποιον αφορά αυτό το όριο και ποιο χρονικό παράθυρο μετράει. Ο συνδυασμός αυτών είναι που ξεχωρίζει κάθε γραμμή.

Η βάση διανυσμάτων δεν παρουσιάζεται με διάγραμμα οντοτήτων και συσχετίσεων επειδή δεν είναι σχεσιακή, δηλαδή δεν έχει πίνακες, δεν έχει ξένα κλειδιά και δεν υπάρχουν συσχετίσεις. Υπάρχει μόνο μια συλλογή που περιέχει κομμάτια κειμένου το ένα δίπλα στο άλλο, όπου κάθε κομμάτι στέκει μόνο του και κουβαλάει ό,τι χρειάζεται ώστε να εντοπιστεί. Στον πίνακα 4.7 φαίνεται τι αποθηκεύεται για κάθε κομμάτι. Οι δύο βάσεις δεν είναι ισότιμες, καθώς τα αρχεία στον δίσκο μαζί με την σχεσιακή βάση αποτελούν το επίσημο αρχείο στο οποίο μέσα καταγράφονται ποια έγγραφα υπάρχουν και σε ποιον ανήκουν ενώ η βάση διανυσμάτων δεν κρατάει τίποτα δικό της αφού ό,τι έχει φτιάξει είναι από εκείνα τα αρχεία και μπορεί να τα ξαναφτιάξει όποτε χρειαστεί. Αν όμως χαθεί ένα αρχείο από τον δίσκο δεν ξαναφτιάχνεται τίποτα από αυτό. Επειδή η ίδια πληροφορία γράφεται σε δύο σημεία τα δύο μπορούν να διαφωνήσουν, οπότε το σύστημα πρέπει να τα κρατάει σε συμφωνία, αλλιώς η αναζήτηση θα συνεχίσει να βρίσκει έγγραφα που έχουν διαγραφεί.



Εικόνα 4.3: Διάγραμμα οντοτήτων-συσχετίσεων της σχεσιακής βάσης.

Πίνακας 4.7: Τι αποθηκεύεται για κάθε κομμάτι κειμένου στη βάση διανυσμάτων.

Στοιχείο	Τι αποθηκεύεται
Το ίδιο το κομμάτι	
Αναγνωριστικό	Αρχείο, σελίδα, σειρά μέσα στη σελίδα και τυχαίο επίθεμα
Κείμενο	Το απόσπασμα, όπως εξήχθη από τη σελίδα
Διάνυσμα	Η αριθμητική αναπαράσταση του αποσπάσματος
Μεταδεδομένα	
file_name	Το αρχείο από το οποίο προέρχεται
page	Ο αριθμός της σελίδας
doc_id	Το αναγνωριστικό του εγγράφου στη σχεσιακή βάση
user_id	Ο κάτοχος του εγγράφου
is_public	Αν είναι ορατό σε όλους τους χρήστες
terms	Οι βασικοί αγγλικοί όροι του εγγράφου
domain	Το γνωστικό πεδίο, όταν είναι διαθέσιμο

4.4 Υλοποίηση του backend

4.4.1 Η ανατομία ενός endpoint

Κάθε λειτουργία του συστήματος εκτίθεται ως ένα σημείο εισόδου του API, δηλαδή μια συνάρτηση που συνδέεται με μια διεύθυνση και μια μέθοδο του HTTP μέσω του διακομιστή όπου ο κώδικας που φαίνεται παρακάτω δηλώνει ότι κάθε αίτημα τύπου POST προς την διεύθυνση της συνομιλίας θα εξυπηρετηθεί από αυτήν την συνάρτηση.

```

1 @app.post("/chat")
2 async def chat_with_ai(query: schemas.ChatMessage,
3                       current_user: models.User = Depends(
4                           get_current_user),
5                       db: Session = Depends(get_db)):
6     rate_limit.check(db, "chat", str(current_user.id),
7                       CHAT_MAX_REQUESTS, CHAT_WINDOW_SEC, ...)
8
9     conv = db.query(models.Conversation).filter(
10         models.Conversation.id == query.conversation_id,
11         models.Conversation.user_id == current_user.id).first()
12     if not conv:
13         raise HTTPException(status_code=404, detail="Conversation not
14                             found")
15     ...

```

Κώδικας 4.1: Σημείο εισόδου συνομιλίας: ταυτοποίηση και έλεγχος ιδιοκτησίας

Το ενδιαφέρον δεν βρίσκεται στο σώμα της συνάρτησης αλλά στην υπογραφή της διότι τα τρία ορίσματα που υπάρχουν δεν είναι τα δεδομένα που περνάει κάποιος στην συνάρτηση αλλά προϋποθέσεις που εκτελούνται πριν ξεκινήσει η συνάρτηση. Το πρώτο επιβάλλει το πώς είναι τα δεδομένα τα οποία έφτασαν, το δεύτερο ελέγχει ότι ο χρήστης έχει δικαίωμα πρόσβασης και το τρίτο ανοίγει μια σύνδεση με την βάση που θα κλείσει σε κάθε περίπτωση. Έτσι ο έλεγχος ταυτότητας δεν είναι μια απλή εντολή μέσα στην συνάρτηση που θα μπορούσε κάποιος να την ξεχάσει κατά λάθος, αλλά μέρος της ίδιας της δήλωσης. Αν λείπει αυτό από την συνάρτηση τότε το σημείο πρόσβασης είναι ελεύθερο προς όλους, ενώ άμα υπάρχει κάποιο λάθος στο αίτημα δεν φτάνει ποτέ στην πρώτη γραμμή του σώματος της συνάρτησης.

```

1 def get_current_user(token: str = Depends(oauth2_scheme),
2                     db: Session = Depends(get_db)):
3     credentials_exception = HTTPException(status_code=401, ...)
4     try:
5         payload = jwt.decode(token, security.SECRET_KEY,
6                               algorithms=[security.ALGORITHM])
7         username: str = payload.get("sub")
8         if username is None:
9             raise credentials_exception          # 1. χωρίς ονομα μέσα
10    except JWTError as e:
11        raise credentials_exception from e       # 2. ακυρο η ληγμενο
12
13    user = db.query(models.User).filter(
14        models.User.username == username).first()
15    if user is None:
16        raise credentials_exception              # 3. ο χρηστης δεν
17    υπάρχει
18    return user

```

Κώδικας 4.2: Ταυτοποίηση χρήστη από το διακριτικό πρόσβασης

Η συνάρτηση που ταυτοποιεί τον χρήστη διαβάζει το διακριτικό που συνόδευε το αίτημα το αποκρυπτογραφεί με το μυστικό κλειδί που έχει ο εξυπηρετητής, εξάγει το όνομα χρήστη και ανακτά την αντίστοιχη γραμμή από την βάση. Υπάρχουν τρία σημεία στα οποία μπορεί να αποτύχει, δηλαδή όταν το διακριτικό δεν περιέχει όνομα, όταν είναι άκυρο ή ληγμένο και όταν ο χρήστης έχει διαγραφεί. Και στις τρεις περιπτώσεις επιστρέφεται το ίδιο ακριβώς σφάλμα ώστε η απάντηση να μην αποκαλύπτει ποιος από τους ελέγχους απέτυχε για να μην δώσει παραπάνω στοιχεία σε κάποιον ο οποίος θα μπει παράνομα στο σύστημα.

4.4.2 Επεξεργασία

Όταν ο χρήστης ανεβάζει ένα PDF δεν γίνονται όλες οι διεργασίες μαζί. Το αίτημα κάνει δύο πράγματα και μετά τελειώνει, γράφει το αρχείο στον δίσκο και γράφει μια γραμμή στην βάση με κατάσταση σε επεξεργασία. Η πραγματική δουλειά, δηλαδή το κόψιμο του κειμένου σε κομμάτια και η δημιουργία διανυσμάτων, γίνεται στο παρασκήνιο. Ο λόγος είναι πρακτικός, επειδή ένα PDF με ογδόντα σελίδες θέλει αρκετή ώρα για να επεξεργαστεί στο σύστημα και αν το αίτημα περίμενε να τελειώσει για να κάνει κάτι άλλο η ιστοσελίδα θα είχε κόψει ήδη την σύνδεση του χρήστη. Έτσι ο χρήστης γνωρίζει την κατάσταση στην οποία βρίσκεται το αρχείο και πράττει ανάλογα, αλλιώς δεν θα είχε κάποιον τρόπο να ξέρει αν το αρχείο είναι έτοιμο να δεχτεί ερωτήσεις.

Πριν γραφτεί το οτιδήποτε στον δίσκο γίνονται κάποιои έλεγχοι. Δεκτά είναι μόνο αρχεία PDF, ενώ από το όνομα που στέλνει η ιστοσελίδα κρατιέται μόνο το τελευταίο κομμάτι, έτσι ώστε ένα όνομα που έχει μέσα διαδρομές να μην μπορεί να γράψει τίποτα έξω από τον φάκελο των εγγράφων. Επιπλέον το αρχείο παίρνει ένα τυχαίο πρόθεμα, οπότε δύο χρήστες που ανεβάζουν και οι δύο ένα ίδιο PDF δεν συναντιούνται ποτέ δηλαδή δεν πατάει ο ένας στον άλλον. Τέλος το γράψιμο γίνεται σε κομμάτια με συνολικό όριο τα πενήντα megabyte, και αν η εγγραφή το ξεπεράσει σταματάει και το ημιτελές αρχείο διαγράφεται γιατί χωρίς αυτό το όριο ένα τεράστιο αρχείο θα γέμιζε τον δίσκο πριν καν φτάσει για να επεξεργαστεί.

Δύο λεπτομέρειες από αυτό το κομμάτι αξίζουν να αναλυθούν και αυτές είναι ότι το αποτύπωμα του περιεχομένου υπολογίζεται πάνω στα ίδια κομμάτια τα οποία έχουν διαβαστεί άρα δεν χρειάζεται δεύτερο πέρασμα στο αρχείο. Η δεύτερη είναι ότι ολόκληρο το γράψιμο βγαίνει σε ξεχωριστό νήμα επειδή αλλιώς ένα ανέβασμα PDF πενήντα megabyte θα κρατούσε απασχολημένο τον εξυπηρετητή και θα περίμεναν όλοι οι υπόλοιποι μέχρι να τελειώσει.

```

1 @app.post("/upload")
2 async def upload_file(background_tasks: BackgroundTasks,
3                        file: UploadFile = File(...),
4                        current_user: models.User = Depends(
5                            get_current_user),
6                        db: Session = Depends(get_db)):
7     safe_name = Path(file.filename or "").name
8     if not safe_name.lower().endswith(".pdf"):
9         raise HTTPException(status_code=400, detail="...")
10
11     disk_name = f"{uuid.uuid4().hex}_{safe_name}"
12     file_path = os.path.join(UPLOAD_DIR, disk_name)
13     MAX_UPLOAD_BYTES = 50 * 1024 * 1024
14
15     def _write_stream() -> tuple[int, bool, str]:

```

```

15     h = hashlib.sha256()
16     n, over = 0, False
17     with open(file_path, "wb") as buffer:
18         while chunk := file.file.read(1024 * 1024):
19             n += len(chunk)
20             if n > MAX_UPLOAD_BYTES:
21                 over = True
22                 break
23             h.update(chunk)
24             buffer.write(chunk)
25     return n, over, h.hexdigest()
26
27     _written, too_big, digest = await asyncio.to_thread(_write_stream)

```

Κώδικας 4.3: Ανέβασμα εγγράφου: καθαρισμός ονόματος, όριο μεγέθους και αποτύπωμα σε ένα πέρασμα

Κάτι επίσης που ελέγχεται είναι αν το αρχείο υπάρχει ήδη, και ο έλεγχος δεν γίνεται με το όνομα, διότι το ίδιο ακριβώς PDF μπορεί να έχει διαφορετικό όνομα, αλλά με το αποτύπωμα του αρχείου. Το ενδιαφέρον είναι ότι ψάχνει μέσα στα έγγραφα του χρήστη, δηλαδή ακριβώς σε ό,τι βλέπει και η αναζήτηση. Ο λόγος δεν είναι τυχαίος: αν το ίδιο κείμενο υπήρχε δύο φορές θα εμφανιζόταν δύο φορές στην ίδια απάντηση παίρνοντας την θέση κάποιου άλλου εγγράφου. Για αυτό όταν βρεθεί ότι ένα αρχείο υπάρχει ήδη το αίτημα απορρίπτεται και το αρχείο που μόλις γράφτηκε διαγράφεται.

Στο κομμάτι του παρασκήνιου η επεξεργασία μπορεί να καταλήξει σε τρεις καταστάσεις αντί για δύο και αυτές είναι έτοιμο, κενό και αποτυχημένο, με περισσότερο ενδιαφέρον να έχει η δεύτερη δηλαδή το κενό. Όταν ένα PDF είναι σαρωμένη εικόνα δεν βγαίνει κανένα κείμενο οπότε δεν γράφεται και κανένα κομμάτι αλλά τίποτα δεν έχει αποτύχει και ο κώδικας τρέχει κανονικά. Αν αυτό δηλωνόταν σαν αποτυχία ο χρήστης θα έπαιρνε το μήνυμα ότι πρέπει να ξαναδοκιμάσει, και η δεύτερη προσπάθεια θα του έδινε τα ίδια ακριβώς αποτελέσματα. Η ξεχωριστή αυτή κατάσταση επιτρέπει να του ειπωθεί το σωστό δηλαδή ότι το συγκεκριμένο αρχείο θέλει οπτική αναγνώριση χαρακτήρων για να διαβαστεί.

```

1     try:
2         ok = ai_core.ingest_pdf(file_path, filename, user_id, doc_id=
doc_id)
3         new_status = "ready" if ok else "empty"
4     except Exception as e:
5         new_status = "failed"
6
7     doc = db.query(models.Document).filter(
8         models.Document.id == doc_id).first()
9     if doc:
10         doc.status = new_status
11         db.commit()
12     else:
13         ai_core.delete_file_from_db(filename, user_id=user_id, doc_id=
doc_id)

```

Κώδικας 4.4: Επεξεργασία στο παρασκήνιο: οι τρεις καταστάσεις και ο καθαρισμός ορφανών κομματιών

Η σειρά των βημάτων για την διαγραφή έχει μεγάλη σημασία δηλαδή πρώτα φεύγουν τα κομμάτια από την βάση των διανυσμάτων μετά το PDF που βρίσκεται στον δίσκο και τελευταία η γραμμή από την σχεσιακή βάση. Αν η σειρά αντιστραφεί η γραμμή χάνεται και το έγγραφο εξαφανίζεται, αλλά τα κομμάτια μένουν πίσω ενεργά, οπότε σε επόμενη ερώτηση

το σύστημα απαντάει με παραπομπές σε αρχείο που ο χρήστης έχει ήδη σβήσει. Για αυτό τον λόγο η συνάρτηση που καθαρίζει τα κομμάτια σε περίπτωση που αποτύχει ξαναπετάει σφάλμα ώστε να μην γίνει η διαγραφή στην σχεσιακή βάση και να μείνει η κατάσταση επισκευάσιμη.

```

1 @app.delete("/documents/{doc_id}")
2 def delete_document(doc_id: int, current_user=Depends(get_current_user),
3
4         db: Session = Depends(get_db)):
5     # ... ευρεση του εγγραφου με ελεγχο ιδιοκτησιας, αλλιως 404 ...
6     ai_core.delete_file_from_db(doc.file_name,          # 1. τα
7     κομματα
8                                     user_id=current_user.id, doc_id=doc.id)
9     try:
10         if doc.file_path and os.path.exists(doc.file_path):
11             os.remove(doc.file_path)          # 2. το PDF
12     except OSError as e:
13         logger.warning(f"...")
14         db.delete(doc)                        # 3. η γραμμη
15         db.commit()
16     return {"message": "Document deleted"}

```

Κώδικας 4.5: Διαγραφή εγγράφου: κομμάτια, αρχείο, εγγραφή – με αυτή τη σειρά

Μένει ακόμα μια περίπτωση η οποία δεν λύνεται με την σειρά. Αν ο χρήστης πατήσει διαγραφή ενώ η επεξεργασία τρέχει ακόμα, η διαγραφή δεν θα βρει τίποτα στην βάση διανυσμάτων και δεν θα σβήσει τίποτα, ενώ η εργασία παρασκηνίου θα γράψει κομμάτια για ένα έγγραφο που δεν υπάρχει πια. Είναι το ίδιο πρόβλημα με το προηγούμενο, απλά από άλλον δρόμο. Η λύση στο συγκεκριμένο πρόβλημα βρίσκεται στο τέλος της εργασίας παρασκηνίου. Όταν πάει να ενημερώσει την κατάσταση και δεν βρίσκει την γραμμή, καταλαβαίνει τι έχει συμβεί και καθαρίζει από μόνο του όσα έγραψε.

4.4.3 Η διαδρομή μιας ερώτησης

Όλη η διαδρομή η οποία κάνει μια ερώτηση ζει σε μια συνάρτηση δηλαδή της δίνεται η ερώτηση έτσι ακριβώς όπως την έγραψε ο χρήστης μαζί με το ποιος είναι και ένα προαιρετικό κομμάτι είναι ποια αρχεία έχει επιλέξει και επιστρέφει έτοιμο κείμενο να μπει στο μοντέλο. Ό,τι γίνεται ενδιάμεσα πραγματοποιείται μέσα σε ένα σημείο και όχι σκορπισμένα σε πολλά.

```

1 def _build_where(target_filenames: list | None = None, user_id: int |
2     None = None):
3     clauses = []
4     if user_id is not None:
5         clauses.append({"$or": [{"user_id": user_id}, {"is_public":
6         True}]})
7     if target_filenames:
8         if len(target_filenames) == 1:
9             clauses.append({"file_name": target_filenames[0]})
10        else:
11            clauses.append({"$or": [{"file_name": f} for f in
12            target_filenames]})
13
14    if not clauses:
15        return None

```

```

13     if len(clauses) == 1:
14         return clauses[0]
15     return {"$and": clauses}

```

Κώδικας 4.6: Το φίλτρο πρόσβασης μπαίνει στο ίδιο το ερώτημα προς τη βάση διανυσμάτων

Το πρώτο πράγμα που δημιουργείται και είναι σημαντικό είναι το κομμάτι του φίλτρου πρόσβασης διότι δεν ψάχνει παντού αλλά η συνθήκη μπαίνει μέσα στο ίδιο το ερώτημα προς την βάση. Ζητούνται από την αρχή μόνο τα κομμάτια που είναι δικά του και κρατιέται αυτή η λίστα για όλα τα επόμενα βήματα που ακολουθούν ενώ αν γινόταν το φιλτράρισμα στο τέλος κάποιος μπορεί να άλλαζε την σειρά των βημάτων και θα το προσπερνούσε χωρίς να το καταλάβει κανείς.

Το φίλτρο έχει δύο σκέλη, το πρώτο κοιτάει το δικαίωμα και ισχύει πάντα δηλαδή να είναι δικά του έγγραφα και το δεύτερο μπαίνει σε περίπτωση που ο χρήστης διαλέξει συγκεκριμένα αρχεία περιορίζοντας έτσι την αναζήτηση. Όταν υπάρχουν και τα δύο πρέπει να ισχύουν μαζί, δηλαδή δεν αρκεί να έχει διαλέξει ένα αρχείο, πρέπει και να δικαιούται να κάνει ερώτηση πάνω σε αυτό.

Σε δύο περιπτώσεις η αναζήτηση σταματάει πριν καν αρχίσει. Η πρώτη είναι όταν ο χρήστης έχει ενεργό φίλτρο αρχείων αλλά κανένα δεν στέκει, η δεύτερη όταν το φίλτρο πρόσβασης δεν βρίσκει τίποτα δικό του, και στις δύο περιπτώσεις επιστρέφει άδαιο διότι είναι προτιμότερο να πει το σύστημα ότι δεν το βρήκε παρά να ρίξει στο μοντέλο ό,τι βρέθηκε μπροστά του. Στο ίδιο ερώτημα ζητούνται επίσης μόνο τα αναγνωριστικά των κομματιών ενώ παλιότερα ζητούνταν και τα metadata, τα οποία δεν χρησιμοποιούσε κανένα επόμενο βήμα και για αυτό αφαιρέθηκαν.

Η σειρά με την οποία εκτελούνται τα βήματα δεν έχει βγει τυχαία. Πρώτα μπαίνει η άδεια και το ευρετήριο των λέξεων και μετά έρχεται η κρυφή μετάφραση, δηλαδή η μετάφραση της ελληνικής ερώτησης στα αγγλικά η οποία γίνεται μόνο για τις ανάγκες της αναζήτησης, επειδή τα έγγραφα του corpus είναι στα αγγλικά. Η μετάφραση μπαίνει δεύτερη γιατί το γλωσσάρι πάνω στο οποίο στηρίζεται χτίζεται από τα ίδια τα αρχεία που δικαιούται ο χρήστης, και γιατί αν το σύνολο βγει κενό δεν έχει νόημα να πληρωθεί μια κλήση στο μοντέλο για ερώτηση που δεν πρόκειται να ψάξει ποτέ.

Μετά από αυτό τρέχουν δύο αναζητήσεις πάνω στα ίδια ακριβώς κομμάτια όπου η μία κοιτάει το νόημα των λέξεων και η άλλη τις ακριβείς λέξεις και η καθεμία γυρνάει τη δική της διαφορετική κατάταξη υποψηφίων. Κρατιούνται και οι δύο επειδή η μία συμπληρώνει τις αδυναμίες της άλλης. Το πώς ενώνονται οι δύο κατατάξεις σε μία περιγράφεται στην ενότητα 4.4.4.

Τα βαριά κομμάτια της διαδρομής στέλνονται σε ξεχωριστό νήμα, για τον ίδιο λόγο που στέλνονται και στο γράψιμο του αρχείου. Στο τέλος η συνάρτηση δεν γυρνάει μεμονωμένα κομμάτια αλλά ολόκληρες τις σελίδες στις οποίες ανήκουν αυτά τα κομμάτια.

4.4.4 Συγχώνευση δύο κατατάξεων

Όπως αναφέρθηκε και στην προηγούμενη ενότητα για την αναζήτηση χρησιμοποιούνται δύο τρόποι η νοηματική αναζήτηση και λεξική αναζήτηση οι οποίες βγάζουν κάποια νούμερα και κάποιες κατατάξεις. Αυτά τα νούμερα δεν μπορούν να συγκριθούν μεταξύ τους γιατί το καθένα εννοεί διαφορετικό πράγμα με τα νούμερα που βγάζει και δεν έχουν την ίδια μονάδα μέτρησης ούτε σταθερό εύρος το ίδιο σκορ σημαίνει άλλο πράγμα σε άλλη ερώτηση. Η νοηματική αναζήτηση βγάζει τιμές που μένουν κάτω από την μονάδα ενώ η λεξική δεν έχει κάποιο σταθερό ανώτατο όριο και οι τιμές της εξαρτώνται από την ερώτηση.

Η λύση που χρησιμοποιείται στην διπλωματική για αυτό το πρόβλημα είναι το RRF [10], το οποίο είναι κατάλληλο γιατί αγνοεί εντελώς τα σκορ τα οποία βλέπει και κοιτάει μόνο σε ποια θέση βρίσκεται το κάθε κομμάτι σε κάθε λίστα. Κάθε κομμάτι παίρνει έναν αριθμό από την λίστα ίσο με $\frac{1}{k+r+1}$, όπου r η θέση του στην λίστα και k η σταθερά της μεθόδου δηλαδή 60 όπως προτείνεται στη δημοσίευση ώστε το πρώτο κομμάτι να έχει μικρή διαφορά με το δεύτερο.

```

1 def _rrf_fuse(dense_ids, sparse_ids, all_ids, all_texts, all_metadatas,
2               k: int = 60, top_n: int = 15):
3     lists = [dense_ids, sparse_ids]
4     unique_ids = list(dict.fromkeys([i for lst in lists for i in lst]))
5     rank_maps = [{id_: rank for rank, id_ in enumerate(lst)} for lst in
6                   lists]
7     pos_by_id = {id_: i for i, id_ in enumerate(all_ids)}
8
9     fused = []
10    for doc_id in unique_ids:
11        rrf_score = sum(1 / (k + rm.get(doc_id, 1000) + 1) for rm in
12                        rank_maps)
13        idx = pos_by_id[doc_id]
14        fused.append((rrf_score, doc_id, all_texts[idx], all_metadatas[
15                      idx]))
16
17    fused.sort(key=lambda x: (-x[0], x[1]))
18    return [(s, t, m) for s, _id, t, m in fused[:top_n]]

```

Κώδικας 4.7: Συγχώνευση δύο κατατάξεων με RRF: μόνο η θέση μετράει, με σταθερή σειρά στις ισοβαθμίες

Ένα παράδειγμα θα το κάνει πιο καθαρό, έστω ότι ένα κομμάτι είναι τέταρτο στην μία λίστα και τέταρτο στην άλλη θα προτιμηθεί από ένα άλλο κομμάτι το οποίο είναι πρώτο στην μια λίστα ενώ στην άλλη λίστα απουσιάζει. Κερδίζει το κομμάτι το οποίο βρήκαν και οι δύο αναζητήσεις και όχι το να έχει την καλύτερη θέση στο ένα και να μην υπάρχει στο άλλο. Για αυτό και χρησιμοποιήθηκαν δύο αναζητήσεις καθώς όταν δύο διαφορετικές αναζητήσεις συμφωνούν, η συμφωνία τους λέει κάτι που καμία από τις δύο δεν μπορεί να πει από μόνη της.

Τα κομμάτια τα οποία λείπουν από την μία λίστα δεν αποκλείονται απλά παίρνουν μια πολύ μακρινή θέση η οποία μπορεί να συνεισφέρει αλλά ελάχιστα και ο λόγος είναι ότι με τον αποκλεισμό τους θα έμενε μόνο η τομή των δύο λιστών και θα χάνονταν ακριβώς οι ερωτήσεις για τις οποίες κρατήθηκαν δύο αναζητήσεις, δηλαδή αυτές που τις βρίσκει μόνο η μία. Ένα κομμάτι το οποίο μπορεί να περιέχει κάποια δύσκολη λέξη ή κάποιο σύμβολο θα το έβρισκε μόνο η λεξική αναζήτηση και αν το απέκλειε επειδή δεν το βρήκε η άλλη αναζήτηση τότε το να γίνει η υβριδική αναζήτηση δεν θα είχε κανένα νόημα.

Ένα πρόβλημα που προέκυψε είναι ότι όταν δύο κομμάτια έβγαζαν ακριβώς το ίδιο σκορ η σειρά τους άλλαζε από εκτέλεση σε εκτέλεση, και επειδή η λίστα κόβεται μετέπειτα στα δεκαπέντε άλλαζαν κάθε φορά τα κομμάτια που περνούσαν στο επόμενο βήμα με αποτέλεσμα η ίδια ερώτηση να δίνει διαφορετική απάντηση χωρίς να αλλάξει τίποτα στην ερώτηση. Η λύση ήρθε βάζοντας ένα δεύτερο κριτήριο ταξινόμησης στο αναγνωριστικό του κομματιού το οποίο είναι σταθερό και δίνει κάθε φορά το ίδιο αποτέλεσμα και χωρίς αυτό δεν θα μπορούσε να υπάρξει καμία μέτρηση διότι θα ήταν αναξιόπιστη θα άλλαζε από μέτρηση σε μέτρηση.

Η συνάρτηση δουλεύει με σκέτες λίστες δηλαδή δεν χρειάζονται ούτε βάση ούτε φορτωμένα μοντέλα για να τρέξει και ελέγχεται με απλό test το οποίο τελειώνει σε ένα δευτερόλεπτο. Από αυτό το στάδιο φεύγουν 15 υποψήφια κομμάτια ταξινομημένα και αν έστω ένα από αυτά είναι όντως αρκετά καλό το κρίνει το εργαλείο της επόμενης ενότητας.

4.4.5 Πύλη συνάφειας και δεύτερη προσπάθεια

Η λίστα χρειάζεται να περάσει και από μια τρίτη βαθμολόγηση, επειδή το σκορ το οποίο δίνει το RRF απλά δείχνει ότι συμφώνησαν και οι δύο αναζητήσεις δεν δίνει πληροφορίες για το αν κάποιο από αυτά απαντάει στην ερώτηση. Καμία από τις δύο δεν είχε δει την ερώτηση μαζί με το κείμενο.

Ο cross-encoder είναι η καλύτερη λύση για αυτό το πρόβλημα αλλά έρχεται με το ανάλογο κόστος, δεν μπορεί να υπολογιστεί εκ των προτέρων και σε δεκαπέντε κομμάτια είναι εφικτό να τρέξει ενώ σε χιλιάδες όχι για αυτό και τα γρήγορα βήματα μαζεύουν τους λίγους υποψήφιους και τους δίνουν στο ακριβό βήμα για να κρίνει τους λίγους σωστά.

Τα νούμερα που βγαίνουν από την μέτρηση είναι ωμά logits, από το $-9,4$ έως το $+3,8$ στις μετρήσεις που έγιναν, και δεν έχουν καμία σχέση με πιθανότητες ούτε με ποσοστά. Χωρίς αυτή τη διευκρίνιση ένα αρνητικό νούμερο θα φαινόταν λάθος, ενώ είναι φυσιολογική τιμή της κλίμακας.

```

1 sorted_final = sorted(final_combined, key=lambda x: x[0], reverse=
  sorted_final)
2
3 if sorted_final[0][0] < MIN_RERANK_SCORE:
4     metrics.inc("rag_gate_blocked_total")
5     sorted_final = await _corrective_retry(
6         query, sorted_final[0][0], allowed_ids, idx, dm)
7     if sorted_final is None:
8         return []
9
10    return await asyncio.to_thread(
11        _expand_to_pages, sorted_final[:EXPAND_INPUT], MAX_PAGES,
12        user_id)

```

Κώδικας 4.8: Πύλη συνάφειας: κόψιμο πριν φτάσει οτιδήποτε στο μοντέλο και δεύτερη προσπάθεια μόνο στη διαδρομή της αποτυχίας

Αν το καλύτερο σκορ από αυτή την μέτρηση δεν περάσει το κατώφλι δεν στέλνεται τίποτα στο μοντέλο. Στην θέση της απάντησης μπαίνει μήνυμα ότι δεν βρέθηκε σχετικό κείμενο στα συγκεκριμένα έγγραφα, και αυτό γίνεται επειδή από μόνο του ένα γλωσσικό μοντέλο που παίρνει άσχετο κείμενο δεν απαντάει «δεν ξέρω» αλλά δίνει μια πειστική απάντηση από τις δικές του γνώσεις που έμαθε κατά την εκπαίδευση και όχι από το αρχείο. Η πύλη συνάφειας είναι το πρώτο στάδιο άμυνας, υπάρχει και δεύτερο, οι οδηγίες που δίνονται στο μοντέλο.

Κανένα κατώφλι δεν κόβει πάντα μόνο τα άσχετα. Μπορεί ένα κομμάτι που όντως θα απαντούσε την ερώτηση να πέσει λίγο κάτω από την γραμμή και να πεταχτεί μαζί με τα υπόλοιπα, οπότε πριν το σύστημα πει ότι δεν βρήκε τίποτα δίνεται στην ερώτηση μια δεύτερη ευκαιρία [16].

```

1 async def _corrective_retry(query, first_best, allowed_ids, idx, dm):
2     if not ENABLE_CORRECTIVE:
3         return None
4     try:
5         new_query = (await gemini_rest.generate_once(

```

```

6         _CORRECTIVE_PROMPT.format(query=query))).strip(" \'\n")
7     except Exception:
8         return None
9     if not new_query or new_query.strip().lower() == query.strip().
lower():
10         return None
11
12     # ... Όλη η διαδρομή ξανα, με τη νέα διατύπωση:
13     # dense + sparse -> _rrf_fuse -> reranker -> sorted_final ...
14
15     if sorted_final[0][0] < CORRECTIVE_MIN_SCORE:
16         return None
17     return sorted_final

```

Κώδικας 4.9: Δεύτερη προσπάθεια: Ξαναγράψιμο της ερώτησης, πλήρης επανάληψη της διαδρομής και χαλαρότερο κατώφλι

Η ερώτηση φεύγει στο μοντέλο ώστε να ξαναγραφτεί με όρους που θα χρησιμοποιούσε ένα άρθρο του χώρου δηλαδή με ονομαστικές φράσεις και μετά ξανατρέχει ολόκληρη την διαδρομή με την νέα διατύπωση και αν το ξαναγράψιμο γυρίσει την ίδια ερώτηση τότε η συνάρτηση σταματάει αμέσως γιατί το ίδιο ερώτημα δίνει το ίδιο αποτέλεσμα. Οποιαδήποτε άλλη αποτυχία, από σφάλμα δικτύου μέχρι άκυρη απάντηση, καταλήγει επίσης σε None δηλαδή σε μια συμπεριφορά που θα είχε το σύστημα και χωρίς αυτή την λειτουργία και ο διακόπτης της πρώτης γραμμής τη σβήνει ολόκληρη και χωρίς αυτόν δεν θα μπορούσε να μετρηθεί το πόσο προσέφερε. Κάτι ακόμα, το κατώφλι για την δεύτερη προσπάθεια είναι πιο ελαστικό από το πρώτο επειδή εδώ η εναλλακτική δεν είναι να δώσει μια καλύτερη απάντηση αλλά να μην δώσει καθόλου.

4.4.6 Από τα κείμενα στην απάντηση

Πριν φτάσει οτιδήποτε στο μοντέλο τα κείμενα τα οποία πέρασαν το κατώφλι μπαίνουν σε σειρά και το κάθε κομμάτι φοράει επικεφαλίδα που το ονομάζει: S1, S2 και ούτω καθεξής. Αυτή η ετικέτα είναι το αγκίστρι της παραπομπής.

```

1 SOURCE_HEADER = "[S{i}: {file}, Page: {page}]"
2
3 CITATION_RULE = (
4     "End every factual sentence with the label of its source in square
5     "
6     "brackets, copying it from the [S...: ...] header of the passage
7     "
8     "used, like this: [S3]. Write ONLY the label - no file name, no
9     "
10    "number, no other text inside the brackets. ONE label per bracket:
11    "
12    "sentence draws on two passages, write two brackets. Cite ONLY
13    "
14    "labels "
15    "that appear in the SOURCE TEXT, and do not add a bibliography at
16    "
17    "the end."
18 )
19
20 for i, (text, meta) in enumerate(top_3_data, 1):
21     file_n = meta.get("file_name", "Unknown File")
22     page_n = meta.get("page", "?")
23     header = SOURCE_HEADER.format(i=i, file=file_n, page=page_n)
24     context_text += f"\n{header}\n{text}\n"

```

```

17     sources_list.append({
18         "label": f"S{i}",
19         "file": file_n,
20         "page": page_n,
21         "preview": text[:400],
22     })

```

Κώδικας 4.10: Χτίσιμο του κειμένου πηγών: η ετικέτα μπαίνει στην επικεφαλίδα και ταξιιδεύει μαζί με το αντικείμενο προς τη διεπαφή

Ο κανόνας που δίνεται στο μοντέλο ζητάει να γραφτεί η παραπομπή στο τέλος κάθε πρότασης και ζητάει μόνο την ετικέτα μέσα σε αγκύλες όχι όνομα αρχείου, όχι αριθμό σελίδας, τίποτε άλλο. Η ετικέτα αυτή είναι δείκτης μέσα σε όσα δόθηκαν οπότε αν δεν υπάρχει η πηγή δεν μπορεί καν να γραφτεί. Έτσι η επινοημένη παραπομπή δεν αποτρέπεται απλώς, γίνεται αδύνατη.

Ο λόγος που δεν ζητείται σελίδα βγήκε από μέτρηση και αυτό διότι όταν ζητούνταν και η σελίδα η ετικέτα ήταν σωστή και στις ενενήντα επτά περιπτώσεις ενώ ο αριθμός της σελίδας βγήκε λάθος δέκα φορές γιατί το μοντέλο έγραφε τον αριθμό που έβλεπε μέσα στο κείμενο αντί για αυτόν της επικεφαλίδας.

Κάθε πηγή κρατάει την ετικέτα της μέσα στο ίδιο αντικείμενο, μαζί με το αρχείο, την σελίδα και ένα απόσπασμα του κειμένου έτσι ώστε να μπορεί ο χρήστης να επαληθεύσει από πού βγήκε η απάντηση.

Το τελικό κείμενο που φεύγει έχει σταθερή σειρά που αυτή είναι ότι πρώτα μπαίνουν οι κανόνες, μετά τα κείμενα των πηγών, μετά το πιο πρόσφατο ιστορικό, μετά η ερώτηση που έχει γίνει και στο τέλος μια υπενθύμιση για την γλώσσα. Τη γλώσσα την καθορίζει ο κώδικας κοιτάζοντας αν η ερώτηση έχει ελληνικούς χαρακτήρες και αυτό γιατί χωρίς αυτό μια αγγλική ερώτηση μέσα σε ελληνική συνομιλία θα απαντιόταν στα ελληνικά επειδή το ιστορικό είχε περισσότερες ελληνικές ερωτήσεις παρά αγγλικές.

```

1 lang_rule = GREEK_RULE if _has_greek(question) else ENGLISH_RULE
2
3 full_prompt = (
4     f"{system_prompt}\n\n"
5     f"--- SOURCE TEXT ---\n{context_text}\n\n"
6     f"{history_text}"
7     f"CURRENT QUESTION: {question}\n\n"
8     f"REMINDER: {lang_rule}\n\n"
9     f"ANSWER:"
10 )

```

Κώδικας 4.11: Συναρμολόγηση του prompt: η οδηγία γλώσσας μπαίνει τελευταία, μετά την ερώτηση

Η απάντηση δεν στέλνεται ολόκληρη αλλά φεύγει σε κομμάτια και κάθε γραμμή είναι ένα από τρία είδη, οι πηγές, ένα κομμάτι κειμένου ή οι μετρήσεις της εκτέλεσης. Η σειρά τους δεν είναι τυχαία, οι πηγές φεύγουν πρώτες πριν γραφτεί το οτιδήποτε ώστε ο χρήστης να βλέπει σε τι στηρίζεται όσο περιμένει την απάντηση και οι μετρήσεις είναι το μόνο που δεν αποθηκεύεται γιατί αφορούν μόνο την συγκεκριμένη εκτέλεση και όχι το γενικό περιεχόμενο.

```

1 async def event_stream():
2     full_answer = ""
3     final_sources = []
4     try:
5         async for packet in ai_core.ask_ai(

```

```

6         question=query.question, history=query.history,
7         target_filenames=query.target_filenames,
8         user_id=current_user.id, persona=query.persona):
9         if packet["type"] == "sources":
10             final_sources = packet["data"]
11             yield f"{json.dumps({'type': 'sources', 'data':
final_sources})}\n"
12         elif packet["type"] == "text":
13             full_answer += packet["data"]
14             yield f"{json.dumps({'type': 'text', 'data': packet['
data']})}\n"
15         elif packet["type"] == "metrics":
16             yield f"{json.dumps({'type': 'metrics', 'data': packet
['data']})}\n"
17         finally:
18             if full_answer:                                     # τρέχει ΚΑΙ στη διακοπή
19                 db.add(models.Message(
20                     conversation_id=query.conversation_id, role="assistant",
21
22                     content=full_answer, sources=json.dumps(final_sources))
23                 )
24                 db.commit()
25
26 return StreamingResponse(event_stream(), media_type="application/x-
ndjson")

```

Κώδικας 4.12: Ροή της απάντησης σε γραμμές JSON και αποθήκευση όσων προλάβαν να παραχθούν

Το σώσιμο της απάντησης δεν γίνεται όταν τελειώσει η ροή αλλά εκτελείται πάντα και στην ουσία σώζει ό,τι πρόλαβε να παραχθεί δηλαδή αν ο χρήστης κλείσει την σελίδα στην μέση ό,τι έχει προλάβει να γραφτεί μπαίνει κανονικά στο ιστορικό της συνομιλίας. Χωρίς αυτό μια απάντηση που έχει κοστίσει ήδη χρόνο και κλήσεις σε μοντέλα θα χανόταν ολόκληρη επειδή έπεσε το ίντερνετ του χρήστη ή επειδή έφυγε από την σελίδα.

4.5 Υλοποίηση του frontend

4.5.1 Ένα σημείο επικοινωνίας με τον server

Το frontend δεν βλέπει ούτε βάση ούτε αρχείο και ό,τι χρειάζεται το ζητάει από τον server μέσω HTTP, ενώ όλες αυτές οι κλήσεις προς το backend περνάνε από μία μόνο συνάρτηση. Αν κάθε σημείο της οθόνης έγραφε την δική του κλήση τότε το χρονικό όριο και ο χειρισμός των σφαλμάτων θα ήταν σκορπισμένα σε δεκαπέντε σημεία και το δέκατο έκτο σημείο θα ξεχνιόταν.

```

1 API_URL = os.getenv("BACKEND_URL", "http://localhost:8000")
2
3 TIMEOUT = (5, 30)
4 TIMEOUT_UPLOAD = (5, 120)
5
6
7 def api(method, path, *, timeout=TIMEOUT, **kwargs):
8     try:
9         return requests.request(method, f"{API_URL}{path}",
10                                timeout=timeout, **kwargs)
11     except requests.exceptions.RequestException:

```

```

12         return None
13
14
15     # Ο έλεγχος για None γίνεται ΠΑΝΤΑ πριν το status_code.
16     msg_res = api("GET", f"/conversations/{conv_id}/messages", headers=
17         headers)
18     if msg_res is None:
19         st.error("Could not load history. Check that the backend is running.
20             ")
21     elif msg_res.status_code == 200:
22         st.session_state.chat_history = msg_res.json()

```

Κώδικας 4.13: Ενιαίο σημείο κλήσης του backend, με χρονικά όρια και χωρίς εξαιρέσεις προς τη σελίδα

Η συνάρτηση αυτή δεν πετάει ποτέ σφάλμα, αν ο server δεν απαντήσει καθόλου επιστρέφει κενό και ο κώδικας το ελέγχει πριν κάνει το οτιδήποτε άλλο και ο λόγος που γίνεται είναι ότι στο Streamlit ένα σφάλμα δικτύου ανεβαίνει ως κόκκινο μήνυμα και δεν σταματάει μόνο το αίτημα που έχει γίνει αλλά το χτίσιμο ολόκληρης της σελίδας. Δηλαδή ο χρήστης δεν χάνει απλά το αίτημα το οποίο απέτυχε αλλά χάνει από μπροστά του και την συνομιλία που χρησιμοποιούσε. Στην φόρτωση του ιστορικού αυτό φαίνεται ξεκάθαρα, αν αποτύχει εμφανίζεται ένα μήνυμα ότι δεν φορτώθηκε η συνομιλία και η υπόλοιπη σελίδα τρέχει κανονικά.

Κάθε κλήση η οποία γίνεται έχει χρονικό όριο γιατί χωρίς όριο η βιβλιοθήκη θα περίμενε για πάντα και επειδή το Streamlit κατά την σύνδεση δίνει ένα νήμα σε κάθε σύνδεση του χρήστη μια κολλημένη κλήση παγώνει την καρτέλα χωρίς να εμφανιστεί τίποτα. Ο λόγος που το όριο είναι απαραίτητο φαίνεται αν ξεχωριστούν δύο βλάβες. Όταν ο server είναι πεσμένος η σύνδεση απορρίπτεται αμέσως, η βιβλιοθήκη το καταλαβαίνει και βγάζει σφάλμα και ο κώδικας το πιάνει κανονικά.

Υπάρχει όμως και η περίπτωση που ο server ζει αλλά δεν απαντάει και αυτή είναι η πιο πιθανή περίπτωση να συμβεί. Στο ξεκίνημα το backend φορτώνει 2,4 gigabyte μοντέλων και για αυτό του δίνεται περιθώριο πέντε λεπτών στον έλεγχο υγείας των containers, ενώ σε αυτό το διάστημα δέχεται συνδέσεις χωρίς να απαντάει σε καμία κάτι που ισχύει και όταν είναι υπό φόρτο, αφού ο κορεσμός μετρήθηκε ήδη στους τέσσερις ταυτόχρονους χρήστες. Και στις δύο περιπτώσεις δεν υπάρχει κανένα σφάλμα για να πιαστεί και η σύνδεση άνοιξε κανονικά απλά δεν έρχεται τίποτα πίσω.

Για αυτό το χρονικό όριο σώζει την κατάσταση μετατρέποντας την από ατέρμονη αναμονή σε σφάλμα ώστε να μπορεί σε αυτήν την περίπτωση να την πιάσει ο κώδικας και να δείξει μήνυμα. Τα όρια όμως δεν είναι ίδια παντού. Το ανέβασμα ενός αρχείου φτάνει τα πενήντα megabyte και περιμένει έως δύο λεπτά, ενώ οι υπόλοιπες κλήσεις έχουν όριο τριάντα δευτερόλεπτα, κάτι το οποίο είναι πολύ μεγαλύτερο από αυτό που μετρήθηκε στο πείραμα ταυτόχρονης χρήσης, όπου η ανάκτηση με οκτώ ταυτόχρονους χρήστες έμεινε κάτω από δέκα δευτερόλεπτα (κεφάλαιο 6).

4.5.2 Επανεκτέλεση σελίδας και ροή απάντησης

Το Streamlit δεν ενημερώνει κομμάτια της σελίδας αλλά χτίζει ολόκληρη την σελίδα από την αρχή σε κάθε του κλικ τρέχοντας όλο το αρχείο ξανά. Για μια φόρμα αυτό είναι βολικό και βοηθάει αλλά για μια απάντηση η οποία γράφεται γράμμα γράμμα το πάτημα οποιουδήποτε κουμπιού την διακόπτει στην μέση.

Η ερώτηση του χρήστη δεν στέλνεται τη στιγμή που γράφεται, πρώτα αποθηκεύεται, ζητείται μια επανεκτέλεση και μόνο στο επόμενο πέρασμα φεύγει προς τον server. Το ενδιαμέσο βήμα χρειάζεται επειδή για να φαίνονται κλειδωμένα τα κουμπιά πρέπει πρώτα να ξαναζωγραφιστούν, και αυτό το κάνει μόνο μια επανεκτέλεση.

Το πλαίσιο στο οποίο γράφει ο χρήστης μπαίνει στον κώδικα πριν από το κομμάτι που τρέχει τη ροή και αυτό γιατί αν έμπαινε μετά δεν θα προλάβαινε να ζωγραφιστεί κλειδωμένο αφού όσο γράφεται η απάντηση η εκτέλεση μένει σταματημένη οπότε η οθόνη θα ήταν στην προηγούμενη ξεκλειδωτή μορφή και ο χρήστης θα μπορούσε να κόψει την απάντηση.

```

1 generating = st.session_state.get("pending_question") is not None
2
3 # Ιστορικο απο τον server - OXI ομως όσο τρεχει απαντηση.
4 if st.session_state.current_conv_id and not generating:
5     ... # η φορτωση ιστορικού της προηγουμενης
6     ενοτητας
7
8 # ... μηνυματα και πλαινη στηλη: ολα με disabled=generating ...
9 render_library(headers, token, generating=generating) # auto-refresh
10 σε παυση
11
12 # Το input ζωγραφιζεται PPIN το streaming block.
13 user_question = st.chat_input("Ask your question...", disabled=
14     generating)
15
16 if user_question and not generating:
17     st.session_state.pending_question = user_question
18     st.rerun()
19
20 if generating:
21     st.button("Stop", key="stop_generation") # το μονο ενεργο widget
22
23 if st.session_state.get("pending_question"):
24     q = st.session_state.pending_question
25     # Ξαναμνηκαμε ενω η αποστολη ειχε ηδη γινει -> MHN ξαναστειλεις.
26     if st.session_state.get("pending_inflight"):
27         st.session_state.pending_inflight = False
28         st.session_state.pending_question = None
29         st.rerun()
30
31     st.session_state.pending_inflight = True
32     # ... POST /chat και st.write_stream(...) ...
33     st.session_state.pending_inflight = False
34     st.session_state.pending_question = None
35     st.rerun()

```

Κώδικας 4.14: Κλείδωμα της οθόνης, καθυστερημένη αποστολή και προστασία από διπλή αποστολή

Το μόνο κουμπί το οποίο μένει ενεργό είναι αυτό της διακοπής της απάντησης και δουλεύει εξαιτίας της επανεκτέλεσης που αναφέρθηκε, κάτι που είναι πρόβλημα παντού αλλά εδώ είναι ακριβώς ο μηχανισμός που επιτρέπει την διακοπή. Στην εμφάνιση δεν αλλάζει τίποτα γιατί το πλαίσιο είναι πάντα στο κάτω μέρος.

Υπάρχει και ένας έλεγχος που κρατάει σημείωση ότι η αποστολή έχει γίνει ήδη. Αν το πρόγραμμα ξαναπεράσει από εκεί ενώ η απάντηση έχει ήδη ζητηθεί, δεν ξαναστέλνει τίποτα, καθαρίζει την κατάσταση και ξαναφορτώνει την συνομιλία από τον server. Χωρίς αυτόν τον έλεγχο μια διακοπή στην μέση ξαναέστειλε την ίδια ερώτηση και εμφανίζονταν

διπλά μηνύματα με ανακατεμένη σειρά, ενώ τώρα δεν χάνεται τίποτα γιατί ό,τι είχε προλάβει να γραφτεί το έχει ήδη σώσει ο server.

Για τον ίδιο λόγο το ιστορικό ξαναφορτώνεται από τον server σε κάθε πέρασμα γιατί χωρίς αυτό θα έδειχνε μόνο το μισό ζευγάρι, αφού ο server έχει σώσει την ερώτηση αλλά όχι ακόμα την απάντηση. Την ίδια λογική ακολουθεί και η βιβλιοθήκη εγγράφων, η οποία ανανεώνεται μόνη της κάθε τέσσερα δευτερόλεπτα αλλά σταματάει αυτή η ανανέωση όσο γράφεται η απάντηση γιατί έτσι και αλλιώς δεν μπορεί να γίνει κάτι εκείνη την ώρα εκτός από το να σταματήσει η απάντηση.

4.5.3 Ο μετρητής που δεν παγώνει

Όσο τρέχει η ανάκτηση και γράφεται η απάντηση, το πρόγραμμα περιμένει σε μια γραμμή και διαβάζει τη ροή που στέλνει ο server, και εκεί η εκτέλεση είναι σταματημένη και δεν μπορεί να ζωγραφίσει τίποτα. Ένας μετρητής ο οποίος θα ήταν γραμμένος σε Python θα έγραφε μια φορά τον αριθμό και μετά θα ήταν παγωμένος ακριβώς την ώρα που ο χρήστης θα ήθελε να δει αν γίνεται κάτι.

```

1 message_placeholder = st.empty()
2 t_start = time.perf_counter()
3
4 def show_status(label):
5     # Ο χρόνος τρέχει στον browser, ΟΧΙ σε Python.
6     base = time.perf_counter() - t_start
7     with message_placeholder.container():
8         components.html(f"""
9             <div>{label}</div><div id='zt'></div>
10            <script>
11                const base = {base}, t0 = Date.now();
12                const el = document.getElementById('zt');
13                setInterval(() => {{ el.textContent =
14                    (base + (Date.now()-t0)/1000).toFixed(1) + 's'; }},
15                    100);
16            </script>
17            """, height=36)
18 show_status("Searching your documents...")
19 with requests.post(f"{API_URL}/chat", json=payload, headers=headers,
20                 stream=True, timeout=(10, 180)) as response:
21     for line in response.iter_lines(): # Εδώ μπλοκάρει το
22         νημα
23         data = json.loads(line.decode("utf-8"))
24         if data["type"] == "sources":
25             show_status("Composing answer...") # η2 ψαση, χωρίς αλμα
26             στο 0
27         elif data["type"] == "text":
28             if first_chunk:
29                 message_placeholder.empty() # σβηνει το iframe ->
30                 σταματα
31                 first_chunk = False
32                 for tok in re.findall(r"\S+|\s+", data["data"]):
33                     yield tok
34                     if tok.strip():
35                         time.sleep(0.012)

```

Κώδικας 4.15: Ζωντανή ένδειξη χρόνου στον browser, επειδή το νήμα της Python είναι μπλοκαρισμένο

Και αυτός είναι ο λόγος για τον οποίον ο χρόνος δεν μετριέται στον server αλλά στον browser και στην θέση της απάντησης μπαίνει ένα μικρό κομμάτι το οποίο είναι γραμμένο σε HTML και JavaScript που ανεβάζει τον αριθμό δέκα φορές το δευτερόλεπτο, αφού ο browser δεν περιμένει τίποτα και δουλεύει κανονικά. Παίρνει μαζί και τον χρόνο που πέρασε ήδη γιατί όταν αλλάζει το μήνυμα ο μετρητής φτιάχνεται από την αρχή και αλλιώς θα ξεκινούσε πάλι από το μηδέν.

Το βολικό με αυτό είναι ότι σταματάει από μόνο του. Βρίσκεται μέσα στο ίδιο κουτί όπου θα γραφτεί και η απάντηση και έτσι με το που γραφτεί το πρώτο κομμάτι κειμένου το κουτί αδειάζει και ο μετρητής χάνεται και αυτός.

Το μήνυμα που εμφανίζεται αλλάζει μόνο μια φορά, από την αναζήτηση στην σύνταξη. Αυτό δεν κόστισε τίποτα, γιατί ο server στέλνει πρώτα τις πηγές και μετά το κείμενο, οπότε μόλις φτάσουν οι πηγές γίνεται γνωστό ότι η αναζήτηση τελείωσε. Η κλήση έχει ένα δικό της όριο, 180 δευτερόλεπτα αντί για 30, και μετράει πόσο μένει η σύνδεση χωρίς να έρθει κομμάτι κειμένου, όχι πόσο κρατάει συνολικά η απάντηση.

Το κείμενο δεν έρχεται ομοιόμορφα από το μοντέλο και αν φαινόταν έτσι θα πεταγόταν με άλματα για αυτό και σπάει σε λέξεις από τις οποίες κάθε λέξη βγαίνει με μια ελάχιστη καθυστέρηση ώστε να μοιάζει με κάτι που γράφεται εκείνη την στιγμή.

4.6 Εργαλεία

4.6.1 Τι περιέχει το αποθετήριο

Όλος ο κώδικας βρίσκεται σε ένα δημόσιο αποθετήριο στο GitHub, όπου υπάρχουν τα πάντα που αφορούν τον κώδικα του συστήματος, δηλαδή το backend, η διεπαφή, οι αυτοματισμοί έλεγχοι, τα scripts με τα οποία έγιναν οι μετρήσεις του κεφαλαίου 6 και οι οδηγίες εγκατάστασης.

```

1 local-rag-assistant/
2 |-- backend/
3 |   |-- ai_core.py           # αναζητηση, συντηξη, gate, απαντηση
4 |   |-- main.py             # τα endpoints
5 |   |-- evaluation/         # τα scripts των μετρησεων του Κεφ. 6
6 |   |-- tests/
7 |-- frontend/
8 |   |-- app_ui.py
9 |-- .github/workflows/ci.yml # οι αυτοματισμοι ελεγχοι
10 |-- docker-compose.yml      # οι τρεις υπηρεσιες
11 |-- .env.example             # υπόδειγμα - τα κλειδια ΔΕΝ ανεβαινουν
12 |-- DEPLOY.md               # οδηγιες εγκαταστασης
13 |-- ARCHITECTURE.md
14 |-- ENGINEERING_LOG.md
15
16 # .gitignore - τι μενει εκτός, και γιατι:
17 .env                         # τα πραγματικα κλειδια
18 uploaded_docs/              # τα PDF των χρηστων
19 vector_db/                  # ετοιμη Chroma εσπερνε παλιά δεδομενα στο volume

```

Κώδικας 4.16: Το περιεχόμενο του αποθετηρίου και όσα μένουν σκόπιμα εκτός

Τρία πράγματα δεν υπάρχουν στο GitHub και το καθένα έχει τον δικό του λόγο. Για αρχή τα πραγματικά κλειδιά δεν ανεβαίνουν ποτέ εκεί για λόγους ασφαλείας και στην θέση τους υπάρχει ένα υπόδειγμα με τα ονόματα χωρίς όμως να περιέχει τις τιμές. Δεύτερον τα PDF που ανεβάζει ο κάθε χρήστης δεν ανήκουν στον κώδικα οπότε δεν υπάρχει κάποιος λόγος να είναι εκεί και τέλος η έτοιμη βάση διανυσμάτων εξαιρέθηκε αφού πρώτα δημιούργησε πρόβλημα, γιατί όσο ήταν μέσα το docker την αντέγραφε στον αποθηκευτικό του χώρο και το σύστημα ξεκινούσε παντού με παλιά κομμάτια κειμένου που κανείς δεν τα είχε ανεβάσει.

4.6.2 Έλεγχοι που τρέχουν σε κάθε ανέβασμα

Κάθε φορά που ανεβαίνει μια αλλαγή στο αποθετήριο, το GitHub τρέχει από μόνο του αυτούς τους ελέγχους, όχι όμως όλους μαζί αλλά σε τρεις ξεχωριστές ομάδες οι οποίες είναι χωρισμένες ανάλογα με τον χρόνο που θέλει η καθεμία για να γίνει. Η μία από τις τρεις πρέπει να κατεβάσει και να φορτώσει τα μοντέλα ενώ οι άλλες δύο τελειώνουν σε δευτερόλεπτα οπότε δεν αποτελούν πρόβλημα. Αν όμως ήταν όλες μαζί κάθε μικρή αλλαγή θα περίμενε πρώτα τα μοντέλα να φορτώσουν ώστε να μάθει κάτι που φαίνεται άμεσα.

```

1 name: tests
2 on: [push, pull_request]
3
4 jobs:
5   lint:                                # ~10 δευτ.
6     runs-on: ubuntu-latest
7     steps:
8       - uses: actions/checkout@v4
9       - run: pip install ruff
10      - run: ruff check .
11        working-directory: backend
12
13   fast-tests:                          # ~40 δευτ. - KANENA μοντελο
14     runs-on: ubuntu-latest
15     steps:
16       # ... checkout, python 3.11, ελαφρια deps ...
17       - run: |
18         pytest tests/test_schemas.py tests/test_security.py \
19           tests/test_gemini_rest.py tests/test_api_endpoints.py
20       \
21         tests/test_metrics.py tests/test_rate_limit.py -v
22
23   core-tests:                          # 2-8 λεπτα - φορτωνει 2.4GB
24     runs-on: ubuntu-latest
25     env:
26       HF_HOME: ${GITHUB_WORKSPACE}/.hf-cache
27     steps:
28       # ... checkout, python 3.11 ...
29       - uses: actions/cache@v4          # τα μοντελα κρατιουνται αναμεσα
30       στα run
31       with:
32         path: ${GITHUB_WORKSPACE}/.hf-cache
33         key: hf-${GITHUB_SHA}
34       - run: pip install torch --index-url https://download.pytorch.org
35         /whl/cpu
36       - run: pip install -r backend/requirements.txt
37       - run: |
38         pytest tests/test_rag_core.py tests/test_corpus_signature.py
39       \

```

```

36 tests/test_corrective_retrieval.py tests/
test_citations.py -v

```

Κώδικας 4.17: Αυτόματοι έλεγχοι χωρισμένοι κατά χρόνο εκτέλεσης

Η πρώτη ομάδα δεν κοιτάει καθόλου αν δουλεύει ο κώδικας αλλά μόνο το πώς είναι γραμμένος και διαρκεί για δέκα δευτερόλεπτα. Η δεύτερη ελέγχει ό,τι μπορεί να ελεγχθεί χωρίς μοντέλα δηλαδή τα σχήματα των δεδομένων, την ασφάλειά τους, το όριο αιτημάτων και τα σημεία του API, ανάμεσά τους και ότι ένας χρήστης δεν φτάνει ποτέ στα αρχεία κάποιου άλλου. Η τρίτη είναι ο πυρήνας της ανάκτησης και θέλει τον περισσότερο χρόνο, από δύο έως και οκτώ λεπτά, ακριβώς επειδή εκεί φορτώνονται τα μοντέλα.

Υπάρχει ένας τρόπος να γίνει η τρίτη ομάδα πιο γρήγορη και αυτός είναι αν οι έλεγχοι έγραφαν την δική τους έκδοση της αναζήτησης αντί να καλούν την πραγματική αλλά αυτό θα ήταν ψεύτικη ασφάλεια διότι μια αντιγραμμένη λογική θα περνούσε ακόμα και με σπασμένο τον πραγματικό κώδικα και θα έδειχνε πράσινο ενώ είναι λάθος. Για να μην γίνει απαγορευτικός ο χρόνος τα μοντέλα κρατιούνται σε αποθήκη ανάμεσα στις εκτελέσεις και η βιβλιοθήκη υπολογισμών εγκαθίσταται στην έκδοση χωρίς κάρτα γραφικών, αφού σε αυτά τα μηχανήματα κάρτα γραφικών δεν υπάρχει, και έτσι γλυτώνει δυόμισι gigabyte.

4.7 Εγκατάσταση και εκτέλεση σε άλλον υπολογιστή

Το μόνο που χρειάζεται κάποιος για να τρέξει το σύστημα στον υπολογιστή του είναι το Docker. Δεν χρειάζεται να εγκαταστήσει Python ούτε κάποια βάση δεδομένων ούτε να κατεβάσει κάποιο μοντέλο με το χέρι. Αντιγράφει τον κώδικα, συμπληρώνει ένα αρχείο με τρεις τιμές και δίνει μια εντολή. Ο λόγος που στήθηκε έτσι είναι ότι η εφαρμογή δεν είναι ένα πρόγραμμα αλλά τρία ξεχωριστά κομμάτια τα οποία πρέπει να βρουν το ένα το άλλο και χωρίς containers οι οδηγίες θα ήταν μια σελίδα προαπαιτούμενων ανά λειτουργικό.

```

1 git clone https://github.com/kostas221/local-rag-assistant.git
2 cd local-rag-assistant
3
4 cp .env.example .env
5 # συμπληρώνεις: GEMINI_API_KEY, POSTGRES_PASSWORD, SECRET_KEY
6
7 docker compose up --build -d          # πρώτη φορά: αρκετά λεπτά
8
9 docker compose ps                      # backend -> "healthy"
10 curl http://127.0.0.1:8010/health
11 # UI: http://localhost:8502
12
13 docker compose down                   # σταματά, τα δεδομένα ΜΕΝΟΥΝ
14 docker compose down -v                # σβηνει ΚΑΙ τα δεδομένα

```

Κώδικας 4.18: Η ακολουθία που στήνει και τρέχει το σύστημα σε καινούριο μηχανήμα

Οι τρεις τιμές που συμπληρώνει ο χρήστης είναι το κλειδί του Gemini που χρειάζεται για να γίνει η απάντηση, ο κωδικός της βάσης και το κλειδί με το οποίο υπογράφονται τα tokens με τα οποία ο χρήστης αποδεικνύει ποιος είναι σε κάθε αίτημα. Κανένα από αυτά δεν υπάρχει μέσα στον κώδικα και το συγκεκριμένο αρχείο δεν ανεβαίνει ποτέ στο GitHub. Αν λείπει κάποιο, το backend σταματάει με μήνυμα αντί να ξεκινήσει με μια προεπιλογή που θα ήταν για όλους ίδια.

Την πρώτη φορά η εντολή κρατάει για κάποια λεπτά επειδή κατεβαίνουν και φορτώνονται 2,4 gigabyte από διάφορα μοντέλα και δεν είναι κάποιο κόλλημα απλά έτσι δουλεύουν αυτά τα συστήματα.

```

1 services:
2   db:
3     ports:
4       - "127.0.0.1:5434:5432"      # OXI εκτεθειμενο στο δικτυο
5     volumes:
6       - postgres_data:/var/lib/postgresql/data
7
8   backend:
9     ports:
10      - "127.0.0.1:8010:8000"      # OXI δημοσιο
11    volumes:
12      - chromadb_data:/app/vector_db
13      - huggingface_cache:/home/appuser/.cache/huggingface
14    healthcheck:
15      start_period: 300s           # φορτωνει 2.4GB μοντελων
16
17    frontend:
18      ports:
19        - "8502:8501"              # το MONO που βγαινει εξω
20      depends_on:
21        backend:
22          condition: service_healthy

```

Κώδικας 4.19: Οι ρυθμίσεις πίσω από την παραπάνω εντολή: θύρες, αναμονή, αποθήκευση

Ο έλεγχος υγείας δίνει περιθώριο πέντε λεπτών πριν πει ότι κάτι δεν πάει καλά, ενώ η διεπαφή δεν ξεκινάει καθόλου όσο το backend δεν έχει δηλώσει ότι είναι έτοιμο. Χωρίς αυτή την λεπτομέρεια ο χρήστης θα έβλεπε την οθόνη να ανοίγει κανονικά και κάθε ερώτηση να μην δουλεύει.

Τα δεδομένα δεν αποθηκεύονται μέσα στα containers αλλά σε τρεις χωριστούς χώρους δίπλα τους, έναν για την βάση, έναν για τα διανύσματα και έναν για τα μοντέλα, κάτι που δίνει δύο αποτελέσματα. Πρώτον το δεύτερο ξεκίνημα γίνεται σε δευτερόλεπτα επειδή τα μοντέλα έχουν ήδη κατέβει από την πρώτη φορά, ενώ όταν αλλάζει ο κώδικας τα αρχεία και οι λογαριασμοί μένουν στην θέση τους. Δεύτερον οι δύο εντολές οι οποίες σταματούν το σύστημα δεν είναι ίδιες, η πρώτη σβήνει μόνο τις υπηρεσίες ενώ η δεύτερη παίρνει μαζί και όλα όσα έχουν ανέβει.

Από τις τρεις υπηρεσίες μόνο η διεπαφή εκτίθεται στο δίκτυο. Η βάση και το API ακούν μόνο την διεύθυνση του ίδιου του υπολογιστή. Αν λοιπόν το μηχανήμα βρίσκεται σε δίκτυο με άλλους κανέναν από αυτούς δεν μπορεί να χτυπήσει κατευθείαν τη βάση ή το API χωρίς να περάσει από σύνδεση με λογαριασμό.

Κεφάλαιο 5

Παρουσίαση της εφαρμογής

Τα προηγούμενα κεφάλαια αφορούσαν το πώς είναι φτιαγμένο το σύστημα, τα εργαλεία και την υλοποίηση. Σε αυτό το κεφάλαιο περιγράφεται η άλλη του πλευρά, αυτή του χρήστη, δηλαδή τι εμφανίζεται στην οθόνη με ποια σειρά και ποια απόφαση κρύβεται πίσω από κάθε στοιχείο. Η περιγραφή ακολουθεί την διαδρομή του χρήστη από την δημιουργία λογαριασμού μέχρι την υποβολή ερωτήσεων και τις απαντήσεις που έρχονται μαζί με τις παραπομπές.

Όλες οι εικόνες είναι από την έκδοση που τρέχει τοπικά, με πραγματικό όμως περιεχόμενο. Η βιβλιοθήκη περιέχει επιστημονικά άρθρα γύρω από το cloud computing και το serverless, μαζί με ένα ιατρικό άρθρο άσχετο με το θέμα, το οποίο μπήκε επίτηδες ώστε να ελεγχθεί ότι το σύστημα ξεχωρίζει τα θέματα και δεν τραβάει άσχετο υλικό όταν η ερώτηση δεν καλύπτεται από τα έγγραφα.

5.1 Είσοδος στην εφαρμογή

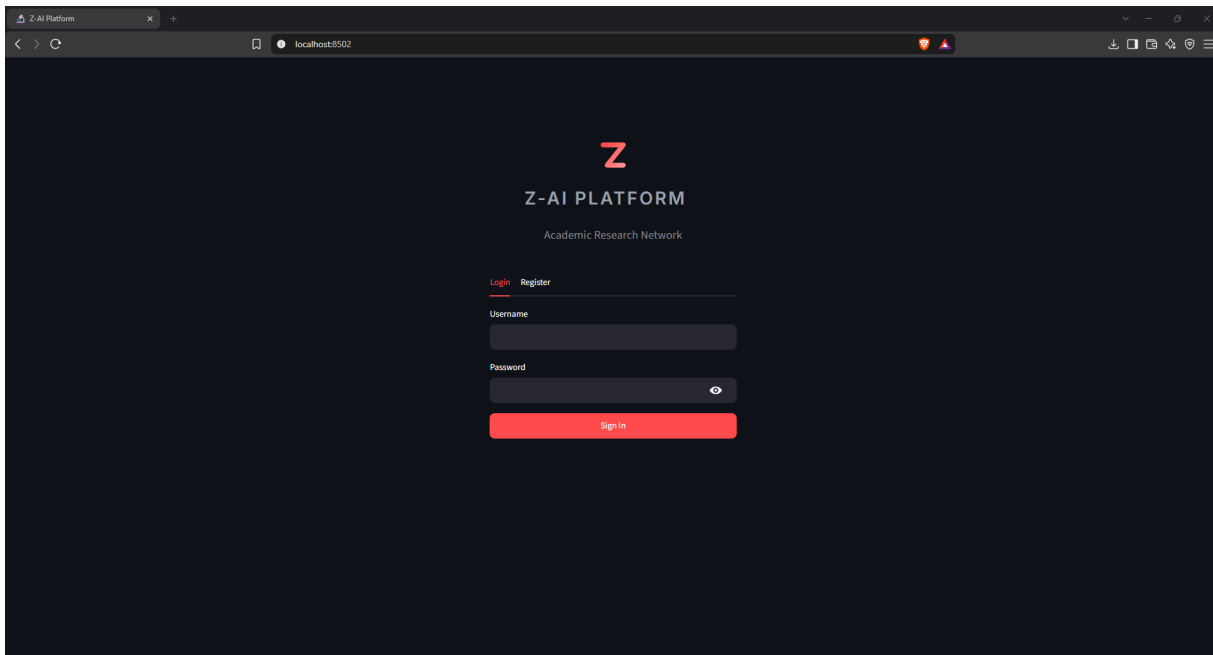
Στην πρώτη οθόνη (εικόνα 5.1), κάτω από το λογότυπο, φαίνονται δύο καρτέλες. Η πρώτη είναι για όσους έχουν ήδη λογαριασμό και συνδέονται με το όνομα χρήστη και τον κωδικό πρόσβασης. Η δεύτερη είναι η εγγραφή του χρήστη, όπου ζητείται επιπλέον και η διεύθυνση ηλεκτρονικού ταχυδρομείου.

Ο λογαριασμός απαιτείται για λόγους ασφαλείας ώστε τα αρχεία που ανεβάζει ο χρήστης στην βιβλιοθήκη να παραμένουν προσωπικά και να τα βλέπει, να τα χρησιμοποιεί και να τα διαγράφει μόνο ο ίδιος.

Αν τα στοιχεία είναι λάθος η οθόνη το λέει με μια γραμμή, ενώ αν οι αποτυχημένες προσπάθειες περάσουν το όριο μέσα σε ένα λεπτό η επόμενη δεν εκτελείται καθόλου και ζητείται από τον χρήστη να περιμένει.

5.2 Η οθόνη εργασίας και τα έγγραφα

Αφού ο χρήστης συνδεθεί επιτυχώς στην εφαρμογή, βλέπει την οθόνη της εικόνας 5.2 όπου ο χώρος είναι μοιρασμένος σε δύο μέρη. Δεξιά είναι η ροή των μηνυμάτων της συζήτησης, με το πεδίο στο οποίο γράφει την ερώτηση ο χρήστης καρφιτσωμένο στο κάτω μέρος, ενώ αριστερά βρίσκεται η στήλη διαχείρισης. Η στήλη μένει η ίδια σε όλη την χρήση ώστε ο χρήστης να μην χάνει ούτε την λίστα με τα έγγραφα που έχει ανεβάσει ούτε το ιστορικό των συζητήσεων.



Εικόνα 5.1: Η οθόνη εισόδου, με τις δύο καρτέλες για σύνδεση και εγγραφή.

Στην κορυφή της στήλης υπάρχει το κουμπί με το οποίο ξεκινάει μια νέα συζήτηση και ακριβώς από κάτω είναι ένας επιλογέας με τρία ύψη απάντησης που αλλάζει τον τρόπο με τον οποίο διατυπώνεται η απάντηση. Αυτό διευκολύνει τον χρήστη σε περίπτωση που θελήσει μια πιο σύντομη ή πιο αναλυτική απάντηση χωρίς όμως να αλλάζει το τι θα βρεθεί μέσα στα έγγραφα, που παραμένει το ίδιο.

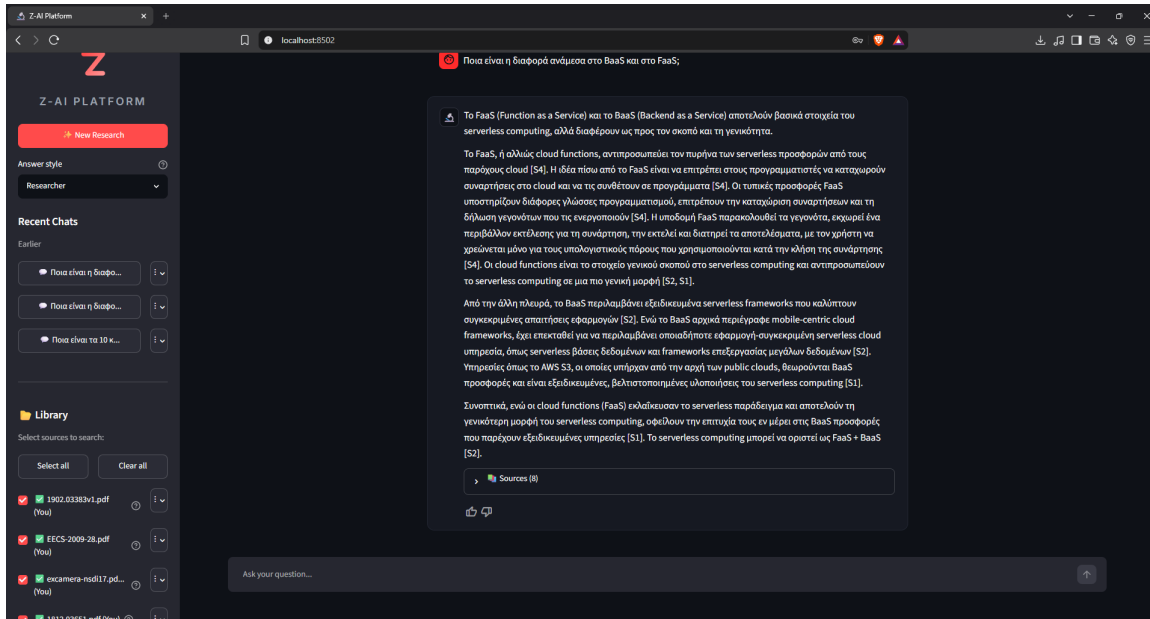
Πιο κάτω εμφανίζονται οι προηγούμενες συζητήσεις που μπορεί να έχει κάνει ο χρήστης οι οποίες είναι ομαδοποιημένες ανάλογα με το πότε δημιουργήθηκαν σε σημερινές, χθεσινές, των τελευταίων επτά ημερών και πιο παλιές, αντί για μια σκέτη ημερομηνία δίπλα σε κάθε γραμμή. Η ημερομηνία θα ήταν πιο ακριβής ένδειξη, αλλά όποιος ψάχνει μια παλιά συζήτηση συνήθως θυμάται την μέρα και όχι την ακριβή ώρα.

Στο κάτω μέρος βρίσκεται η βιβλιοθήκη δηλαδή η λίστα με τα αρχεία που έχει ανεβάσει ο χρήστης με το καθένα να έχει ένα πλαίσιο επιλογής. Τα πλαίσια αυτά δεν είναι διακοσμητικά αλλά ορίζουν σε ποια αρχεία θα ψάξει η επόμενη ερώτηση. Και αυτό γιατί όταν μια βιβλιοθήκη διαθέτει πάρα πολλά κείμενα πάνω στο ίδιο θέμα, μια ερώτηση που στοχεύει ένα συγκεκριμένο αρχείο, για παράδειγμα αυτό για το MapReduce, δεν κερδίζει τίποτα αν διαβαστούν και όλα τα υπόλοιπα. Για αυτό ο χρήστης περιορίζει από μόνος του την αναζήτηση. Τα κουμπιά Select all και Clear all υπάρχουν επειδή οι δύο συνηθέστερες κινήσεις είναι τα δύο άκρα: όλα τα αρχεία, ή ένα μόνο.

Μπροστά από το κάθε όνομα υπάρχει ένα εικονίδιο που δείχνει την κατάσταση του αρχείου, δηλαδή το πράσινο σημάδι σημαίνει ότι το έγγραφο είναι έτοιμο για να του γίνουν ερωτήσεις και η κλεψύδρα δείχνει ότι το αρχείο είναι ακόμα σε επεξεργασία. Στην δεύτερη περίπτωση το πλαίσιο επιλογής μένει κλειδωμένο ώστε να μην μπορεί να επιλεγεί κάτι το οποίο δεν είναι έτοιμο και δεν δέχεται ερωτήσεις ακόμα. Για την αποτυχία υπάρχουν δύο ενδείξεις ενώ οι αποτυχίες που μπορούν να συμβούν είναι τρεις, να μην διαβαστεί καθόλου το αρχείο, να διαβαστεί κανονικά αλλά να μην έχει κείμενο μέσα και να έχει ξανανέβει το ίδιο αρχείο με άλλο όνομα. Η τρίτη δεν έχει εικονίδιο επειδή δεν φτάνει ποτέ στην λίστα δηλαδή το αρχείο απορρίπτεται την ώρα του ανεβάσματος με μήνυμα, αφού η ταυτοποίηση γίνεται πάνω στο περιεχόμενο και όχι στο όνομα.

Η δεύτερη περίπτωση αφορά σαρωμένα PDF, δηλαδή φωτογραφίες σελίδων χωρίς να περιέχουν κείμενο από πίσω και χωρίς ξεχωριστή ένδειξη ένα τέτοιο αρχείο έδειχνε πράσινο σημάδι και δεν απαντούσε ποτέ, οπότε ο χρήστης νόμιζε ότι φταίει το σύστημα. Η λίστα ανανεώνεται κάθε τέσσερα δευτερόλεπτα ώστε να δίνει μια πλήρη αναφορά αν τα αρχεία είναι έτοιμα και να μην χρειάζεται ο χρήστης να φορτώνει και να ξαναφορτώνει την σελίδα ώστε να δει άμα τελείωσε η εξαγωγή του κειμένου.

Κάθε γραμμή διαθέτει επίσης ένα μενού το οποίο είναι οι τρεις τελείες που δίνει την επιλογή της διαγραφής και όπως είναι φυσικό το αρχείο το βλέπει και το διαγράφει μόνο ο χρήστης που το ανέβασε.



Εικόνα 5.2: Ολόκληρη η οθόνη εργασίας. Αριστερά η στήλη διαχείρισης με τα ύψη απάντησης, το ιστορικό και τη βιβλιοθήκη των εγγράφων, δεξιά η συζήτηση και το πεδίο της ερώτησης.

5.3 Η απάντηση

5.3.1 Όταν υπάρχει απάντηση

Η εικόνα 5.3 δείχνει τη στήλη της συζήτησης από πιο κοντά με δύο ερωτήσεις οι οποίες γίνονται η μία μετά την άλλη. Η πρώτη παίρνει απάντηση ενώ η δεύτερη όχι, και η διαφορά τους είναι το θέμα αυτής της ενότητας.

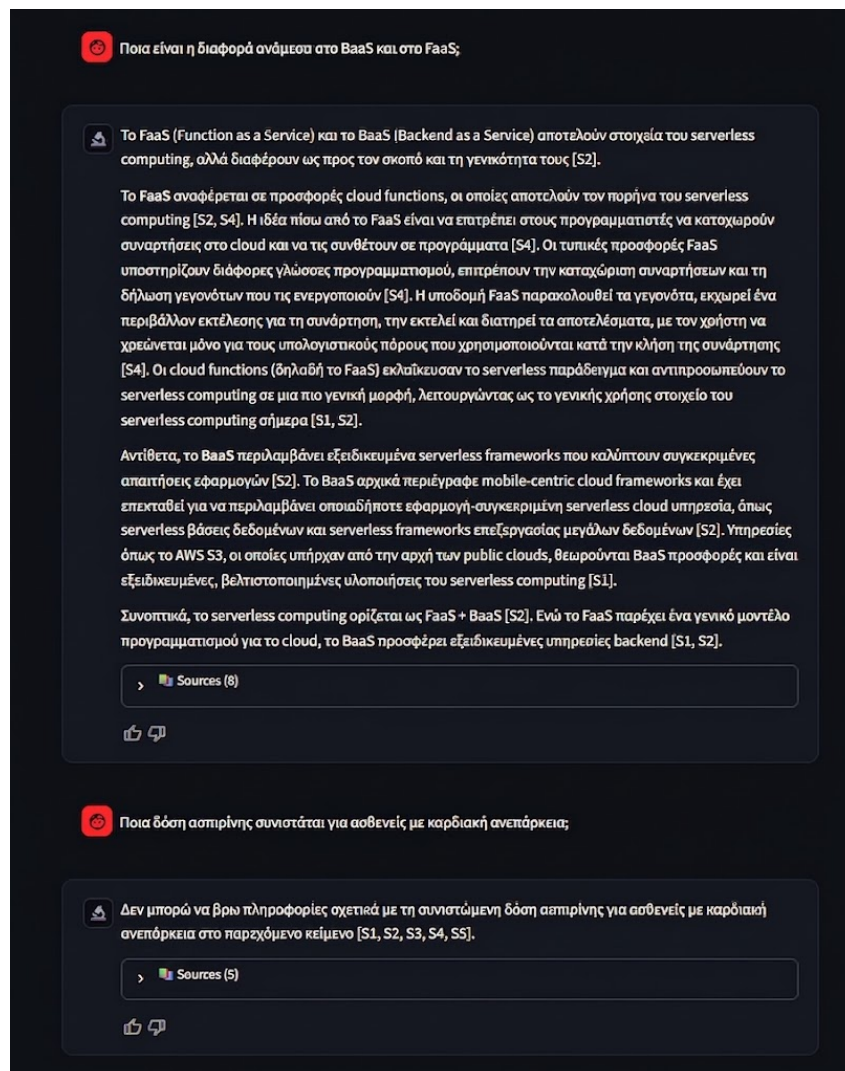
Στην πρώτη περίπτωση η ερώτηση η οποία είναι στα ελληνικά απαντιέται επίσης στα ελληνικά, ενώ όλα τα έγγραφα της βιβλιοθήκης είναι αγγλικά δηλαδή η απάντηση βγαίνει στην γλώσσα της ερώτησης. Επίσης η απάντηση στηρίζεται στα κείμενα που έχουν επιλεχτεί και όχι στην γενική γνώση του μοντέλου.

Μέσα στην απάντηση φαίνονται μικρές ετικέτες της μορφής [S1] και [S2] στο τέλος των προτάσεων που η καθεμία δείχνει σε ένα συγκεκριμένο απόσπασμα από αυτά που δόθηκαν, και μπαίνει δίπλα στην πρόταση που το χρησιμοποιεί και όχι όλες μαζεμένες στο τέλος. Έτσι ο αναγνώστης δεν ελέγχει όλη την απάντηση μονομιάς αλλά πρόταση πρόταση.

5.3.2 Όταν δεν υπάρχει απάντηση

Η δεύτερη ερώτηση της ίδιας εικόνας ζητάει την συνιστώμενη δόση της ασπιρίνης για ασθενείς με καρδιακή ανεπάρκεια. Το ιατρικό άρθρο μιλάει όντως για την καρδιακή ανεπάρκεια, η λέξη ασπιρίνη όμως δεν εμφανίζεται ούτε μία φορά μέσα σε αυτό, οπότε η δοσολογία που ζητείται δεν υπάρχει πουθενά στα κείμενα της βιβλιοθήκης.

Η απόκριση που παρατηρείται δεν είναι απάντηση αλλά δήλωση ότι η πληροφορία δεν βρέθηκε στο κείμενο που δόθηκε στο μοντέλο. Το πλαίσιο των πηγών από κάτω δείχνει ότι βρέθηκαν αποσπάσματα, όλα όμως μιλούσαν για την καρδιακή ανεπάρκεια χωρίς να αναφέρουν δοσολογία. Και αυτό γιατί το γλωσσικό μοντέλο μπορεί να δώσει μια πειστική απάντηση στο ερώτημα ακόμα και χωρίς να υπάρχει κάτι στο κείμενο. Σε ένα σύστημα όμως που προορίζεται για μελέτη αρχείων, μια τέτοια παράγραφος που δεν στηρίζεται στο κείμενο είναι χειρότερη από το να μην απαντήσει καθόλου. Για αυτό το σύστημα είναι φτιαγμένο ώστε σε αυτή την περίπτωση να το δηλώνει αντί να βγάλει έναν αριθμό από μόνο του.



Εικόνα 5.3: Δύο ερωτήσεις στην ίδια συζήτηση. Επάνω, απάντηση στα ελληνικά με βάση αγγλικά έγγραφα, με τις παραπομπές μέσα στο κείμενο. Κάτω, ερώτηση που δεν καλύπτεται από τα έγγραφα: το σύστημα δηλώνει ότι δεν βρήκε την πληροφορία αντί να συνθέσει κείμενο χωρίς στήριγμα.

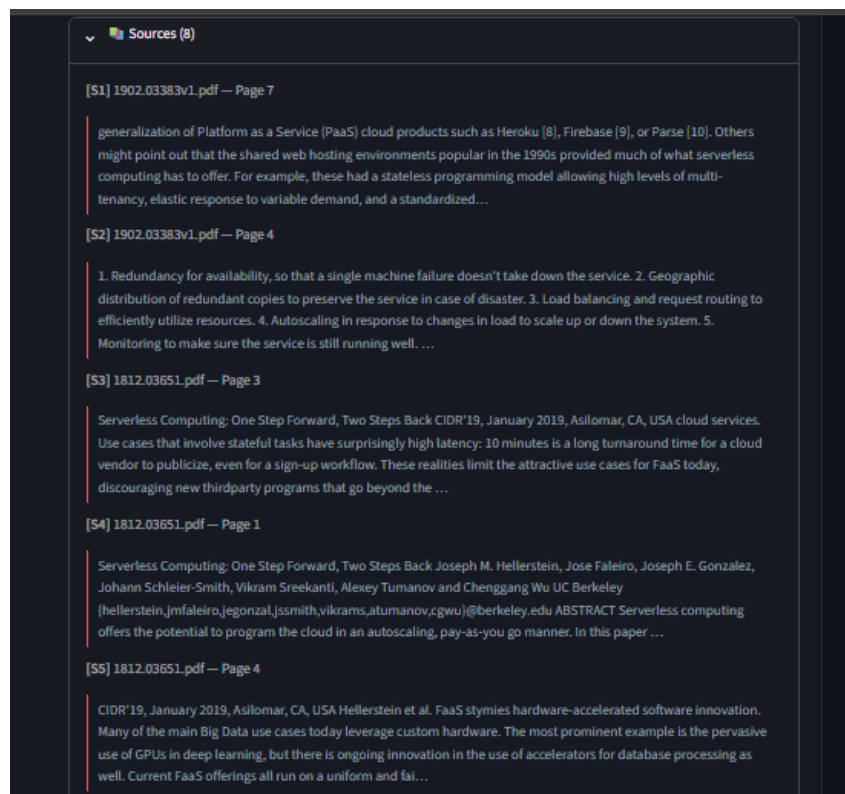
5.4 Οι πηγές και η αξιολόγηση

Κάτω από κάθε απάντηση υπάρχει ένα κλειστό πλαίσιο με τον αριθμό των πηγών που βρέθηκαν, και όταν ανοίξει εμφανίζεται η λίστα στην εικόνα 5.4. Κάθε γραμμή έχει την ίδια ετικέτα που εμφανίζεται και μέσα στο κείμενο, μαζί με το όνομα του αρχείου, τον αριθμό σελίδας και ένα μικρό απόσπασμα του πρωτοτύπου.

Το πλαίσιο εμφανίζεται κλειστό επειδή οι δύο χρήσεις είναι διαφορετικές. Ο χρήστης θέλει πρώτα να διαβάσει την απάντηση και μόνο αν αμφιβάλλει για κάτι ανοίγει τις πηγές ώστε να το διασταυρώσει. Αν οι πηγές ήταν πάντα ανοιχτές θα έπιαναν πάρα πολύ χώρο και η απάντηση θα ήταν πολύ πιο μικρή από τις ίδιες τις πηγές, κάνοντας το ιστορικό συζήτησης να είναι τεράστιο.

Το απόσπασμα το οποίο δίνεται στις πηγές υπάρχει ώστε ο χρήστης να μην χρειάζεται να ανοίξει το PDF για να ελέγξει την πηγή δηλαδή ο χρήστης διαβάζει τι έλεγε στην πραγματικότητα το πρωτότυπο και κρίνει εκεί αν η πρόταση της απάντησης αποδίδει σωστά το περιεχόμενο της πηγής. Φαίνεται και κάτι μη αυτονόητο ότι οι παραπομπές δείχνουν σε διαφορετικές σελίδες και μάλιστα περισσότερες από μία στο ίδιο αρχείο. Η αναζήτηση δηλαδή δεν επιστρέφει απλά ένα έγγραφο αλλά συγκεκριμένα αποσπάσματα που μπορεί να προέρχονται από διαφορετικά σημεία του ίδιου κειμένου.

Στο τέλος κάθε απάντησης κάτω από το πλαίσιο των πηγών, υπάρχουν δύο κουμπιά με τα οποία ο χρήστης μπορεί να αξιολογήσει την απάντηση θετικά ή αρνητικά και πατώντας το αρνητικό κουμπί ανοίγει ένα πεδίο όπου μπορεί προαιρετικά να γράψει ο χρήστης τι δεν πήγε καλά, τι δεν του άρεσε στην απάντηση. Στο θετικό δεν ζητείται τίποτα γιατί όταν η απάντηση είναι σωστή ο χρήστης δεν έχει κάποιο λόγο να σταματήσει για να γράψει κάποιο σχόλιο.



Εικόνα 5.4: Το πλαίσιο των πηγών ανοιχτό. Κάθε παραπομπή δίνει αρχείο, σελίδα και απόσπασμα του πρωτοτύπου. Εικονίζονται οι πέντε πρώτες από τις οκτώ.

Κεφάλαιο 6

Αξιολόγηση

Ένα σύστημα RAG μπορεί να δείχνει ότι δουλεύει ενώ στην πραγματικότητα μπορεί να μην δουλεύει, δηλαδή δίνει απαντήσεις οι οποίες διαβάζονται και μπορεί να φαίνονται σωστές, με παραπομπές που μοιάζουν να στηρίζονται στα σωστά κείμενα. Το πρόβλημα φαίνεται όταν κάνει ο χρήστης τον κόπο να ελέγξει αν το κείμενο στην απάντηση είναι αυτό που ισχυρίζεται.

6.1 Τι μετρήθηκε και με τι

Τα σώματα των κειμένων είναι δύο και το καθένα κάνει διαφορετική δουλειά. Το πρώτο είναι επτά επιστημονικά άρθρα γύρω από το cloud και το serverless computing, από τα οποία μόνο στα δύο γράφτηκαν ερωτήσεις. Τα άλλα πέντε δουλεύουν σαν distractors δηλαδή κείμενα του ίδιου πεδίου με το ίδιο λεξιλόγιο για τα οποία όμως δεν γράφηκε καμία ερώτηση. Ο λόγος που υπάρχουν είναι να δυσκολέψουν το σύστημα ώστε να ξεχωρίσει μόνο τα σωστά κείμενα για την απάντησή τους ώστε να γίνει πιο κατανοητό αν όλο το σύστημα δουλεύει σωστά ακόμα και σε δύσκολες καταστάσεις. Στην ρύθμιση παραγωγής το σώμα αυτό κόβεται σε 418 τμήματα.

Το δεύτερο σώμα δεν έχει καμία σχέση με το πρώτο, που γράφτηκε πιο πριν, και αυτός ακριβώς είναι ο λόγος που υπάρχει και ελέγχει κάτι διαφορετικό. Στο πρώτο σετ ελέγχθηκε πώς αντιδράει το σύστημα όταν όλα τα άρθρα μιλάνε για παρόμοια πράγματα, δηλαδή αν βρίσκει τη σωστή επιλογή ενώ γύρω του υπάρχουν τόσα παρεμφερή κείμενα. Το δεύτερο σώμα αποτελείται από δύο άρθρα, ένα ιατρικό το οποίο μιλάει για την καρδιακή ανεπάρκεια και ένα ηφαιστειολογικό για τις εκρήξεις του Στρόμπολι που περιείχε είκοσι πέντε ερωτήσεις οι οποίες ήταν όλες στα ελληνικά. Το σύστημα φτιάχνει μια περιγραφή του πεδίου διαβάζοντας τα ίδια τα κείμενα, ώστε να μεταφράζει σωστά τους όρους και αυτό γίνεται γιατί η μετάφραση από ένα θέμα σε ένα άλλο είναι διαφορετική και όπως παρατηρήθηκε χωρίς αυτό δυσκολευόταν η μετάφραση ανάμεσα σε διαφορετικά άρθρα. Οπότε για αυτό δημιουργήθηκε το δεύτερο σώμα ώστε να ελέγξει και να αποδείξει ότι γίνεται σωστή μετάφραση για να γίνει η απάντηση.

Οι πέντε ερωτήσεις του δεύτερου σώματος που δεν απαντώνται είναι σχεδιασμένες έτσι ώστε οι δύο να είναι εκτός θέματος και οι άλλες τρεις γειτονικές, στο ίδιο πεδίο αλλά εκτός των κειμένων, για παράδειγμα τότε εξερράγη τελευταία φορά ο Βεζούβιος, κάτι που δεν υπάρχει στα αντίστοιχα κείμενα. Στην ενότητα 6.3.1 φαίνεται τι έγινε ακριβώς όταν το έκανε. Το κύριο σετ του πίνακα 6.1 έχει πενήντα ερωτήσεις από τις οποίες οι σαράντα πέντε απαντώνται από τα κείμενα και οι πέντε όχι οι οποίες υπάρχουν ώστε να ελεγχθεί αν το σύστημα απαντάει «δεν ξέρω» όταν η ερώτηση δεν καλύπτεται από τα άρθρα. Οι υπόλοιπες

Πίνακας 6.1: Τα σετ αξιολόγησης που χρησιμοποιούνται στο κεφάλαιο αυτό.

Σετ	Τι πιέζει	Ερωτ.	EL	EN
<i>Πρώτο σώμα – cloud και serverless, 7 papers</i>				
golden_set_50	κύριο σετ αναφοράς	50	20	30
golden_hard_paraphrase	παραφράσεις χωρίς κοινές λέξεις	16	5	11
golden_multihop_new	σύνθεση από δύο πηγές	11	2	9
golden_tables	αριθμοί μέσα σε πίνακες	16	0	16
golden_conversations	συνέχεια συνομιλίας	12	4	8
<i>Δεύτερο σώμα – ιατρική και ηφαιστειολογία, 2 papers</i>				
golden_test_domains	άλλο γνωστικό πεδίο	25	25	0

χωρίζονται σε τέσσερις κατηγορίες ερωτήσεων ανάλογα με το τι ζητάει η ερώτηση, από απλή αναζήτηση κάποιου γεγονότος μέχρι συνδυασμό από διαφορετικά άρθρα.

Οι μετρήσεις που έγιναν χωρίζονται σε τρία επίπεδα. Στο πρώτο μετρήθηκε αν η ανάκτηση φέρνει τα σωστά κομμάτια κειμένου, με MRR [17] για την σειρά με την οποία έρχονται και με κάλυψη λέξεων κλειδιών για το αν οι λέξεις που χρειάζεται η απάντηση βρίσκονται μέσα στις σελίδες που ανακτήθηκαν. Στο δεύτερο είναι ένας κριτής LLM ο οποίος βαθμολογεί το κομμάτι από την στιγμή που φτάνει στο γλωσσικό μοντέλο για να δημιουργήσει την απάντηση μέχρι την τελική απάντηση και την βαθμολογεί σε ακρίβεια, πληρότητα, συνάφεια και πιστότητα. Το τρίτο και τελευταίο είναι το RAGAS το οποίο ξαναμετράει την ποιότητα με εντελώς άλλη μεθοδολογία γιατί είναι ανεξάρτητος ελεγκτής.

Το MRR δεν προβλέπει την ποιότητα της απάντησης αλλά το πόσο ψηλά βγήκε το σωστό κομμάτι όμως μια ερώτηση μπορεί να έχει μέτριο MRR και να είναι τέλεια απάντηση γιατί μπορεί η απάντηση να βρίσκεται και στα υπόλοιπα όχι μόνο στο πρώτο για αυτό όπου κρίνεται ποιότητα κρίνεται με το coverage και τον κριτή και όχι με την κατάταξη MRR.

Ο περιορισμός που ενοχλεί περισσότερο αφορά την επαναληψιμότητα και αυτό γιατί μια ελληνική ερώτηση μεταφράζεται πρώτα στα αγγλικά από το γλωσσικό μοντέλο και ψάχνει η μετάφρασή της το μοντέλο όμως δεν γράφει δύο φορές την ίδια πρόταση. Το μέγεθος του φαινομένου φαίνεται επειδή οι αγγλικές ερωτήσεις δεν περνάνε από το βήμα αυτό σε δεκαπέντε τρεξίματα έδωσαν MRR 0,7866 κάθε φορά, ενώ το ελληνικό υποσύνολο στα ίδια τρεξίματα κυμάνθηκε από 0,802 έως 0,846 — και 0,846, 0,832, 0,811 σε τρεις επαναλήψεις της ίδιας ακριβώς ρύθμισης. Διαφορά μικρότερη από 0,035 εκεί μέσα δεν σχολιάζεται ως εύρημα.

Τέλος ο κριτής LLM [18] είναι το ίδιο το γλωσσικό μοντέλο το οποίο έγραψε και τις απαντήσεις δίνοντας πιο ευνοϊκούς βαθμούς στα δικά τους κείμενα, για αυτό ακριβώς τον λόγο μπήκε το RAGAS [15], που αποσυνθέτει την απάντηση σε επιμέρους ισχυρισμούς και ελέγχει το καθένα ξεχωριστά. Και έτσι αν διαπιστωθεί ότι αυτά τα δύο διαφορετικά μοντέλα συμφωνούν το αποτέλεσμα δείχνει αξιοπιστία και όχι μεροληψία.

Όσα ακολουθούν προέρχονται από 81 σενάρια που παρήγαγαν 135 αρχεία αποτελεσμάτων σε διαφορετικές χρονικές στιγμές, με τα περισσότερα να είναι συγκρίσεις δύο ρυθμίσεων. Όλα τα σκέλη που συγκρίνονται μετρήθηκαν με τον ίδιο κώδικα οπότε ισχύουν οι αλλαγές που έγιναν και οι απορρίψεις και όχι απλώς μια τρέχουσα επίδοση του συστήματος. Το ένα απόλυτο νούμερο του συστήματος σήμερα είναι η κάλυψη στο κύριο σετ που κυμαίνεται από 97,78% έως 98,52% ανάλογα με το τρέξιμο.

6.2 Ανάκτηση

Δύο νούμερα περιγράφουν την ανάκτηση. Το MRR δείχνει πόσο ψηλά βρίσκεται το πρώτο σωστό απόσπασμα, και η κάλυψη δείχνει αν το υλικό έφτασε καν στο μοντέλο. Το δεύτερο είναι το κρίσιμο για την απάντηση, το πρώτο για τη σύγκριση ρυθμίσεων: η κάλυψη πιάνει ταβάνι νωρίς και σταματάει να ξεχωρίζει.

6.2.1 Τι προσθέτει κάθε στάδιο

Ο πίνακας 6.2 χτίζει το pipeline σκαλοπάτι σκαλοπάτι, στις ίδιες ερωτήσεις. Κάθε στάδιο τερματίζει στο ίδιο σημείο, ώστε να συγκρίνεται αυτό που φτάνει στο μοντέλο και όχι πόσα αποσπάσματα γύρισε το καθένα.

Πίνακας 6.2: Θετική σκάλα: τι προσθέτει κάθε στάδιο (45 ερωτήσεις εντός corpus, 5 εκτός).

Στάδιο	MRR	Αγγλικά	Ελληνικά	Εκτός corpus
Μόνο dense	0,731	0,705	0,771	0/5
+ BM25 / RRF	0,774	0,764	0,788	0/5
+ cross-encoder	0,793	0,787	0,802	0/5
+ relevance gate	0,793	0,787	0,802	5/5

Τα πρώτα δύο σκαλοπάτια δίνουν μικρή αύξηση, +0,043 και +0,019, που όμως δεν είναι καθαρό κέρδος. Περνώντας από τη διανυσματική στην υβριδική αναζήτηση δεκαπέντε ερωτήσεις βελτιώθηκαν και δέκα χειροτέρεψαν, ενώ με τον cross-encoder δεκατρείς βελτιώθηκαν και έξι χειροτέρεψαν. Ο μέσος όρος ανεβαίνει επειδή τα κέρδη είναι μεγαλύτερα από τις απώλειες.

Το τελευταίο σκαλοπάτι δεν αλλάζει το MRR καθόλου, δηλαδή η τιμή είναι ταυτόσημη και στις σαράντα πέντε ερωτήσεις, γιατί το gate δεν αναδιατάσσει τίποτα, μόνο αρνείται. Όλη του η δουλειά φαίνεται στην τελευταία στήλη όπου το σύστημα περνάει από το να απαντάει πάντα, στο να μην απαντάει και στις πέντε ερωτήσεις για τις οποίες δεν υπάρχει υλικό. Είναι το μόνο στάδιο που αλλάζει είδος συμπεριφοράς αντί για βαθμό και ταυτόχρονα το μόνο που μια μετρική κατάταξης δεν μπορεί να επιβραβεύσει.

6.2.2 Το μέγεθος του τμήματος

Η επιλογή των 1500 χαρακτήρων δεν προέκυψε από το κύριο σετ ερωτήσεων. Όπως δείχνει ο πίνακας 6.3 τα 750 βγάζουν λίγο καλύτερο MRR, αλλά και στις δύο ρυθμίσεις η κάλυψη είναι 98,52%, δηλαδή στο ταβάνι, οπότε το σετ αυτό δεν μπορεί να δείξει τι χάνεται.

Πίνακας 6.3: Μέγεθος τμήματος στα δύο σετ ερωτήσεων. Το πρόσημο αντιστρέφεται.

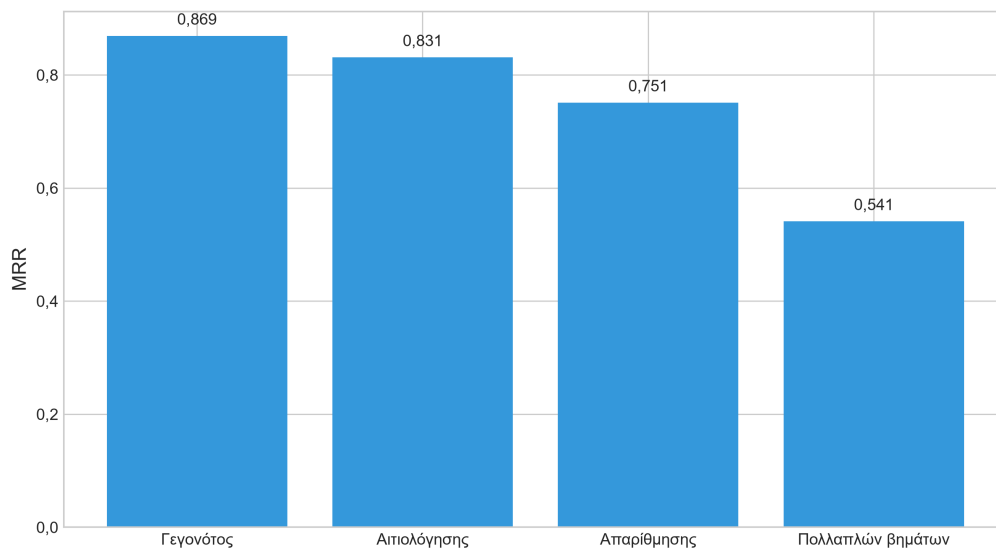
Μέγεθος τμήματος	Τμήματα	MRR κύριο σετ	MRR δύσκολο σετ
750 χαρακτήρες	808	0,834	0,189
1000 χαρακτήρες	622	0,837	0,254
1500 χαρακτήρες	418	0,793	0,298

Για αυτό δημιουργήθηκε ένα δύσκολο σετ δεκαέξι ερωτήσεων γραμμένες με καθημερινό λεξιλόγιο που αποφεύγουν να χρησιμοποιήσουν τους όρους που έχουν τα κείμενα εκεί

το πρόσημο αντιστρέφεται και η διαφορά ανάμεσα σε αυτά τα δύο είναι μεγαλύτερη. Πιο καθαρά φαίνεται στις ερωτήσεις πολλαπλών βημάτων, όπου τα 750 δίνουν 0,079 ενώ τα 1500 δίνουν 0,204 γιατί όταν η απάντηση απαιτεί δύο σημεία του κειμένου και όχι ένα τα μικρά τμήματα τα σπάνε και το ένα από τα δύο χάνεται οπότε κρατήθηκε το 1500 με κόστος 0,04 σε ένα σετ το οποίο δεν μπορεί να δει την διαφορά.

Προς τα πάνω ο δρόμος είναι κλειστός και ο λόγος είναι ότι ο cross-encoder δέχεται 512 tokens και τα υπόλοιπα τα κόβει. Το όριο αυτό δεν αντιστοιχεί σε σταθερό αριθμό χαρακτήρων, γιατί κείμενα με πίνακες ή τύπους ξεοδεύουν πολύ περισσότερα tokens ανά χαρακτήρα. Για αυτό ακόμη και στα 1500 περικόπτονται 15 από τα 418 τμήματα ενώ στα 2000 θα έφτανε το ένα τέταρτο αλλά η περικοπή αφορά μόνο την κατάταξη αφού το τμήμα μπαίνει ολόκληρο στο ερώτημα προς το μοντέλο.

6.2.3 Πού αποτυγχάνει



Εικόνα 6.1: MRR ανά κατηγορία ερώτησης, στο πλήρες pipeline.

Η Εικόνα 6.1 δείχνει πού σπάει η ανάκτηση, δηλαδή οι ερωτήσεις του γεγονότος φτάνουν 0,869 και οι αιτιολόγησης 0,831 αλλά παρατηρείται ότι οι απαριθμήσεις πέφτουν στο 0,751 και αυτό γιατί η λίστα ξεπερνάει τα όρια του τμήματος οπότε το πρώτο σωστό απόσπασμα περιέχει μέρος μόνο της απάντησης. Και τέλος είναι οι ερωτήσεις πολλαπλών βημάτων [19] οι οποίες μένουν στο 0,541 με την διευκρίνιση ότι είναι μόλις τέσσερις στο κύριο σετ και το νούμερο δεν αντέχει βάρος.

6.3 Το ερώτημα πριν την αναζήτηση

Μια ελληνική ερώτηση δεν φτάνει ποτέ αυτούσια στην βάση και ο λόγος είναι ότι μεταφράζεται στα αγγλικά και μαζί είναι μια περιγραφή του πεδίου και ένα λεξιλόγιο του corpus ώστε η μετάφραση να διαλέγει όρους βάσει των κειμένων. Δοκιμάστηκαν έντεκα παραλλαγές αυτής της πληροφορίας ώστε να φτάσει σε αυτό το σημείο από πλήρη απουσία έως πλούσια περιγραφή.

Στο κύριο σετ ερωτήσεων καμία δεν ξεχώρισε όλες κυμάνθηκαν από 0,802 έως 0,846 όλα μέσα στο περιθώριο θορύβου και ο λόγος είναι ότι οι ερωτήσεις αυτές γράφτηκαν διαβάζοντας τα ίδια τα κείμενα, οπότε χρησιμοποιούν ήδη από μόνες τους το λεξιλόγιο δεν υπάρχει κενό να καλυφθεί.

Το κενό φάνηκε στην δεύτερη συλλογή όπου υπάρχουν μόνο ελληνικές ερωτήσεις που στηρίζονταν σε ιατρική και ηφαιστειολογία και χωρίς λεξιλόγιο η κάλυψη όπως φαίνεται είναι στο 85% ενώ με λεξιλόγιο βρίσκεται στο 95% οπότε παρατηρείται ότι η μετάφραση χρειάζεται βοήθεια.

6.3.1 Το κόστος: διαρροή στις ερωτήσεις χωρίς υλικό

Η βοήθεια όπως δείχνει ο πίνακας 6.4 έχει τίμημα. Κάθε διαρροή οφείλεται στην ίδια

Πίνακας 6.4: Τρεις διατυπώσεις της ίδιας πληροφορίας (20 ερωτήσεις εντός corpus, 5 εκτός).

Οδηγία προς τον μεταφραστή	Κάλυψη	Διαρροές
Χωρίς λεξιλόγιο	85%	0/5
«Προτίμησε το λεξιλόγιο του corpus»	95%	1/5
Ίδιες λέξεις, χωρίς προστακτική	95%	1/5
Περιοριστική οδηγία	95%	0/5

ερώτηση «πότε εξερράγη για τελευταία φορά ο Βεζούβιος» που το corpus απαντά για το Στρόμπολι και δεν έχει καμία πληροφορία για τον Βεζούβιο οπότε η σωστή συμπεριφορά είναι να πει ότι δεν γνωρίζει. Το λεξιλόγιο όμως έδωσε τον όρο eruption rate που ανήκει στο κείμενο και η ερώτηση έπαψε να ρωτάει πότε. Το ξαναγραμμένο ερώτημα πέρασε το κατώφλι, οριακά έστω, και το σύστημα απάντησε στην ερώτηση.

Το μόνο που βοήθησε όντως και κρατήθηκε στο σύστημα είναι η περιοριστική εκδοχή η οποία επιτρέπει όρο του λεξιλογίου μόνο ως μετάφραση λέξης που υπάρχει ήδη στην ερώτηση και απαγορεύει την αντικατάσταση κυρίων ονομάτων και λόγω αυτού η κάλυψη ανέβηκε στο 95% χωρίς καμία διαρροή.

Τέλος το baseline της ανάκτησης είναι προγενέστερο της αυτόματης περιγραφής του πεδίου, οπότε ξαναμετρήθηκε με το χαρακτηριστικό κλειστό έτσι η κάλυψή του βγήκε 98,52% ίδια με πριν.

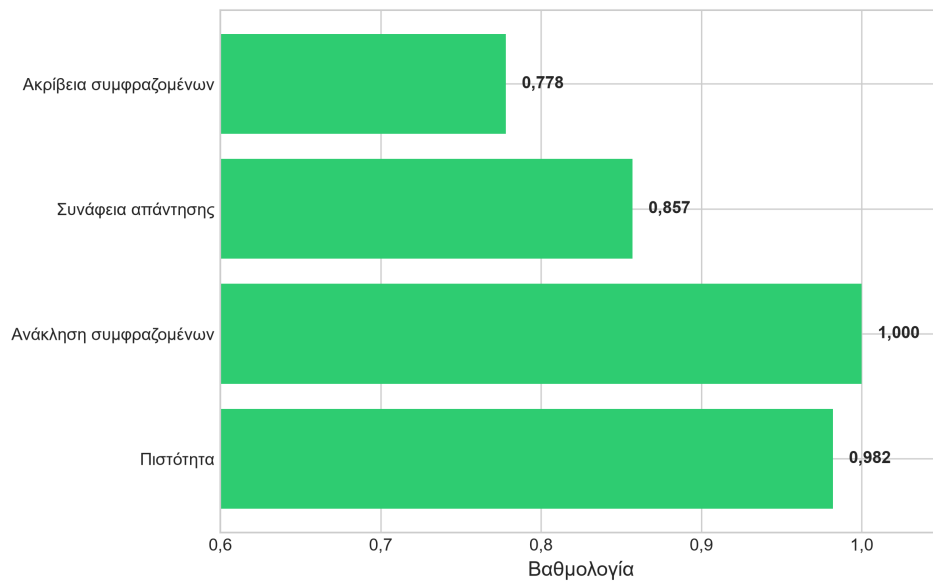
6.4 Ποιότητα των απαντήσεων

Η ανάκτηση κρίνεται στο κομμάτι της κατάταξης, η απάντηση όμως όχι, και για αυτό χρησιμοποιήθηκαν δύο κριτές. Ο πρώτος έχει γνωστή μεροληψία, αφού είναι το ίδιο γλωσσικό μοντέλο που παρήγαγε τις απαντήσεις και τείνει να ευνοεί τα δικά του κείμενα. Ο δεύτερος, το RAGAS, δεν λύνει το πρόβλημα του μοντέλου γιατί και αυτό τρέχει σε Gemini, αλλάζει όμως τη μέθοδο ελέγχου: αντί για ολιστική βαθμολογία σπάει την απάντηση σε ισχυρισμούς και ελέγχει τον καθένα ξεχωριστά απέναντι στο ανακτημένο κείμενο.

6.4.1 Ο πρώτος κριτής

Ο πρώτος δίνει 41 άριστες απαντήσεις από τις 45 και ο μέσος όρος είναι στο 4,93 στα τέσσερα κριτήρια και γενικότερα χρησιμοποιήθηκε ως έλεγχος. Το ενδιαφέρον σε αυτήν

την μέτρηση είναι οι τέσσερις εξαιρέσεις οι οποίες δεν ήταν άριστες από τις οποίες οι δύο είναι ερωτήσεις πολλαπλών βημάτων και οι μοναδικές οι οποίες έχουν πιστότητα 3 δηλαδή η αδυναμία που η ενότητα 6.2.3 μέτρησε ως 0,541 και φτάνει μέχρι το τελικό κείμενο. Η



Εικόνα 6.2: Μετρικές RAGAS στις 45 ερωτήσεις εντός corpus.

εικόνα 6.2 δείχνει τι είδε ο δεύτερος κριτής που το κάθε κριτήριο σημαίνει κάτι διαφορετικό, δηλαδή η πιστότητα που είναι 0,982 σημαίνει ότι σχεδόν κάθε ισχυρισμός που υπάρχει στην απάντηση στηρίζεται στο κείμενο που ανακτάται και η ανάκληση συμφραζομένων 1,000 ότι το σωστό υλικό έφτασε πάντα. Η ακρίβεια συμφραζομένων που είναι 0,778 θέλει μια διευκρίνιση γιατί στο μοντέλο δεν στέλνονται μεμονωμένα τμήματα αλλά ολόκληρες σελίδες, οπότε όταν η απάντηση βρίσκεται σε μια παράγραφο μιας σελίδας η υπόλοιπη σελίδα την οποία παίρνει είναι άσχετη και το επιπλέον αυτό κείμενο δεν βρίσκεται στην απάντηση και αυτό φαίνεται από την πιστότητα που αναφέρθηκε προηγουμένως αλλά και την ακρίβεια του πρώτου κριτή.

Μένει όμως το ότι ο κριτής είναι από την ίδια οικογένεια με το μοντέλο που γράφει τις απαντήσεις. Για αυτό το ίδιο dataset ξαναβαθμολογήθηκε με GPT-4o-mini και embeddings `text-embedding-3-small`: ίδια βιβλιοθήκη (RAGAS 0.4.3), ίδιες ερωτήσεις, ίδια ανακτημένα κείμενα, καμία αλλαγή στο σύστημα. Η πιστότητα του πίνακα 6.5, που ελέγχει

Πίνακας 6.5: Οι ίδιες 45 ερωτήσεις, όπως τις βαθμολόγησαν δύο διαφορετικές οικογένειες μοντέλων. Αλλάζει μόνο ο κριτής· το dataset είναι το ίδιο.

μετρική	Gemini 2.5 Flash	GPT-4o-mini	διαφορά
πιστότητα (faithfulness)	0,982	0,978	−0,004
ανάκληση συμφραζομένων (context recall)	1,000	1,000	0,000
συνάφεια απάντησης (answer relevancy)	0,857	0,804	−0,054
ακρίβεια συμφραζομένων (context precision)	0,778	0,927	+0,149

κείμενο γραμμένο από το Gemini και άρα είναι η πιο εκτεθειμένη σε αυτοπροτίμηση, δεν κουνήθηκε: 0,982 έναντι 0,978. Η ανάκληση συμφραζομένων βγήκε 1,000 και στους δύο κριτές, και μάλιστα στις 45 ερωτήσεις χωριστά, ό,τι χρειάζεται η σωστή απάντηση υπάρχει μέσα σε αυτά που ανακτήθηκαν, όποιος κι αν κρίνει.

Οι άλλες δύο μετρήσεις κουνήθηκαν, για διαφορετικό λόγο η καθεμία. Η συνάφεια απάντησης δεν συγκρίνεται καθαρά, γιατί μετράει ομοιότητα διανυσμάτων και μαζί με τον κριτή άλλαξε και ο χώρος των embeddings. Η ακρίβεια συμφραζομένων ανέβηκε κατά 0,149, και η φορά έχει σημασία: ο εξωτερικός κριτής είναι ο πιο γενναϊόδωρος, όχι ο αυστηρότερος. Το Gemini έβαλε κάτω από 0,5 σε επτά ερωτήσεις, το GPT-4o-mini σε καμία. Η διαφορά μαζεύεται ακριβώς εκεί που γεννιέται ο βαθμός, σε σελίδες που έχουν την απάντηση μαζί με το γύρω κείμενο, όπου το πού τραβάς τη γραμμή είναι κρίση και όχι γεγονός.

Το ίδιο το RAGAS πάντως έχει θόρυβο. Όταν το σύνολο ξαναμετρήθηκε αργότερα, μια αγγλική ερώτηση με ανακτιμένα αποσπάσματα ταυτόσημα μέχρι τον χαρακτήρα και ταυτόσημη απάντηση πήρε ακρίβεια συμφραζομένων 0,000 τη μια φορά και 1,000 την άλλη. Το ίδιο φαίνεται και ανάμεσα στους δύο κριτές: κόβουν βαθμό στην πιστότητα σε οκτώ απαντήσεις ο καθένας, αλλά μόνο οι δύο είναι κοινές. Ο μέσος όρος των 45 ερωτήσεων είναι σταθερός, η βαθμολογία μιας μεμονωμένης ερώτησης δεν είναι. Τα νούμερα δίνουν επίπεδο, όχι μέτρο σύγκρισης εκδόσεων.

Οι δύο κριτές, με διαφορετικό τρόπο ελέγχου ο καθένας, καταλήγουν στην ίδια εικόνα. Η επιφύλαξη αφορά μόνο το σκέλος της βαθμολόγησης: η ανάκτηση κρίθηκε απέναντι σε σύνολο ερωτήσεων με ανθρώπινη απάντηση-κλειδί και οι αρνήσεις εκτός corpus είχαν αντικειμενικά ελέγξιμη έκβαση, χωρίς καμία κρίση μοντέλου.

6.5 Επιδόσεις

Όλες οι μετρήσεις έγιναν τοπικά, σε AMD Ryzen 7 5700X χωρίς κάρτα γραφικών με το backend να είναι μέσα στο container του.

Σε ζεστό σύστημα και έναν χρήστη η ανάκτηση παίρνει 846 ms, από τα οποία τα 784 είναι το κομμάτι της αναδιάταξης ενώ όλα τα υπόλοιπα στάδια μαζί δεν φτάνουν τα 62 ms. Η ολοκληρωμένη όμως απάντηση θέλει 2,8 έως 4,3 δευτερόλεπτα άρα η ανάκτηση είναι το ένα πέμπτο έως το ένα τρίτο του χρόνου και τα υπόλοιπα είναι η γέννηση κειμένου. Η μετάβαση από τον L-6 στον L-12 κόστισε 275 ms στον συνολικό χρόνο. Ο κορεσμός όπως

Πίνακας 6.6: Ανάκτηση υπό ταυτόχρονο φορτίο. Μετρίεται μόνο το σκέλος που δεσμεύει CPU, χωρίς τη γέννηση.

χρήστες	p50	p95	throughput	p50 προς 1 χρήστη
1	0,86 s	0,88 s	1,18 req/s	1,00×
4	2,82 s	3,16 s	1,38 req/s	3,28×
8	8,66 s	9,39 s	0,90 req/s	10,07×
16	13,84 s	16,94 s	0,98 req/s	16,09×
32	28,08 s	32,04 s	1,00 req/s	32,65×

φαίνεται στον πίνακα 6.6 έρχεται στους τέσσερις χρήστες όπου το throughput πιάνει τα 1,38 req/s που από εκεί και πάνω δεν καταρρέει σταθεροποιείται γύρω στο 1,0 και σχηματίζει ουρά η εξυπηρέτηση μένει κοντά στην μια αίτηση ανά δευτερόλεπτο όσο και αν ανέβει το φορτίο οπότε ο χρόνος αναμονής είναι ουσιαστικά το πόσοι περιμένουν μπροστά του. Τριάντα δύο χρήστες με μια αίτηση το δευτερόλεπτο δίνουν 32 δευτερόλεπτα ακριβώς όπως η πρόβλεψη του νόμου του Little [20] για ουρά σε κορεσμό. Είναι όμως και η χειρότερη περίπτωση, γιατί το benchmark μετράει μόνο την ανάκτηση, ενώ στην πραγματική χρήση ο κάθε χρήστης περιμένει 3 δευτερόλεπτα το μοντέλο αφήνοντας τη CPU ελεύθερη για τους υπόλοιπους.

Στα 418 τμήματα της βιβλιοθήκης καμία από τις δύο αναζητήσεις δεν κοστίζει τίποτα στο σύστημα και σε συνθετική δοκιμή με 50.000 τμήματα το BM25 [9] φτάνει τα 61 ms ενώ η διανυσματική παραμένει σταθερή. Αν η βιβλιοθήκη μεγαλώσει το πρώτο που θα χρειαστεί είναι το ευρετήριο λέξεων και όχι η βάση διανυσμάτων.

Κεφάλαιο 7

Αξιολόγηση ευχρηστίας από χρήστες

Το προηγούμενο κεφάλαιο μέτρησε αν το σύστημα βρίσκει το σωστό υλικό και εάν οι απαντήσεις του στηρίζονται σε αυτό. Οι μετρήσεις έγιναν τοπικά και βαθμολογήθηκαν από εργαλεία, χωρίς να έχει χρησιμοποιήσει κανείς την εφαρμογή. Ένα εργαλείο μπορεί να είναι σωστό αλλά το σύστημα χρησιμοποιείται από ανθρώπους οπότε δημιουργήθηκε ένα ερωτηματολόγιο για το πώς φάνηκε η εφαρμογή σε ανθρώπους που μπήκαν και την είδαν για πρώτη φορά.

Η διάκριση σε αυτά τα δύο έχει σημασία γιατί τα αποτελέσματα του ερωτηματολογίου δεν δίνουν κάποια απάντηση στο αν οι απαντήσεις ήταν σωστές αλλά μόνο στο κομμάτι το πώς έμοιαζε η εμπειρία χρήσης της εφαρμογής όπως την ανέφεραν οι ίδιοι οι χρήστες. Η ορθότητα έχει μετρηθεί ξεχωριστά στο προηγούμενο κεφάλαιο.

7.1 Το εργαλείο μέτρησης

Για την μέτρηση του ερωτηματολογίου χρησιμοποιήθηκε το SUS (System Usability Scale), το οποίο είναι ένα ερωτηματολόγιο δέκα ερωτήσεων που πρότεινε ο Brooke το 1996 [21] και έχει γίνει από τότε ένα από τα πιο διαδεδομένα εργαλεία του είδους. Ο χρήστης απαντάει σε κάθε ερώτηση με μια κλίμακα από το ένα έως το πέντε, όπου το ένα είναι η πλήρης διαφωνία και το πέντε η πλήρης συμφωνία.

Δεν επιλέχθηκε επειδή είναι το πληρέστερο γιατί όπως φαίνεται και από τις ερωτήσεις δεν είναι αλλά για άλλους δύο λόγους. Ξεκινώντας από το πρώτο είναι ότι δίνει αριθμό συγκρίσιμο με κάτι έξω από αυτή την εργασία και μπορεί να συγκριθεί με άλλα επειδή ότι το συγκεκριμένο έχει εφαρμοστεί σε χιλιάδες συστήματα και υπάρχει δημοσιευμένο σημείο αναφοράς για ό,τι θεωρείται καλό [22]. Ένα ερωτηματολόγιο φτιαγμένο ειδικά για την συγκεκριμένη εφαρμογή μπορεί να ήταν πληρέστερο σε ερωτήσεις αλλά ο μέσος όρος που θα έβγαινε δεν θα σήμαινε τίποτα, επειδή ένα μοναδικό ερωτηματολόγιο δεν συγκρίνεται με κανένα άλλο. Ο δεύτερος λόγος αφορά το μέγεθος: δέκα ερωτήσεις συμπληρώνονται σε δύο λεπτά και δεν κουράζουν τον χρήστη.

Το χαρακτηριστικό του SUS που επηρεάζει όλα τα υπόλοιπα είναι ότι οι ερωτήσεις εναλλάσσονται σε διατύπωση δηλαδή οι μονές ερωτήσεις είναι θετικές δηλαδή το 5 σαν απάντηση είναι καλό ενώ οι ζυγές είναι οι αρνητικές ερωτήσεις δηλαδή το 5 σαν απάντηση είναι κακό. Η εναλλαγή αυτή είναι σκόπιμη και υπάρχει ώστε να μην μπορέσει ο χρήστης να απαντήσει το ερωτηματολόγιο μηχανικά με 4-5 σε όλα αλλά πρέπει να διαβάσει τις ερωτήσεις ώστε να δώσει μια πιο σωστή απάντηση.

7.2 Η διαδικασία

Οι απαντήσεις οι οποίες συγκεντρώθηκαν είναι είκοσι εννέα στο διάστημα από τις 26 Αυγούστου μέχρι 4 Σεπτεμβρίου του 2026. Το ερωτηματολόγιο δόθηκε στην αρχική αγγλική του διατύπωση χωρίς μετάφραση μέσω ηλεκτρονικής φόρμας με τους συμμετέχοντες να δοκιμάζουν πρώτα την εφαρμογή και αμέσως μετά να συμπληρώνουν το ερωτηματολόγιο.

Οι συμμετέχοντες δεν επιλέχθηκαν με κάποιο κριτήριο αλλά όσοι προσφέρθηκαν να πάρουν μέρος κάτι που σημαίνει ότι ο αριθμός ο οποίος ακολουθεί περιγράφει αυτούς τους είκοσι εννέα χρήστες.

7.3 Ο υπολογισμός του σκορ

Το σκορ του SUS δεν είναι ο μέσος όρος των απαντήσεων και αυτό γιατί οι ζυγές ερωτήσεις όπως αναφέρθηκε και πιο πάνω είναι αρνητικά διατυπωμένες οπότε ένας απλός μέσος όρος θα πρόσθετε μαζί πράγματα που δείχνουν αντίθετα δηλαδή ότι το πέντε στην τρίτη ερώτηση είναι ότι η εφαρμογή είναι εύκολη ενώ το πέντε στην όγδοη ερώτηση σημαίνει ότι είναι δύσκολη. Ο υπολογισμός γίνεται με διαφορετικό τρόπο και χωρίζεται σε τρία βήματα.

Πρώτα κάθε απάντηση μετατρέπεται σε συνεισφορά από μηδέν έως τέσσερα όπου το τέσσερα είναι πάντα το καλό. Στις μονές ερωτήσεις αφαιρείται ένα από την απάντηση στις ζυγές αφαιρείται η απάντηση από το πέντε. Μετά αθροίζονται οι δέκα συνεισφορές του κάθε συμμετέχοντα, δίνοντας έναν αριθμό από μηδέν έως σαράντα και τέλος το άθροισμα πολλαπλασιάζεται επί δυόμισι ώστε να απλωθεί σε κλίμακα από το μηδέν έως το εκατό.

Ο μέσος όρος βγαίνει στο τέλος πάνω στα είκοσι εννέα ολοκληρωμένα σκορ και όχι πάνω στις ωμές απαντήσεις, γιατί χωρίς την αντιστροφή των αρνητικών ερωτήσεων ο αριθμός δεν θα σήμαινε τίποτα. Ο μετασχηματισμός πάντως είναι γραμμικός, οπότε το ίδιο νούμερο βγαίνει και από το άθροισμα των μέσων συνεισφορών των δέκα ερωτήσεων επί δυόμισι, κάτι που λειτουργεί σαν έλεγχος ανάμεσα στους δύο πίνακες που ακολουθούν.

7.4 Αποτελέσματα

Η κλίμακα μπορεί να φτάνει έως το εκατό αλλά δεν είναι ποσοστό και το αποτέλεσμα που είναι 84,74 δεν σημαίνει ότι το 84,74% των χρηστών είναι ικανοποιημένοι ούτε ότι η εφαρμογή πέρασε κάποιο διαγώνισμα με αυτόν τον βαθμό αλλά είναι η θέση της πάνω σε μια κλίμακα. Τα συγκεντρωτικά μεγέθη δίνονται στον πίνακα 7.1.

Πίνακας 7.1: Συγκεντρωτικά αποτελέσματα του SUS, n = 29.

Μέγεθος	Τιμή
Μέσος όρος	84,74
Διάμεσος	85
Τυπική απόκλιση	10,42
Διάστημα εμπιστοσύνης 95%	[80,8–88,7]
Ελάχιστο – Μέγιστο	55–100

Το δημοσιευμένο σημείο αναφοράς για το SUS είναι το 68, που αντιστοιχεί στον μέσο όρο εκατοντάδων συστημάτων [22]. Ο μέσος όρος εδώ το ξεπερνά κατά σχεδόν δεκαεπτά μονάδες, και το διάστημα εμπιστοσύνης δεν το αγγίζει καν: ακόμα και το κάτω άκρο του,

το 80,8, μένει πάνω από το σημείο αναφοράς. Είκοσι επτά από τους είκοσι εννέα συμμετέχοντες έδωσαν σκορ πάνω από το 68.

Η διάμεσος είναι ελαφρώς υψηλότερη από τον μέσο όρο και η κατανομή έχει ουρά προς τα κάτω, δηλαδή οι βαθμολογίες είναι συγκεντρωμένες ψηλά και λίγες μεμονωμένες χαμηλώνουν τον μέσο όρο. Είκοσι επτά από τις είκοσι εννέα είναι 72,5 και πάνω, ενώ οι δύο χαμηλότερες απέχουν περισσότερο από δύο τυπικές αποκλίσεις. Η χαμηλότερη ήταν 55. Στη συγκεκριμένη περίπτωση ο συμμετέχων απάντησε ένα στην πρώτη και στην τρίτη και πέντε στην όγδοη δηλαδή δήλωσε ότι δεν θα την χρησιμοποιούσε συχνά και ότι την βρήκε δύσχρηστη ενώ την ίδια στιγμή την βαθμολόγησε πέντε στην ενσωμάτωση των λειτουργιών και στην σιγουριά που ένιωσε δείχνοντας ότι ο συνδυασμός αυτός είναι αντιφατικός και δεν επιτρέπει ασφαλές συμπέρασμα προς βελτίωση. Όπως όλα τα αποτελέσματα όμως κρατιέται και δεν αφαιρείται, ούτε επειδή χαλάει τον μέσο όρο ούτε επειδή δεν συμφωνεί κάποιος με αυτήν την άποψη. Τα ανά ερώτηση αποτελέσματα δίνονται στον πίνακα 7.2. Οι τιμές είναι οι μέσες συνεισφορές μετά την αντιστροφή, σε κλίμακα από μηδέν έως τέσσερα, όπου το τέσσερα είναι πάντα το καλό.

Πίνακας 7.2: Μέση συνεισφορά ανά ερώτηση, μετά την αντιστροφή των αρνητικών. Κλίμακα 0–4, όπου 4 είναι το καλύτερο.

#	Ερώτηση	Μέση τιμή
1	Θα τη χρησιμοποιούσα συχνά	2,90
2	Την βρήκα αδικαιολόγητα περίπλοκη	3,41
3	Ήταν εύκολη στη χρήση	3,07
4	Θα χρειαζόμουν βοήθεια από ειδικό	3,62
5	Οι λειτουργίες ήταν καλά ενσωματωμένες	3,24
6	Υπήρχε υπερβολική ασυνέπεια	3,69
7	Οι περισσότεροι θα τη μάθαιναν γρήγορα	3,38
8	Ήταν δύσχρηστη	3,59
9	Ένωσα σιγουριά χρησιμοποιώντας την	3,59
10	Έπρεπε να μάθω πολλά πριν ξεκινήσω	3,41

7.5 Τι δείχνουν οι επιμέρους ερωτήσεις

Ο συνολικός αριθμός κρύβει το ενδιαφέρον μέρος. Οι δέκα ερωτήσεις δεν βαθμολογήθηκαν ομοιόμορφα, και η διαφορά μεταξύ της υψηλότερης και της χαμηλότερης λείει περισσότερα από τον μέσο όρο.

Την υψηλότερη τιμή έχει η έκτη ερώτηση για την ασυνέπεια, με βαθμό 3,69 δηλαδή οι χρήστες δεν συνάντησαν στοιχεία που συμπεριφέρονται με διαφορετικό τρόπο από ό,τι έχουν μάθει οι ίδιοι. Το περιθώριο να εμφανιστεί ασυνέπεια είναι εξ αρχής στενότερο και η ερώτηση έχει εδώ μικρότερη διακριτική ικανότητα από όση θα είχε σε ένα σύστημα με πολλές απλές οθόνες και ροές. Το εύρημα είναι συμβατό με τη σχεδιαστική επιλογή ενός σκόπιμα περιορισμένου περιβάλλοντος, αλλά δεν μπορεί να σταθεί ως ανεξάρτητη ένδειξη ποιότητας σχεδιασμού.

Την χαμηλότερη έχει η πρώτη ερώτηση, δηλαδή αν ο χρήστης θα την χρησιμοποιούσε συχνά, η οποία έχει πάρει τον βαθμό 2,90. Η απόκλιση από τις υπόλοιπες είναι σαφής και η εξήγηση δεν είναι κατ' ανάγκη αρνητική για την εφαρμογή γιατί δεν μετράει αν είναι καλή η εφαρμογή, αλλά αν την έχει κάποιος μεγάλη ανάγκη. Η εφαρμογή απευθύνεται σε ένα

σύνολο ανθρώπων οι οποίοι έχουν την δική τους συλλογή από επιστημονικά κείμενα και θέλουν να μάθουν πράγματα για αυτά τα κείμενα που μάλλον κάποιοι συμμετέχοντες δεν είχαν αυτή την ανάγκη.

Οι Lewis και Sauro πρότειναν το 2009 τον χωρισμό [23] του SUS σε δύο υποκλίμακες, την ευχρηστία από τις οκτώ ερωτήσεις και την ευκολία μάθησης από δύο, την τέταρτη και την δέκατη. Εδώ η ευχρηστία βγαίνει 83,9 και η ευκολία μάθησης 87,9. Η δεύτερη είναι υψηλότερη, κάτι που συμφωνεί με τις ίδιες τις απαντήσεις: κανένας από τους είκοσι εννέα δεν δήλωσε ότι θα χρειαζόταν βοήθεια από ειδικό και είκοσι οκτώ στους είκοσι εννέα ότι δεν χρειάστηκε να μάθουν πολλά πριν ξεκινήσουν.

Η διαφορά αυτών των τεσσάρων μονάδων, σε συνδυασμό με το μέγεθος του δείγματος, δεν επαρκεί για να στηρίξει στατιστικά αυτόν τον ισχυρισμό. Αξίζει επίσης να σημειωθεί ότι οι ίδιοι Lewis και Sauro μετά από κάποια χρόνια αμφισβήτησαν αυτόν τον χωρισμό [24] παρατηρώντας ότι αυτές οι δύο υποκλίμακες διαχωρίζονται κυρίως επειδή είναι και οι δύο αρνητικά διατυπωμένες όχι επειδή μετράνε πραγματικά διαφορετικό πράγμα. Ο χωρισμός αυτός δίνεται εδώ για πληρότητα και όχι σαν εύρημα.

7.6 Περιορισμοί της μέτρησης

Ο αριθμός που δίνει το SUS περιγράφει αυτούς τους είκοσι εννέα ανθρώπους και όχι κάποιον ευρύτερο πληθυσμό. Η επιλογή τους δεν έγινε με κλήρωση αλλά με βάση το ποιος προσφέρθηκε οπότε το διάστημα εμπιστοσύνης του πίνακα 7.1 πρέπει να διαβάζεται σαν περιγραφή της διασποράς μέσα στο δείγμα και όχι σαν εκτίμηση για το τι θα έλεγαν άλλοι χρήστες. Το ίδιο ισχύει και για την σύγκριση με το 68 που είναι ο μέσος όρος εκατοντάδων συστημάτων και όχι κάποιο ανταγωνιστικό εργαλείο για την ίδια δουλειά. Η μέτρηση τοποθετεί την εφαρμογή πάνω σε μια γενική κλίμακα δεν την συγκρίνει με εναλλακτική λύση.

Το ερωτηματολόγιο όπως αναφέρθηκε δόθηκε στην αγγλική του διατύπωση σε ελληνόφωνους συμμετέχοντες. Ο Finstad [25] έδειξε ότι όσοι δεν έχουν τα αγγλικά ως μητρική γλώσσα δυσκολεύονται ιδιαίτερα με την όγδοη ερώτηση εκεί όπου η λέξη *cumbersome* δεν έχει προφανή αντιστοιχία. Η συνεισφορά της συγκεκριμένης ερώτησης κουβαλάει επομένως μεγαλύτερη αβεβαιότητα από τις υπόλοιπες. Η μετάφραση αποφεύχθηκε ώστε να μείνει το εργαλείο στην μορφή για την οποία υπάρχουν τα δημοσιευμένα σημεία αναφοράς, αλλιώς η σύγκριση με το 68 δεν θα στεκόταν.

Κάθε συμμετέχων δοκίμασε την εφαρμογή μία φορά και συμπλήρωσε το ερωτηματολόγιο αμέσως μετά. Αυτό που καταγράφεται είναι η πρώτη εντύπωση και όχι η εμπειρία κάποιου που θα την χρησιμοποιούσε για εβδομάδες πάνω στην δική του συλλογή. Δύσκολα φαίνονται σε μία συνεδρία πράγματα όπως η κόπωση από την επαναλαμβανόμενη χρήση ή το τι γίνεται όταν η βιβλιοθήκη μεγαλώσει, και ο βαθμός 2,90 της πρώτης ερώτησης που σχολιάστηκε πιο πάνω δείχνει ακριβώς ότι οι περισσότεροι δεν είχαν την ανάγκη την οποία εξυπηρετεί η εφαρμογή.

Τέλος το SUS μετράει αντιλαμβανόμενη ευχρηστία και τίποτα άλλο. Το 84,74 δεν λέει αν οι απαντήσεις που πήραν οι συμμετέχοντες ήταν σωστές ούτε αν οι παραπομπές έδειχναν πράγματι εκεί που έπρεπε. Ένα σύστημα μπορεί να είναι ευχάριστο στην χρήση και να απαντάει λάθος και το αντίστροφο. Αυτό το κομμάτι το καλύπτει το έκτο κεφάλαιο με εντελώς άλλα εργαλεία και τα δύο νούμερα πρέπει να διαβάζονται μαζί.

Κεφάλαιο 8

Συμπεράσματα

Η εργασία ξεκίνησε από ένα πρόβλημα το οποίο δεν λύνεται με κάποιο καλύτερο μοντέλο, δηλαδή ότι ένα γλωσσικό μοντέλο όταν του δίνεις υλικό για να απαντήσει από αυτό γράφει πειστικά ακόμα και όταν δεν υπάρχει στο υλικό που του δίνεις και αυτός που το διαβάζει δεν μπορεί να ξεχωρίσει αν είναι από το υλικό που του δόθηκε ή από τις δικές του γνώσεις. Για αυτό το σύστημα Z-AI Platform απαντάει με τον μοναδικό τρόπο που μπορεί να ελεγχθεί, δηλαδή απαντάει μόνο από το υλικό που του έχει δοθεί από τον χρήστη και αυτό γίνεται δείχνοντας σε κάθε απάντηση το αρχείο και την σελίδα από την οποία πήρε το κάθε κομμάτι και όταν το υλικό δεν υπάρχει το λέει αντί να το συμπληρώσει μόνο του με δικές του γνώσεις.

8.1 Ανακεφαλαίωση

Σχεδιάστηκε και υλοποιήθηκε μια εφαρμογή ιστού με λογαριασμούς και ιδιωτικά έγγραφα στην οποία ο χρήστης μπορεί να ανεβάσει τα δικά του άρθρα μια φορά στην βιβλιοθήκη και να κάνει ερωτήσεις πάνω σε αυτά όσες φορές θελήσει. Το κείμενο επεξεργάζεται με το που ανέβει μια φορά και μένει αποθηκευμένο οπότε η κάθε ερώτηση δεν πληρώνει ξανά το κόστος της ανάγνωσης ούτε χρειάζεται να το ξαναδιαβάσει.

Η διαδρομή την οποία ακολουθεί η ερώτηση χτίστηκε στάδιο το στάδιο και το καθένα από αυτά περιγράφεται στο κεφάλαιο 4, ενώ το κεφάλαιο 6 δείχνει με μετρήσεις γιατί μπήκε το καθένα ξεχωριστά και τι θα άλλαζε αν έλειπε.

8.2 Τι έδειξαν οι μετρήσεις

Το κάθε στάδιο μπήκε μετά από μέτρηση η οποία έγινε και όχι επειδή υπάρχει στην βιβλιογραφία. Από την σκέτη διανυσματική αναζήτηση στην υβριδική το MRR ανέβηκε από το 0,731 σε 0,774 και με τον cross-encoder φτάνει στο 0,793, κέρδη μικρά. Η πύλη συνάφειας δεν κουνάει το MRR καθόλου γιατί δεν είναι ο σκοπός της γιατί αυτό που αλλάζει είναι το είδος της συμπεριφοράς αφού το σύστημα περνάει από το να απαντάει και στις πέντε ερωτήσεις στο να μην απαντάει στις πέντε ερωτήσεις που υπάρχουν χωρίς υλικό.

Στην ποιότητα της απάντησης οι δύο κριτές οι οποίοι υπάρχουν ελέγχουν με διαφορετική μέθοδο και καταλήγουν στο ίδιο σημείο. Η πιστότητα βγαίνει 0,982 με τον έναν και 0,978 με τον άλλο ενώ η ανάκληση συμφραζομένων είναι 1,000 και στους δύο σε καθεμία από τις σαράντα πέντε ερωτήσεις δηλαδή το υλικό που χρειάζεται η σωστή απάντηση

έφτασε με επιτυχία πάντα στο μοντέλο και τέλος η κάλυψη λέξεων κλειδιών στο κύριο σετ κυμαίνεται από 97,78% έως 98,52% ανάλογα με το τρέξιμο.

Η υποστήριξη των ελληνικών δεν είναι απλά μια μετάφραση της ερώτησης και αυτό παρατηρείται ότι στην δεύτερη συλλογή που υπάρχουν μόνο ελληνικές ερωτήσεις και τα κείμενα ιατρικά και ηφαιστειολογικά η κάλυψη χωρίς λεξιλόγιο μένει 85% και αυτό γιατί κάθε θέμα έχει το δικό του λεξιλόγιο ενώ με το λεξιλόγιο πάει στο 95%. Από τις έντεκα παραλλαγές οι οποίες δοκιμάστηκαν μόνο η περιοριστική εκδοχή κρατάει και τα δύο δηλαδή το 95% κάλυψη και το να μην υπάρχει καμία διαρροή στις ερωτήσεις για αυτές που δεν έχουν υλικό στα άρθρα, και ο λόγος είναι ότι επιτρέπει όρο του λεξιλογίου ως μετάφραση της λέξης η οποία υπάρχει ήδη στην ερώτηση.

Πραγματοποιήθηκε και ένα ερωτηματολόγιο το SUS από χρήστες και βγήκε 84,74 με διάστημα εμπιστοσύνης [80,8–88,7] και είκοσι επτά από τους είκοσι εννέα που έκαναν το ερωτηματολόγιο έδωσαν βαθμό πάνω από το σημείο αναφοράς του 68 που υπάρχει. Το ενδιαφέρον δεν είναι ο μέσος όρος αλλά η χειρότερη ερώτηση η οποία είχε 2,90 και αναφέρεται στο αν ο χρήστης θα την χρησιμοποιούσε συνέχεια το οποίο είναι κάτι που δεν μετράει αν η εφαρμογή είναι καλή ή όχι αλλά αν την έχει ανάγκη.

Μια τελευταία παρατήρηση αφορά τα ίδια τα εργαλεία της μέτρησης. Το MRR δεν προβλέπει την ποιότητα της απάντησης και οι βαθμοί των κριτών δίνουν επίπεδο και όχι μέτρο σύγκρισης, αφού όπως παρατηρείται η ίδια ερώτηση με ταυτόσημα αποσπάσματα και ταυτόσημη απάντηση πήρε ακρίβεια συμφραζομένων 0,000 την μια φορά και 1,000 την άλλη. Όποιο συμπέρασμα βγαίνει από εδώ στηρίζεται σε συγκρίσεις με τον ίδιο κώδικα ώστε να μπορεί να γίνει σύγκριση μεταξύ τους σε διαφορές μεγαλύτερες από το περιθώριο του θορύβου, δηλαδή στην άρνηση και στις πέντε ερωτήσεις χωρίς υλικό, στην κάλυψη η οποία πάει από το 85% στο 95% και στην συμφωνία ανάμεσα στους δύο κριτές και όχι στις μικρές διαφορές του MRR ανάμεσα σε δύο ρυθμίσεις.

8.3 Τα όρια του συστήματος

Η ανάκτηση δεν αποτυγχάνει ομοιόμορφα. Οι ερωτήσεις γεγονότος φτάνουν 0,869 και οι αιτιολόγησης 0,831 ενώ οι απαριθμήσεις πέφτουν στο 0,751 επειδή η λίστα ξεπερνάει τα όρια του τμήματος και το πρώτο σωστό απόσπασμα περιέχει μέρος μόνο της απάντησης. Το χαμηλότερο νούμερο το έχουν οι ερωτήσεις πολλαπλών βημάτων με 0,541 σε τέσσερις μόνο ερωτήσεις οπότε το ίδιο το νούμερο δεν αντέχει βάρος, η αδυναμία όμως φαίνεται και στην άλλη άκρη του συστήματος αφού από τις τέσσερις απαντήσεις οι οποίες δεν πήραν άριστα από τον πρώτο κριτή οι δύο είναι ακριβώς αυτού του είδους.

Το μέγεθος του τμήματος είναι κλειστό και από τις δύο μεριές. Προς τα κάτω τα 750 δίνουν καλύτερο MRR στο κύριο σετ αλλά στις ερωτήσεις πολλαπλών βημάτων πέφτουν στο 0,079 έναντι 0,204 των 1500 γιατί σπάνε στα δύο την απάντηση, και προς τα πάνω σταματάει ο cross-encoder ο οποίος δέχεται 512 tokens και τα υπόλοιπα τα κόβει.

Στο φορτίο το όριο είναι καθαρό. Με έναν χρήστη η ανάκτηση παίρνει 846 ms από τα οποία τα 784 είναι η αναδιάταξη σε επεξεργαστή χωρίς κάρτα γραφικών, ο κορεσμός έρχεται στους τέσσερις χρήστες και από εκεί και πάνω η εξυπηρέτηση μένει γύρω στην μια αίτηση το δευτερόλεπτο οπότε ο χρόνος αναμονής είναι απλώς το πόσοι περιμένουν μπροστά. Η βιβλιοθήκη αντέχει πολύ περισσότερο, αφού στα 418 τμήματα καμία από τις δύο αναζητήσεις δεν κοστίζει τίποτα και σε συνθετική δοκιμή με 50.000 τμήματα το BM25 φτάνει τα 61 ms ενώ η διανυσματική μένει σταθερή.

8.4 Μελλοντικές επεκτάσεις

Η πρώτη επέκταση βγαίνει από την μέτρηση της ενότητας 6.2.3. Οι ερωτήσεις πολλαπλών βημάτων χρειάζονται σπάσιμο της ερώτησης σε υποερωτήματα και δεύτερο γύρο ανάκτησης πάνω στο αποτέλεσμα του πρώτου, και η γενική μορφή αυτού είναι ένα σχήμα το οποίο αποφασίζει μόνο του πόσους γύρους χρειάζεται και πότε σταματάει. Το σύστημα κάνει ήδη ένα τέτοιο βήμα, δηλαδή ξαναγράφει την ερώτηση και ξανατρέχει ολόκληρη την διαδρομή όταν το καλύτερο σκορ δεν περάσει το κατώφλι. Δύο πράγματα εμποδίζουν το να γίνει ελεύθερος. Το πρώτο είναι ότι δεν υπάρχει σύνολο ερωτήσεων το οποίο να μπορεί να το κρίνει αφού οι τέσσερις ερωτήσεις πολλαπλών βημάτων του κύριου σετ δεν φτάνουν για να φανεί αν μια τέτοια αλλαγή θα βοηθούσε ή όχι. Το δεύτερο είναι το κόστος γιατί ο κάθε γύρος πληρώνει από την αρχή ολόκληρη την αναδιάταξη δηλαδή τα εννέα δέκατα του χρόνου της ανάκτησης σε ένα σύστημα το οποίο φτάνει στον κορεσμό του ήδη στους τέσσερις χρήστες.

Οι απαριθμήσεις σκοντάφτουν σε άλλο εμπόδιο. Μια τμηματοποίηση η οποία κρατάει ολόκληρη την λίστα μαζί δίνει τμήματα μεγαλύτερα από όσα σηκώνει ο cross-encoder στα 512 tokens, οπότε το ζητούμενο δεν είναι ένας άλλος τρόπος κοψίματος του κειμένου αλλά σχήμα αναδιάταξης το οποίο να μην χρειάζεται να κόψει το τμήμα για να το βαθμολογήσει.

Το δεύτερο σκέλος αφορά την κλίμακα. Το όριο των τεσσάρων χρηστών δεν μετακινείται με ρύθμιση, οπότε ο δρόμος είναι είτε κάρτα γραφικών είτε ξεχωριστή υπηρεσία η οποία θα δέχεται τις αιτήσεις μαζεμένες ώστε να μοιράζεται το κόστος ανάμεσά τους.

Το τρίτο αφορά τον τρόπο με τον οποίο μετρήθηκε η ευχρηστία. Οι είκοσι εννέα συμμετέχοντες δοκίμασαν την εφαρμογή μία φορά και έδωσαν την πρώτη τους εντύπωση, ενώ αυτό που θα έλεγε περισσότερα είναι η χρήση για εβδομάδες από ανθρώπους οι οποίοι έχουν δική τους συλλογή από άρθρα και πραγματική ανάγκη να τη διαβάσουν.

Αυτό που κρατάει η εργασία είναι ότι ένα σύστημα RAG δεν κρίνεται από το πόσο καλά διαβάζονται οι απαντήσεις του αλλά κρίνεται από το αν το υλικό στο οποίο στηρίζεται φτάνει πράγματι στο μοντέλο, από το αν δίνονται οι παραπομπές στον χρήστη μαζί με την απάντηση ώστε να μπορεί να πάει στην σελίδα και να επιβεβαιώσει την απάντηση και από το αν το σύστημα απαντάει με δεν ξέρω όταν δεν έχει βρει κάτι στο υλικό που του δόθηκε. Και τα τρία είναι σχεδιαστικές αποφάσεις οι οποίες κοστίζουν, σε χρόνο και σε απαντήσεις οι οποίες δεν δίνονται, και αυτή η εργασία έδειξε πόσο ακριβώς κοστίζουν αυτά.

Βιβλιογραφία

- [1] G. W. Furnas, T. K. Landauer, L. M. Gomez και S. T. Dumais, «The Vocabulary Problem in Human-System Communication», *Communications of the ACM*, τόμ. 30, αρθμ. 11, σσ. 964–971, 1987. doi: 10.1145/32206.32212
- [2] Z. Ji κ.ά., «Survey of Hallucination in Natural Language Generation», *ACM Computing Surveys*, τόμ. 55, αρθμ. 12, σσ. 1–38, 2023. doi: 10.1145/3571730
- [3] N. F. Liu κ.ά., «Lost in the Middle: How Language Models Use Long Contexts», *Transactions of the Association for Computational Linguistics*, τόμ. 12, σσ. 157–173, 2024. doi: 10.1162/tacl_a_00638
- [4] P. Lewis κ.ά., «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks», στο *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020, σσ. 9459–9474.
- [5] A. Vaswani κ.ά., «Attention Is All You Need», στο *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017, σσ. 5998–6008.
- [6] N. Reimers και I. Gurevych, «Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks», στο *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*, 2019, σσ. 3982–3992. doi: 10.18653/v1/D19-1410
- [7] V. Karpukhin κ.ά., «Dense Passage Retrieval for Open-Domain Question Answering», στο *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, σσ. 6769–6781. doi: 10.18653/v1/2020.emnlp-main.550
- [8] J. Johnson, M. Douze και H. Jégou, «Billion-Scale Similarity Search with GPUs», *IEEE Transactions on Big Data*, τόμ. 7, αρθμ. 3, σσ. 535–547, 2021. doi: 10.1109/TBDATA.2019.2921572
- [9] S. Robertson και H. Zaragoza, «The Probabilistic Relevance Framework: BM25 and Beyond», *Foundations and Trends in Information Retrieval*, τόμ. 3, αρθμ. 4, σσ. 333–389, 2009. doi: 10.1561/15000000019
- [10] G. V. Cormack, C. L. A. Clarke και S. Buettcher, «Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods», στο *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2009, σσ. 758–759. doi: 10.1145/1571941.1572114
- [11] R. Nogueira και K. Cho, «Passage Re-ranking with BERT», *arXiv preprint arXiv:1901.04085*, 2019.
- [12] N. Provos και D. Mazières, «A Future-Adaptable Password Scheme», στο *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, 1999, σσ. 81–91.

- [13] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian και Z. Liu, «BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation», *arXiv preprint arXiv:2402.03216*, 2024.
- [14] The Unicode Consortium. «Unicode Standard Annex #15: Unicode Normalization Forms», επίσκεψη 6 Σεπτ. 2026. διεύθυν.: <https://www.unicode.org/reports/tr15/>
- [15] S. Es, J. James, L. Espinosa-Anke και S. Schockaert, «RAGAS: Automated Evaluation of Retrieval Augmented Generation», στο *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2024): System Demonstrations*, 2024, σσ. 150–158.
- [16] S.-Q. Yan, J.-C. Gu, Y. Zhu και Z.-H. Ling, «Corrective Retrieval Augmented Generation», *arXiv preprint arXiv:2401.15884*, 2024.
- [17] E. M. Voorhees, «The TREC-8 Question Answering Track Report», στο *Proceedings of the 8th Text REtrieval Conference (TREC-8)*, 1999, σσ. 77–82.
- [18] L. Zheng κ.ά., «Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena», στο *Advances in Neural Information Processing Systems 36 (NeurIPS 2023) Datasets and Benchmarks Track*, 2023.
- [19] Z. Yang κ.ά., «HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering», στο *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018, σσ. 2369–2380. doi: 10.18653/v1/D18-1259
- [20] J. D. C. Little, «A Proof for the Queuing Formula: $L = \lambda W$ », *Operations Research*, τόμ. 9, αριθμ. 3, σσ. 383–387, 1961. doi: 10.1287/opre.9.3.383
- [21] J. Brooke, «SUS: A ‘Quick and Dirty’ Usability Scale», στο *Usability Evaluation in Industry*, P. W. Jordan, B. Thomas, B. A. Weerdmeester και I. L. McClelland, επιμελητές, London: Taylor & Francis, 1996, σσ. 189–194.
- [22] J. Sauro, *A Practical Guide to the System Usability Scale: Background, Benchmarks & Best Practices*. Denver, CO: Measuring Usability LLC, 2011.
- [23] J. R. Lewis και J. Sauro, «The Factor Structure of the System Usability Scale», στο *Human Centered Design (HCD 2009)*, σειρά Lecture Notes in Computer Science, τόμ. 5619, Springer, 2009, σσ. 94–103. doi: 10.1007/978-3-642-02806-9_12
- [24] J. R. Lewis και J. Sauro, «Revisiting the Factor Structure of the System Usability Scale», *Journal of Usability Studies*, τόμ. 12, αριθμ. 4, σσ. 183–192, 2017.
- [25] K. Finstad, «The System Usability Scale and Non-Native English Speakers», *Journal of Usability Studies*, τόμ. 1, αριθμ. 4, σσ. 185–188, 2006.