ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΥΠΟΛΟΓΙΣΤΩΝ

Διερεύνηση αξιοποίησης εργαλείων LLM στην αρχιτεκτονική λογισμικού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

ΜΙΧΑΗΛ ΤΣΙΛΙΜΙΓΚΟΥΝΑΚΗ

Επιβλέπων : Βασίλειος Βεσκούκης, Καθηγητής ΕΜΠ

Αθήνα, Νοέμβριος 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΥΠΟΛΟΓΙΣΤΩΝ

Διερεύνηση αξιοποίησης εργαλείων LLM στην αρχιτεκτονική λογισμικού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

ΜΙΧΑΗΛ ΤΣΙΛΙΜΙΓΚΟΥΝΑΚΗ

Επιβλέπων : Βασίλειος Βεσκούκης, Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20η Νοεμβρίου 2024.

Β. Βεσκούκης Καθηγητής ΕΜΠ	Ν. Παπασπύρου Καθηγητής ΕΜΠ	Γ. Στάμου Καθηγητής ΕΜΠ

Αθήνα, Νοέμβριος 2024

(Υπογραφή)

……..
Τσιλιμιγκουνάκης Μιχάλης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

Περίληψη

Η συνεισφορά των μεγάλων γλωσσικών μοντέλων (LLMs) σε όλες τις φάσεις του κύκλου ζωής ανάπτυξης λογισμικού έχει αποκτήσει ιδιαίτερη δυναμική με την εισαγωγή ισχυρών LLMs τα τελευταία χρόνια. Πέρα από τη δημιουργία κώδικα, αξίζει να διερευνηθεί εάν η χρήση των LLMs στα πρώιμα στάδια της ανάπτυξης λογισμικού, όπως στη κατανόηση απαιτήσεων και την αρχιτεκτονική, μπορεί να επιταχύνει την ανάπτυξη και να βελτιώσει την ποιότητα των συστημάτων λογισμικού. Στο πλαίσιο αυτό, η συνδρομή της τεχνητής νοημοσύνης στην αρχιτεκτονική λογισμικού θα μπορούσε να παράγει περιγραφές υψηλού επιπέδου ή ακόμα και λεπτομερείς αρχιτεκτονικές περιγραφές, όπως διαγράμματα κλάσεων ή διαγράμματα στοιχείων UML, από κειμενικές απαιτήσεις.

Η παρούσα εργασία διερευνά τις δυνατότητες και τους περιορισμούς επιλεγμένων εμπορικών (commercial) και ανοιχτού κώδικα (open-source) LLMs στη δημιουργία διαγραμμάτων κλάσεων που συμμορφώνονται με συγκεκριμένα αρχιτεκτονικά πρότυπα, όπως τα Client-Server, Three-Tier και Model-View-Controller. Μέσω συστηματικού σχεδιασμού και αξιολόγησης 480 σεναρίων, εκτιμάται ο τρόπος με τον οποίο παράγοντες όπως η περιγραφή των λειτουργικών και μη λειτουργικών απαιτήσεων, τα υλικά και οι διαδικασίες RAG (Retrieval-Augmented Generation), τα μοντέλα ενσωμάτωσης (embedding models) και η επιλογή συγκεκριμένων LLMs – από μικρότερα μοντέλα έως μεγαλύτερα, πιο εξελιγμένα μοντέλα – επηρεάζουν την ποιότητα των παραγόμενων διαγραμμάτων κλάσεων UML που περιγράφουν την αρχιτεκτονική λογισμικού για μια μικρής κλίμακας εφαρμογή.

Η μελέτη μας παρουσιάζει επίσης ένα προσαρμοσμένο εργαλείο για τη διαχείριση, την αξιολόγηση και την ανάλυση των παραγόμενων διαγραμμάτων κλάσεων, διευκολύνοντας την αξιολόγηση τόσο από ανθρώπινους εμπειρογνώμονες όσο και από συστήματα τεχνητής νοημοσύνης. Τα αποτελέσματα αυτής της εργασίας αποκαλύπτουν ένα ευρύ φάσμα αποτελεσμάτων, από καλά δομημένα διαγράμματα που ευθυγραμμίζονται με τις απαιτήσεις λογισμικού και τις ζητούμενες αρχιτεκτονικές αρχές, έως ελλιπείς ή λανθασμένες εξόδους που επηρεάζονται από τους περιορισμούς των μοντέλων, τις προκλήσεις του RAG και άλλους παράγοντες που σχετίζονται με το εγγενές φάσμα ερμηνειών του υλικού που χρησιμοποιείται για την εκπαίδευση των LLMs.

Αυτή η έρευνα αποτελεί ένα πρώτο βήμα στη διερεύνηση των δυνατοτήτων των LLMs για δημιουργική εργασία στον κύκλο ζωής του λογισμικού και, ειδικότερα, στον αρχιτεκτονικό σχεδιασμό. Εξετάζουμε επίσης πως τα τοπικά μοντέλα LLMs μπορούν να καθοδηγηθούν ώστε να παράγουν αρχιτεκτονικές που εφαρμόζουν τις αρχές που απαιτούνται από κάθε πλαίσιο ανάπτυξης, χωρίς να εκθέτουν πληροφορίες σε δημόσιες εμπορικές υπηρεσίες, υποστηρίζοντας την ασφαλή και αποτελεσματική αυτοματοποίηση στην ανάπτυξη λογισμικού.

Λέξεις κλειδιά : Μεγάλα γλωσσικά μοντέλα, αρχιτεκτονική λογισμικού, τεχνητή νοημοσύνη, Retrieval Augmented Generation (RAG), UML, prompt engineering, τοπικά LLMs.

Abstract

The support across all phases of the software life cycle by large language models (LLMs) has gained momentum with the introduction of powerful LLMs during the past couple of years. Apart from code generation, it is worth exploring if using LLMs in early phases of software development, such as requirements engineering and architecture, can accelerate development and improve the quality of software systems. In this regard, AI-support in software architecture could generate high-level or even detailed architectural descriptions, such as UML class or component diagrams, from textual requirements. This thesis explores the capabilities and limitations of selected commercial and open source LLMs in generating class diagrams that comply with specific architectural patterns, namely Client-Server, Three-Tier, and Model-View-Controller (MVC). Through a systematic planning and evaluation of 480 scenarios, we assess how factors such as the description of functional and non-functional requirements, RAG (Retrieval-Augmented Generation) materials and processes, embedding models, and the choice of specific LLMs, ranging from smaller quantized models to larger, more advanced ones, impact the quality of LLM-generated UML class diagrams that describe software architectures for a small application.

Our study also introduces a custom tool for managing, evaluating, and analyzing the generated class diagrams, facilitating both human expert and AI-based assessments. The results of this work reveal a broad range of outcomes, from well-structured diagrams aligned with software requirements and the targeted architectural principles to incomplete or erroneous outputs influenced by model limitations, RAG challenges and other factors relating to the inherent wide spectrum of interpretations of the materials that have been used in the training of LLMs. This research is a first step in the investigation of the potential of LLMs for creative work in the software life cycle, and specifically in architectural design. We also investigate how locally-run LLMs can be guided to produce architectures that apply the principles required by each development context, without exposing any information to publicly available commercial services, to support secure and efficient automation in software development.

Keywords: Large Language Models (LLMs) , software architecture, artificial intelligence, Retrieval Augmented Generation (RAG), UML, prompt engineering, local LLMs.

Ευχαριστίες

Με την ολοκλήρωση της διπλωματικής μου εργασίας, φτάνει στο τέλος του επίσης το πενταετές μου ταξίδι στη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου (ΕΜΠ). Αυτό δεν θα ήταν δυνατό χωρίς την παρουσία των φίλων και της οικογένειάς μου, που με στήριξαν καθ' όλη τη διάρκεια των προπτυχιακών μου σπουδών.

Θα ήθελα να ευχαριστήσω ειλικρινά τον καθηγητή μου, κ. Βασίλη Βεσκούκη, για την ευκαιρία να δουλέψουμε μαζί, καθώς και για την καθοδήγηση, τον χρόνο και την εξαιρετική συνεργασία κατά τη διάρκεια της διπλωματικής μου εργασίας.

Είμαι επίσης ιδιαίτερα ευγνώμων στον υποψήφιο διδάκτορα κ. Χρήστο Χατζηχριστοφή για τη πολύτιμη βοήθεια και τη συνεχή διαθεσιμότητά του να βρίσκεται δίπλα μου σε όλη αυτή τη διαδικασία. Εκτιμάω πραγματικά το χρόνο και την υποστήριξή τους.

Τέλος, εκφράζω τις ευχαριστίες μου σε όλους όσοι αποτέλεσαν μέρος αυτού του ταξιδιού στο Εθνικό Μετσόβιο Πολυτεχνείο και συνέβαλαν σε αυτή την εμπειρία

Πίνακας Περιεχομένων

Πίνακας εικόνων

1 Εισαγωγή

Η ανάπτυξη λογισμικού είναι μια σύνθετη και πολυεπίπεδη διαδικασία που περιλαμβάνει τη μετατροπή των απαιτήσεων των χρηστών σε πλήρως λειτουργικά συστήματα μέσω διαφόρων σταδίων, όπως η ανάλυση, ο σχεδιασμός, ο προγραμματισμός, οι δοκιμές και η συντήρηση. Ένα από τα πιο κρίσιμα βήματα είναι η αρχιτεκτονική και η φάση σχεδιασμού όπου ο ορισμός και η δόμηση high level συστατικών ικανοποιείται συνήθως από τη δημιουργία διαγραμμάτων της Ενοποιημένης Γλώσσας Μοντελοποίησης (Unified Modeling Language - UML), και ειδικότερα των διαγραμμάτων κλάσεων, τα οποία διαδραματίζουν κεντρικό ρόλο στον σχεδιασμό και την ανάλυση συστημάτων. Τα διαγράμματα κλάσεων παρέχουν μια δομική αναπαράσταση ενός συστήματος λογισμικού, καθορίζοντας τις κλάσεις, τα χαρακτηριστικά, τις μεθόδους και τις σχέσεις τους. Λειτουργούν ως ένα σχέδιο που καθοδηγεί ολόκληρη τη διαδικασία ανάπτυξης, βοηθώντας προγραμματιστές και ενδιαφερόμενους να κατανοήσουν καλύτερα την αρχιτεκτονική και τις αλληλεπιδράσεις εντός του συστήματος. Ωστόσο, παρά τη σημασία τους, η δημιουργία διαγραμμάτων κλάσεων παραμένει μια εργασία που απαιτεί πολύ χρόνο, είναι επιρρεπής σε σφάλματα και συχνά υπόκεινται σε ανθρώπινες παρανοήσεις. Αυτό μπορεί να οδηγήσει σε αναποτελεσματικότητα και πιθανές ανακρίβειες που επηρεάζουν αρνητικά ολόκληρο τον κύκλο ζωής της ανάπτυξης λογισμικού.

Η εμφάνιση της τεχνητής νοημοσύνης (Artificial Intelligence - AI), και ιδιαίτερα η ανάπτυξη μεγάλων γλωσσικών μοντέλων (Large Language Models - LLMs) βασισμένων στη βαθιά μάθηση, έχει ανοίξει νέες δυνατότητες για την αυτοματοποίηση εργασιών στην τεχνολογία λογισμικού. Αυτά τα μοντέλα AI, εκπαιδευμένα σε τεράστια σύνολα δεδομένων, έχουν επιδείξει αξιοσημείωτες ικανότητες στην κατανόηση και την παραγωγή φυσικής γλώσσας. Η πρόοδος αυτή έχει θέσει τις βάσεις για την εφαρμογή τους σε εργασίες όπως η δημιουργία κώδικα, η ανίχνευση σφαλμάτων και η αυτοματοποίηση σχεδιαστικών εργασιών, όπως η δόμηση αρχιτεκτονικών και η δημιουργία διαγραμμάτων UML. Η δυνατότητα αξιοποίησης της τεχνητής νοημοσύνης για τη δημιουργία ακριβών και καλά δομημένων αρχιτεκτονικών μέσω διαγραμμάτων κλάσεων απευθείας από τις κειμενικές απαιτήσεις λογισμικού προσφέρει μια ελπιδοφόρα λύση σε μια μακροχρόνια πρόκληση του σχεδιασμού λογισμικού.

Ο κύριος στόχος αυτής της μελέτης είναι η διερεύνηση της σκοπιμότητας και της αποτελεσματικότητας της χρήσης LLMs για τη δημιουργία αρχιτεκτονικών μέσω διαγραμμάτων UML, ειδικά στο πλαίσιο συγκεκριμένων αρχιτεκτονικών προτύπων, όπως η αρχιτεκτονική Πελάτη-Εξυπηρετητή (Client-Server) ή των Τριών Επιπέδων (Three-Tier). Εστιάζουμε στη μελέτη του τρόπου με τον οποίο διαφορετικές παράμετροι—όπως η δομή των λειτουργικών και μη λειτουργικών απαιτήσεων, η επιλογή του μοντέλου και η εφαρμογή της Μεθόδου Επαυξημένης Ανάκτησης (Retrieval-Augmented Generation - RAG)—επηρεάζουν την ποιότητα και την ακρίβεια των παραγόμενων διαγραμμάτων. Επιπλέον, στοχεύουμε να αξιολογήσουμε την αξιοπιστία τοπικών LLMs, από μικρότερα, αποδοτικά σε πόρους μοντέλα έως μεγαλύτερα και πιο προηγμένα, ως βιώσιμες εναλλακτικές λύσεις έναντι διαδικτυακών λύσεων AI όπως το ChatGPT και το Gemini. Αυτή η διερεύνηση καθοδηγείται από την αυξανόμενη ανάγκη των επιχειρήσεων για την προστασία ευαίσθητων δεδομένων, η οποία συχνά επιβάλλει την αποφυγή χρήσης διαδικτυακών LLMs υπέρ των τοπικά αναπτυγμένων μοντέλων. Αξιολογώντας την απόδοση αυτών των τοπικών μοντέλων όσον αφορά την τήρηση της αρχιτεκτονικής, τις σχέσεις κλάσεων, τη συνοχή, τη συζευκτικότητα και

τη συνέπεια με τις απαιτήσεις λογισμικού, επιδιώκουμε να παρέχουμε πολύτιμες πληροφορίες για τη χρησιμότητά τους ως ασφαλή και αποτελεσματικά εργαλεία αυτοματοποίησης σχεδιασμού λογισμικού.

Μέσω ενός συνδυασμού αξιολόγησης από ανθρώπινους ειδικούς και αξιολογήσεων με τη χρήση AI, αυτή η μελέτη επιδιώκει να αναδείξει τις δυνατότητες, τους περιορισμούς και τη δυνατότητα κλιμάκωσης τόσο των τοπικών όσο και των διαδικτυακών LLMs στον AI-υποβοηθούμενο σχεδιασμό λογισμικού.

Στις επόμενες ενότητες, θα αναλύσουμε τη μεθοδολογία και τη διαμόρφωση των πειραμάτων μας, θα παρουσιάσουμε το προσαρμοσμένο εργαλείο που αναπτύξαμε για τη διαχείριση και αξιολόγηση διαγραμμάτων κλάσεων, θα παρουσιάσουμε τα αποτελέσματα και τις αναλύσεις μας και θα κλείσουμε με μια συζήτηση για τις μελλοντικές κατευθύνσεις της έρευνας στον AI-υποβοηθούμενο σχεδιασμό αρχιτεκτονικής λογισμικού.

1.1 Η εξέλιξη του σύγχρονου AI

Η τεχνητή νοημοσύνη αναδεικνύεται ως κινητήρια δύναμη αλλαγής σε ποικίλους τομείς, προσφέροντας πρωτοποριακές λύσεις σε σύνθετες προκλήσεις και δημιουργώντας νέες προοπτικές. Στον πυρήνα της, η τεχνητή νοημοσύνη περιλαμβάνει την ανάπτυξη συστημάτων που μιμούνται την ανθρώπινη νοημοσύνη, εκτελώντας εργασίες όπως λογική, μάθηση, λήψη αποφάσεων και κατανόηση γλώσσας. Με την αξιοποίηση προηγμένων αλγορίθμων, το AI επιτρέπει στις μηχανές να επεξεργάζονται μεγάλες ποσότητες δεδομένων, να προσαρμόζονται σε δυναμικά περιβάλλοντα και να βελτιώνονται συνεχώς, υπερβαίνοντας την παραδοσιακή επίλυση προβλημάτων και ανοίγοντας νέους ορίζοντες σε διάφορες επιστημονικές και πρακτικές εφαρμογές.

Η σύγχρονη τεχνητή νοημοσύνη βασίζεται σε εξελιγμένα υπολογιστικά μοντέλα, όπως η μηχανική μάθηση (Machine Learning) και η βαθιά μάθηση (Deep Learning). Η μηχανική μάθηση επιτρέπει στα συστήματα να εντοπίζουν πρότυπα και να πραγματοποιούν προβλέψεις μέσω της ανάλυσης μεγάλων όγκων δεδομένων [1], ενώ η βαθιά μάθηση, μια υποκατηγορία της μηχανικής μάθησης, χρησιμοποιεί νευρωνικά δίκτυα για να αντιμετωπίσει πιο σύνθετα προβλήματα, όπως η αναγνώριση εικόνων και η επεξεργασία ομιλίας [2]. Αυτές οι μέθοδοι έχουν φέρει επανάσταση στον τρόπο που οι μηχανές κατανοούν και επεξεργάζονται πληροφορίες υψηλής πολυπλοκότητας, αποτελώντας τη βάση για εφαρμογές AI, όπως η επεξεργασία φυσικής γλώσσας (NLP). Το NLP επικεντρώνεται στη δυνατότητα των μηχανών να κατανοούν, να παράγουν και να αλληλεπιδρούν με την ανθρώπινη γλώσσα, υποστηρίζοντας τεχνολογίες όπως εικονικούς βοηθούς, εργαλεία μετάφρασης γλωσσών και συστήματα ανάλυσης συναισθημάτων [3].

Οι εφαρμογές του AI καλύπτουν ποικίλους τομείς, μετασχηματίζοντας τη βιομηχανία και την καθημερινή ζωή. Στην υγειονομική περίθαλψη, εργαλεία διάγνωσης που βασίζονται στο AI βελτιώνουν την ακρίβεια και την ταχύτητα, ενώ στις μεταφορές, τα αυτόνομα οχήματα αξιοποιούν δεδομένα σε πραγματικό χρόνο και προβλεπτικούς αλγορίθμους. Σύγχρονα εργαλεία AI, όπως τα TensorFlow, PyTorch και μοντέλα της OpenAI, όπως το ChatGPT, παρέχουν στους ερευνητές και τους προγραμματιστές τη δυνατότητα να σχεδιάζουν πολύπλοκα συστήματα με ευκολία [4]. Αυτές οι πλατφόρμες διευκολύνουν τη γρήγορη πρωτοτυποποίηση, την κλιμάκωση και την ενσωμάτωση στις υπάρχουσες τεχνολογίες.

Παρά τις επιτυχίες, το σύγχρονο AI εγείρει ηθικά ζητήματα σχετικά με την ιδιωτικότητα και τη διαφάνεια στη λήψη αποφάσεων, υπογραμμίζοντας την ανάγκη για υπεύθυνη ανάπτυξη και ρύθμιση [5].

1.2 Χρήση της τεχνητής νοημοσύνης στην τεχνολογία λογισμικού

Στην τεχνολογία λογισμικού, η τεχνητή νοημοσύνη ασκεί πλέον καθοριστική επιρροή, αξιοποιώντας τη δύναμή της στην ανάλυση δεδομένων για την ενίσχυση της ανάπτυξης και της βελτιστοποίησης λογισμικού [6]. Μέσω της αυτοματοποίησης εργασιών όπως η συγγραφή κώδικα, η αναδιάρθρωση (refactoring) και η ανίχνευση σφαλμάτων (debugging), βελτιώνει την αποδοτικότητα των προγραμματιστών και διασφαλίζει υψηλότερα πρότυπα ποιότητας. Παράλληλα, συμβάλλει στην αναβάθμιση της λειτουργίας των ομάδων ανάπτυξης, αναλύοντας δεδομένα απόδοσης και προτείνοντας στρατηγικές βελτίωσης. Παρ' όλα αυτά, η ενσωμάτωση αυτών των τεχνολογιών απαιτεί προσεκτική αξιολόγηση, καθώς οι περιορισμοί των υπαρχόντων μοντέλων AI, η ποιότητα των δεδομένων και οι ηθικές επιπτώσεις διαδραματίζουν κρίσιμο ρόλο στην επιτυχημένη εφαρμογή τους.

Η χρήση μεγάλων γλωσσικών μοντέλων στη τεχνολογία λογισμικού αναδεικνύει νέες δυνατότητες, όπως η αυτοματοποίηση της αρχιτεκτονικής σχεδίασης, η πρόβλεψη σημείων συμφόρησης και η επιτάχυνση των δοκιμών. Ωστόσο, τέτοιες εφαρμογές δεν είναι έτοιμες προς χρήση ("ready out of the box")· απαιτούν εξατομικευμένο σχεδιασμό, προσεκτική προσαρμογή στις ανάγκες του εκάστοτε έργου και συνεχή αξιολόγηση για την αποφυγή σφαλμάτων ή προκαταλήψεων. Η επιτυχής αξιοποίηση της τεχνητής νοημοσύνης στον κύκλο ζωής ανάπτυξης λογισμικού (Software Development Lifecycle - SDLC) προϋποθέτει την εξισορρόπηση της καινοτομίας με τις απαιτήσεις ακρίβειας, κλιμακωσιμότητας και ευθυγράμμισης με θεμελιώδεις αρχές σχεδίασης.

Συνοψίζοντας, η τεχνητή νοημοσύνη δεν αποσκοπεί προς το παρόν στην πλήρη αντικατάσταση της ανθρώπινης εξειδίκευσης αλλά στην ενίσχυσή της, προσφέροντας εργαλεία που ενισχύουν την παραγωγικότητα, προωθούν την καινοτομία και απλοποιούν την πολυπλοκότητα της σύγχρονης ανάπτυξης λογισμικού. Η αποτελεσματική ενσωμάτωσή της, ωστόσο, προϋποθέτει βαθιά κατανόηση των δυνατοτήτων και των περιορισμών της, καθώς και δέσμευση για υπεύθυνη χρήση. Το αναδυόμενο αυτό παράδειγμα δεν αφορά μόνο την υιοθέτηση τεχνολογιών αλλά και τη προσαρμοζόμενη σκέψη, προωθώντας μια πιο έξυπνη, ολιστική προσέγγιση στον σχεδιασμό, τη συντήρηση και την εξέλιξη των λογισμικών συστημάτων [7], [8].

Η ακόλουθη βιβλιογραφική αναφορά συγκεντρώνει πρόσφατες μελέτες ανά πεδίο, οι οποίες εξερευνούν τις εφαρμογές, τα πλεονεκτήματα αλλά και τους περιορισμούς της συνεισφοράς του AI στην τεχνολογία λογισμικού.

1.2.1 Δημιουργία και υποστήριξη στον κώδικα προγραμματισμού

Το παραγωγικό AI (generative AI), αξιοποιώντας αλγορίθμους μηχανικής μάθησης (ML) και βαθιάς μάθησης (DL), διαδραματίζει καθοριστικό ρόλο στον αυτοματισμό διαφόρων πτυχών της δημιουργίας κώδικα. Ο Tembhekar υπογραμμίζει ότι αυτές οι τεχνολογίες επιτρέπουν την αυτοματοποίηση εργασιών που παραδοσιακά εκτελούνταν από προγραμματιστές, ενισχύοντας την αποδοτικότητα στις πρακτικές του DevOps [9]. Η ενσωμάτωση τεχνικών επεξεργασίας φυσικής

γλώσσας έχει επιπλέον διευκολύνει την ανάπτυξη συστημάτων AI που κατανοούν και παράγουν κώδικα, απλοποιώντας τεχνικά κομμάτια της εργασίας.

Η συμπλήρωση κώδικα έχει εξελιχθεί σε έναν κρίσιμο τομέα έρευνας, με τα νευρωνικά γλωσσικά μοντέλα να παίζουν σημαντικό ρόλο στην αύξηση της παραγωγικότητας των προγραμματιστών. Ο Gao εξετάζει την εξέλιξη των στατιστικών γλωσσικών και νευρωνικών συστημάτων συμπλήρωσης κώδικα, υπογραμμίζοντας την αποτελεσματικότητά τους στη βελτίωση της αποδοτικότητας κατά τον προγραμματισμό [10]. Παρόλα αυτά η συζήτηση γύρω από τις ηθικές προεκτάσεις της χρήσης του AI στη δημιουργία κώδικα καθίσταται επιτακτική. Σύμφωνα με τον Atemkeng et al. , η υπεύθυνη χρήση εργαλείων παραγωγικού AI αποτελεί αντικείμενο ανησυχίας για την επιστημονική κοινότητα [11]. Η απαίτηση από το AI να παράγει κώδικα που είναι ταυτόχρονα λειτουργικός και ασφαλής αναδεικνύει την ανάγκη για συνεχή συζήτηση σχετικά με τη ρύθμιση και την εποπτεία αυτών των τεχνολογιών.

Συνοψίζοντας η ενσωμάτωση του AI στη παραγωγή κώδικα συνιστά μια σημαντική πρόοδο στην τεχνολογία λογισμικού, καθοδηγούμενη από τεχνικές βαθιάς μάθησης και επεξεργασίας φυσικής γλώσσας. Η συνεχιζόμενη έρευνα στον τομέα αυτό συνεχίζει να εξερευνά την ισορροπία μεταξύ αυτοματοποίησης και των ηθικών παραμέτρων της χρήσης AI, διασφαλίζοντας ότι τα οφέλη αυτών των τεχνολογιών αξιοποιούνται με υπευθυνότητα.

1.2.2 Αυτοματοποιημένος Έλεγχος

Η αξιοποίηση της τεχνητής νοημοσύνης στον αυτοματοποιημένο έλεγχο λογισμικού αναδεικνύεται ως μια εξαιρετικά αποτελεσματική μέθοδος για τη βελτίωση της ποιότητας του λογισμικού και τη μείωση του χρόνου που απαιτείται για τη διαδικασία των δοκιμών. Η Mulla τονίζει ότι οι τεχνικές αυτοματοποίησης με τη χρήση AI επιτρέπουν την άμεση εκτέλεση εκτεταμένων σειρών ελέγχου μετά από κάθε αλλαγή στο λογισμικό, υποστηρίζοντας έτσι την πρακτική της συνεχούς ενσωμάτωσης [12]. Αυτό είναι ιδιαίτερα σημαντικό σε σύγχρονα περιβάλλοντα ανάπτυξης λογισμικού, όπου οι γρήγορες αναθεωρήσεις αποτελούν αναπόσπαστο μέρος της διαδικασίας. Παράλληλα, ο Job σημειώνει ότι η αυξανόμενη πολυπλοκότητα των συστημάτων λογισμικού καθιστά την αυτοματοποίηση στον έλεγχο αναγκαία για τη διασφάλιση των ποιοτικών προτύπων μέσα σε στενά χρονικά πλαίσια [13]

Η ενσωμάτωση της μηχανικής μάθησης στις αυτοματοποιημένες δοκιμές έχει μελετηθεί εκτενώς. Οι Gautam et al. παρέχουν μια ολοκληρωμένη επισκόπηση του τρόπου με τον οποίο οι αλγόριθμοι ML μπορούν να αυτοματοποιήσουν τη διαδικασία ανίχνευσης σφαλμάτων, αυξάνοντας την αξιοπιστία των συστημάτων λογισμικού [14].

Παρά τα οφέλη, η εφαρμογή του AI στις αυτοματοποιημένες δοκιμές αντιμετωπίζει σημαντικές προκλήσεις. Η Marijan επισημαίνει την ανάγκη για εξειδικευμένες μεθοδολογίες δοκιμών που λαμβάνουν υπόψη τα μοναδικά χαρακτηριστικά των συστημάτων AI [15]. Η εργασία της προτείνει ένα ερευνητικό πλαίσιο για την ανάπτυξη αποτελεσματικών τεχνικών δοκιμών για εφαρμογές μηχανικής μάθησης. Παρομοίως, η μελέτη της Latika Kharb εξετάζει τη σημασία της εμπιστοσύνης και της διαφάνειας στα συστήματα AI, υπογραμμίζοντας την ανάγκη για πιο διαφανείς διαδικασίες αυτοματοποιημένων δοκιμών [16].

Η βιβλιογραφία λοιπόν μας δείχνει ότι το AI έχει τη δυνατότητα να επαναπροσδιορίσει τον αυτοματοποιημένο έλεγχο λογισμικού, βελτιώνοντας την αποδοτικότητα, την ακρίβεια και την αξιοπιστία. Ωστόσο, οι προκλήσεις που σχετίζονται με τη δοκιμή συστημάτων AI απαιτούν

περαιτέρω έρευνα και ανάπτυξη για τη δημιουργία ισχυρών μεθοδολογιών που να ανταποκρίνονται στις ιδιαίτερες απαιτήσεις του πεδίου.

1.2.3 Επεξεργασία Φυσικής Γλώσσας για Απαιτήσεις και Σχεδιασμό

Η Επεξεργασία Φυσικής Γλώσσας αποτελεί βασική τεχνολογία στον τομέα της τεχνολογίας λογισμικού, ιδίως όσον αφορά την ανάλυση και τον σχεδιασμό των απαιτήσεων. Συμβάλλει σημαντικά στη διαμόρφωση, την αποτύπωση και την ανάλυση των απαιτήσεων, καθιστώντας αυτές τις διαδικασίες πιο ακριβείς και αποδοτικές.

Μια συστηματική βιβλιογραφική ανασκόπηση από τους Calle και Zapata εισάγει το QUARE (Question Answering for Requirements Elicitation), ένα μοντέλο ερωτήσεων-απαντήσεων για τη βελτίωση της διαδικασίας εξαγωγής των απαιτήσεων επεκτείνοντας την ιδέα του NLP4RE (Natural Language Processing for Requirements Engineering) [17]. Αυτό το μοντέλο έχει στόχο την διευκόλυνση των αναλυτών λογισμικού στην εξαγωγή απαιτήσεων (Requirements Elication - RE) απαντώντας σε RE-related ερωτήσεις χρησιμοποιώντας τα έγγραφα απαιτήσεων. Επιπλέον, η ενσωμάτωση τεχνικών μηχανικής μάθησης στη διαμόρφωση απαιτήσεων έχει αποδειχθεί αποτελεσματική στην αυτοματοποίηση και την απλοποίηση των παραδοσιακά χρονοβόρων διαδικασιών, αυξάνοντας την αποδοτικότητα [18], [19] .

Ένας άλλος κρίσιμος ρόλος του NLP είναι η μετατροπή των απαιτήσεων που περιγράφονται σε φυσική γλώσσα σε τυπικές αναπαραστάσεις. Τεχνικές σημασιολογικής ανάλυσης (semantic parsing) έχουν χρησιμοποιηθεί για να προσδώσουν σημασιολογική δομή στις απαιτήσεις, επιτρέποντας την καλύτερη κατανόηση των προδιαγραφών λειτουργικότητας [20]. Αυτή η μετατροπή είναι απαραίτητη για να διασφαλιστεί ότι οι απαιτήσεις είναι κατανοητές και μπορούν να αξιοποιηθούν άμεσα στα επόμενα στάδια του κύκλου ζωής του λογισμικού.

Παρά τις προόδους, υπάρχουν ακόμα σημαντικές προκλήσεις στη χρήση του NLP στο σχεδιασμό απαιτήσεων. Το γεγονός ότι η πλειονότητα ,περίπου 95%, των απαιτήσεων λογισμικού εξακολουθεί να περιγράφεται σε φυσική γλώσσα μπορεί να οδηγήσει σε παρεξηγήσεις και σφάλματα [21] . Για το λόγο αυτό, είναι αναγκαία η περαιτέρω βελτίωση των τεχνικών NLP, ώστε να αυξηθεί η ποιότητα και η αξιοπιστία της τεκμηρίωσης των απαιτήσεων.

Η ενσωμάτωση τεχνικών NLP και ML στη μηχανική απαιτήσεων μπορούμε με ασφάλεια λοιπόν να ισχυριστούμε ότι προσφέρει σημαντικές δυνατότητες για τη βελτίωση της σαφήνειας, της ακρίβειας και της αποδοτικότητας στις διαδικασίες ανάλυσης και σχεδιασμού. Ωστόσο παρότι έχει σημειωθεί σημαντική πρόοδος, οι υφιστάμενες προκλήσεις καθιστούν απαραίτητη τη συνεχή έρευνα και ανάπτυξη για την πλήρη αξιοποίηση αυτών των τεχνολογιών στον τομέα.

Συμπερασματικά, η βιβλιογραφία αναδεικνύει ότι η τεχνητή νοημοσύνη και η μηχανική μάθηση έχουν τη δυναμική να μετασχηματίσουν ριζικά την τεχνολογία λογισμικού, προσφέροντας αυξημένη αποδοτικότητα, βελτιωμένη ποιότητα και εκτεταμένη αυτοματοποίηση διαδικασιών. Ωστόσο, η επιτυχία αυτής της ενσωμάτωσης εξαρτάται από την προσεκτική ισορροπία μεταξύ της ανθρώπινης τεχνογνωσίας και των δυνατοτήτων των αλγορίθμων, καθώς και από την προσαρμογή των υφιστάμενων πρακτικών, ώστε να αξιοποιηθεί πλήρως το δυναμικό αυτών των τεχνολογιών.

1.3 Προκλήσεις της τεχνητής νοημοσύνης στη τεχνολογία λογισμικού

Η ενσωμάτωση της τεχνητής νοημοσύνης στην τεχνολογία λογισμικού συνοδεύεται από μια σειρά τεχνικών προκλήσεων που πρέπει να αντιμετωπιστούν για την πλήρη αξιοποίηση των δυνατοτήτων της. Ένα βασικό ζήτημα είναι η πολυπλοκότητα της επαλήθευσης και επικύρωσης συστημάτων που βασίζονται στο AI. Οι παραδοσιακές τεχνικές επαλήθευσης λογισμικού συχνά αποδεικνύονται ανεπαρκείς στο πλαίσιο του AI λόγω της περίπλοκης φύσης των αλγορίθμων του και της εξάρτησής τους από μεγάλα σύνολα δεδομένων, τα οποία μπορεί να εισαγάγουν προκαταλήψεις και αβεβαιότητες [22].

Ένα άλλο κρίσιμο ζήτημα αφορά τη διασφάλιση ποιότητας (Quality Assurance) στο λογισμικό που βασίζεται στο AI. Η ενσωμάτωση τεχνολογιών AI δημιουργεί ανησυχίες σχετικά με την ποιότητα των δεδομένων, τη διαφάνεια των αλγορίθμων και τις ηθικές επιπτώσεις. Οι υπεύθυνοι επαγγελματίες για τη διασφάλιση ποιότητας αντιμετωπίζουν προκλήσεις στον έλεγχο συστημάτων AI, καθώς οι παραδοσιακές μεθοδολογίες δεν επαρκούν για να αντιμετωπίσουν τα μοναδικά χαρακτηριστικά του, όπως η μάθηση από δεδομένα και η διαρκής εξέλιξή του [23]. Επιπλέον, η πιθανότητα προκατειλημμένων αποτελεσμάτων λόγω μεροληπτικών δεδομένων εκπαίδευσης απαιτεί αυστηρό έλεγχο των δεδομένων που χρησιμοποιούνται στις εφαρμογές AI [23]. Η ανάγκη για επεξηγησιμότητα είναι επίσης κομβική, καθώς οι ενδιαφερόμενοι (stakeholders) απαιτούν διαφάνεια στις αποφάσεις που λαμβάνονται από τους αλγόριθμους AI για τη διασφάλιση της λογοδοσίας και της εμπιστοσύνης [24].

Η **αρχιτεκτονική λογισμικού** αποτελεί αναμφίβολα ένα θεμελιώδη συστατικό του κύκλου ζωής ανάπτυξης λογισμικού. Ωστόσο, το ζήτημα της δυνατότητας αυτοματοποίησης διαφόρων πτυχών της μέσω τεχνητής νοημοσύνης παραμένει ακόμα αντικείμενο ενδελεχούς διερεύνησης. Από τη δημιουργία αρχιτεκτονικών μοντέλων έως τον έγκαιρο εντοπισμό σχεδιαστικών σφαλμάτων, το AI θα μπορούσε να διαδραματίσει καθοριστικό ρόλο.

Ωστόσο, αυτή η διαδικασία συνοδεύεται από σημαντικές προκλήσεις:

1. **Ασάφεια και Πολυπλοκότητα στη Φυσική Γλώσσα**
   Οι απαιτήσεις συνήθως περιγράφονται σε φυσική γλώσσα, η οποία συχνά χαρακτηρίζεται από ασάφεια και έλλειψη ακρίβειας, που είναι κρίσιμες για αποφάσεις αρχιτεκτονικής όπως ο σχεδιασμός κλάσεων. Τα περισσότερα εργαλεία απαιτούν συνεχή παρέμβαση και αλληλεπίδραση του χρήστη για την εξαγωγή διαγραμμάτων UML, ενώ οι υφιστάμενες λύσεις δεν μπορούν να παράγουν αυτόματα όλα τα στοιχεία και τις σημασιολογικές λεπτομέρειες των διαγραμμάτων, όπως τα ονόματα των κλάσεων, τις λειτουργίες και τις σχέσεις, π.χ., γενίκευση, συσχέτιση και εξάρτηση [25].

2. **Δυσκολία στην Εξαγωγή Σχέσεων και Ιεραρχικών Δομών**
   Ένα από τα βασικά ζητήματα στη δημιουργία διαγραμμάτων κλάσεων είναι η ακριβής αναπαράσταση σχέσεων όπως η κληρονομικότητα, η συσχέτιση, η σύνθεση και η συγκέντρωση. Τα έγγραφα απαιτήσεων σπάνια παρέχουν σαφείς λεπτομέρειες για αυτές τις σχέσεις, απαιτώντας αφαιρετική σκέψη και ιεραρχική λογική, χαρακτηριστικά τα οποία τα LLMs δεν είναι εγγενώς σχεδιασμένα να εκτελούν [26].

3. **Έλλειψη Εξειδικευμένης Εκπαίδευσης σε Τυπικά Μοντέλα Λογισμικού**
   Τα περισσότερα LLMs εκπαιδεύονται σε μεγάλα σύνολα γενικού κειμένου (π.χ., Wikipedia, αποθετήρια κώδικα), τα οποία περιλαμβάνουν περιορισμένα δεδομένα σχετικά με σχεδιαστικά πρότυπα λογισμικού όπως η γλώσσα UML. Αυτό οδηγεί σε περιορισμούς όταν πρόκειται για την

παραγωγή διαγραμμάτων που πρέπει να συμμορφώνονται με τις συμβάσεις των τυπικών μοντέλων λογισμικού.

4. **Ανάγκη για Συμφραζόμενη Γνώση και Σχεδιαστικά Πρότυπα**
   Η μετάφραση περιγραφών από φυσική γλώσσα σε διαγράμματα κλάσεων απαιτεί γνώση του αντικειμένου και των σχεδιαστικών προτύπων. Τα LLMs συνήθως δεν είναι εγγενώς εξοπλισμένα για να εφαρμόσουν τέτοια συμφραζόμενη γνώση, σε εξειδικευμένα σενάρια που απαιτούν σαφώς δομημένα πρότυπα όπως MVC ή Singleton, τα οποία μπορεί να είναι κρίσιμα για την αρχιτεκτονική σχεδίαση. Χωρίς αυτή την ικανότητα τα μοντέλα συχνά τείνουν να δημιουργούν απλουστευμένα διαγράμματα χωρίς έμφαση στη λεπτομέρεια και με λανθασμένες σχέσεις μεταξύ των κλάσεων. Επιπλέον το υφιστάμενο υλικό συχνά περιέχει πλουραλισμό και συγκρούσεις σχετικά με την εφαρμογή του εκάστοτε αρχιτεκτονικού προτύπου, γεγονός που προσθέτει επιπλέον δυσκολίες στο συγκεκριμένο τομέα. Οι Vaidhyanathan et al. διερευνούν αν τα μεγάλα γλωσσικά μοντέλα (LLMs) μπορούν να δημιουργήσουν αποτελεσματικά αποφάσεις αρχιτεκτονικού σχεδιασμού σε μια zero-shot προσέγγιση. Τα ευρήματά τους υποδεικνύουν ότι, ενώ τα LLMs μπορούν να συμβάλλουν στη δημιουργία αποφάσεων σχεδιασμού, δεν είναι ικανά να το κάνουν αυτό αυτόνομα. Αντίθετα, αυτά τα μοντέλα είναι πιο κατάλληλα για να υποστηρίζουν τους αρχιτέκτονες λογισμικού στην τεκμηρίωση και τη βελτίωση των αποφάσεων σχεδιασμού[27] .

Με αφορμή τις παραπάνω προκλήσεις, αποφασίσαμε να εξετάσουμε σε μεγαλύτερο βάθος την σχέση τεχνητής νοημοσύνης και αρχιτεκτονικής λογισμικού. Στις επόμενες ενότητες θα αναλύσουμε τη μεθοδολογία μας, θα παρουσιάσουμε τα αποτελέσματα της αξιολόγησής μας για την απόδοση των LLMs στην αυτοματοποίηση της αρχιτεκτονικής λογισμικού και θα συζητήσουμε τις μελλοντικές προκλήσεις που προκύπτουν από τα ευρήματά μας.

2 Αρχιτεκτονική λογισμικού με χρήση AI

Αυτό το κεφάλαιο εξετάζει τη συναφή βιβλιογραφία στον ταχέως εξελισσόμενο τομέα της AI-υποβοηθούμενης αρχιτεκτονικής λογισμικού, παρουσιάζει τεκμηρίωση για βασικά αρχιτεκτονικά πρότυπα και περιγράφει τη διάκριση και το κίνητρο της προσέγγισής μας σε σχέση με την υπάρχουσα βιβλιογραφία.

2.1 Σχετικές μελέτες για τη χρήση του AI στην αρχιτεκτονική λογισμικού

Η αυτόματη δημιουργία διαγραμμάτων κλάσεων UML (Unified Modeling Language) από κειμενικές απαιτήσεις έχει αποκτήσει σημαντική δυναμική τα τελευταία χρόνια, χάρη στην ανάπτυξη μεγάλων γλωσσικών μοντέλων. Το αναδυόμενο αυτό θέμα εστιάζει στις δυνατότητες προηγμένων συστημάτων τεχνητής νοημοσύνης να αντιμετωπίσουν διαχρονικές προκλήσεις στην αρχιτεκτονική λογισμικού, όπως το να γεφυρώσουν το χάσμα μεταξύ φυσικής γλώσσας και δομικών στοιχείων σχεδιασμού συστημάτων.

Παραδοσιακές προσεγγίσεις για τη δημιουργία διαγραμμάτων κλάσεων UML βασίζονται συχνά σε τεχνικές επεξεργασίας φυσικής γλώσσας συνδυασμένες με ευρετικούς κανόνες ή οντολογίες. Παρότι αυτές οι μέθοδοι έδειξαν την κατεύθυνση για τη δυνατότητα αυτοματοποίησης, περιορίζονται από την εξάρτησή τους σε δομημένες μορφές εισόδου και την ανάγκη για χειροκίνητη παρέμβαση. Για παράδειγμα, προηγούμενα εργαλεία δυσκολεύονταν να διαχειριστούν τις ασάφειες της φυσικής γλώσσας και συχνά παρήγαγαν ελλιπή διαγράμματα.

Οι **Eisenreich, Speth, και Wagner** προτείνουν μια δομημένη διαδικασία έξι σταδίων για τη γεφύρωση των κειμενικών απαιτήσεων με τη δημιουργία αρχιτεκτονικής λογισμικού [28]. Η διαδικασία ξεκινά με την αυτόματη δημιουργία domain model και σεναρίων χρήσης βάσει των κειμενικών απαιτήσεων, ακολουθούμενη από χειροκίνητη βελτίωση για την αύξηση της ακρίβειας και της συνάφειας. Στη συνέχεια, χρησιμοποιώντας το domain model, τα σενάρια και τις μη λειτουργικές απαιτήσεις, παράγονται αυτόματα πολλαπλές υποψήφιες αρχιτεκτονικές μαζί με τις αντίστοιχες αρχιτεκτονικές αποφάσεις. Οι αρχιτεκτονικές αυτές υποβάλλονται σε αυτόματη αξιολόγηση και σύγκριση. Έπειτα, πραγματοποιούνται χειροκίνητες βελτιώσεις στις επιλεγμένες υποψήφιες αρχιτεκτονικές, καταλήγοντας στην τελική επιλογή της καταλληλότερης αρχιτεκτονικής για υλοποίηση. Στην εξερευνητική τους ανάλυση, οι συγγραφείς χρησιμοποίησαν προηγμένα LLMs όπως το **LLaMA2 70-B** και το **GPT-3.5** για τη δημιουργία domain models από κειμενικές απαιτήσεις. Παρέχοντας στα μοντέλα τα σύνολα απαιτήσεων και ζητώντας τους να παράγουν domain model σε PlantUML, παρατήρησαν ότι, ενώ και τα δύο LLMs αναγνώρισαν βασικές έννοιες, δυσκολεύτηκαν να ευθυγραμμιστούν με τις συγκεκριμένες απαιτήσεις των εργασιών, συχνά παρερμηνεύοντας τα προτροπές (prompts). Αντί να μοντελοποιούν το domain, όπως αναμενόταν, τα LLMs επικεντρώνονταν στη μοντελοποίηση του ίδιου του συστήματος, αναδεικνύοντας την αναντιστοιχία μεταξύ των προτροπών και των παραγόμενων αποτελεσμάτων.

Μια άλλη προσπάθεια για την αντιμετώπιση των προκλήσεων ασάφειας στη φυσική γλώσσα παρουσιάστηκε από τους **Yang και Sahraoui**, με μια προσέγγιση βασισμένη στο AI για τη δημιουργία διαγραμμάτων κλάσεων UML [29]. Η μέθοδός τους χρησιμοποιεί έναν δυαδικό

ταξινομητή (classifier), ο οποίος μέσω μηχανικής μάθησης χαρακτηρίζει προτάσεις είτε ως περιγραφές κλάσεων είτε ως περιγραφές σχέσεων. Η διαδικασία περιλαμβάνει τη διάσπαση αγγλικών κειμένων σε μεμονωμένες προτάσεις, τη δημιουργία δομικών στοιχείων διαγραμμάτων UML από αυτές και, τέλος, τη σύνθεση των στοιχείων αυτών σε ένα συνολικό διάγραμμα. Για την υποστήριξη αυτής της μεθόδου, οι ερευνητές δημιούργησαν ένα σύνολο δεδομένων διαγραμμάτων UML συνδεδεμένων με αντίστοιχες αγγλικές προδιαγραφές, όπου οι περιγραφές κειμένου αντιστοιχούνταν σε δομικά στοιχείων UML. Παρότι το σύνολο δεδομένων δημιουργήθηκε μέσω crowdsourcing και ήταν σχετικά μικρό, ήταν επαρκές για την εκπαίδευση του ταξινομητή και την αξιολόγηση της μεθοδολογίας τους. Παρά τον καινοτόμο χαρακτήρα της προσέγγισης, η ορθότητα των παραγόμενων διαγραμμάτων ήταν περιορισμένη, γεγονός που οι συγγραφείς απέδωσαν στην ανακρίβεια των εργαλείων NLP που χρησιμοποιήθηκαν. Υπογράμμισαν ότι η αξιοποίηση πιο προηγμένων εργαλείων NLP θα μπορούσε να βελτιώσει σημαντικά την ακρίβεια και την ανθεκτικότητα των αποτελεσμάτων, τονίζοντας τη δυνατότητα περαιτέρω βελτίωσης.

Μια ακόμα προσέγγιση που αξιοποιεί τις δυνατότητες επεξεργασίας φυσικής γλώσσας του **GPT-3.5** για την αυτοματοποίηση της εξαγωγής στοιχείων διαγραμμάτων κλάσεων UML παρουσιάστηκε από τους **Iyad και Areen** μέσω του εργαλείου **ClassDiagGen** [30]. Το ClassDiagGen αντιπροσωπεύει μια σημαντική πρόοδο, βελτιώνοντας τις δυνατότητες του GPT-3.5 μέσω της τεχνικής της εκπαίδευσης με παραδείγματα (**few-shot learning**) για την αυτοματοποίηση της δημιουργίας διαγραμμάτων UML. Με την εκπαίδευση του GPT-3.5 σε ένα σύνολο δεδομένων 50 διαφορετικών περιπτώσεων (ζεύγη προτροπών και ιδανικών παραγόμενων κειμένων), το ClassDiagGen παρουσίασε εξαιρετική ακρίβεια (**98,6%**) και ανάκληση (**93,3%**), υπερέχοντας σημαντικά σε σχέση με προηγούμενες μεθόδους. Η ροή εργασίας του συνδυάζει ανάλυση κειμένου, δημιουργία κώδικα **PlantUML** και απόδοση διαγραμμάτων σε μια συνεκτική διαδικασία, αναδεικνύοντας τις πρακτικές δυνατότητες των LLMs στη βελτιστοποίηση των διαδικασιών σχεδιασμού λογισμικού.

2.2 Τεκμηρίωση αρχιτεκτονικών προτύπων

Αυτή η ενότητα παρέχει μια περιγραφή των βασικών αρχιτεκτονικών προτύπων, αναλύοντας τη δομή, τα συστατικά στοιχεία και την εφαρμογή τους στον σχεδιασμό λογισμικού. Στόχος της είναι να παρουσιάσει αυτά τα πρότυπα με σαφήνεια και ακρίβεια, λειτουργώντας ως σημείο αναφοράς για την κατανόηση του ρόλου και της υλοποίησής τους στον αρχιτεκτονικό σχεδιασμό.

2.2.1 Αρχιτεκτονική Πελάτη-Εξυπηρετητή (Client-Server)

Η αρχιτεκτονική πελάτη-εξυπηρετητή αποτελεί ένα από τα πλέον διαδεδομένα μοντέλα στα δίκτυα υπολογιστών, όπου πολλαπλοί πελάτες (απομακρυσμένοι επεξεργαστές) αλληλεπιδρούν με έναν κεντρικό εξυπηρετητή για να αιτηθούν και να λάβουν υπηρεσίες. Αυτή η δομή διαχωρίζει τους ρόλους των παρόχων υπηρεσιών και των καταναλωτών υπηρεσιών σε ένα δικτυακό περιβάλλον.

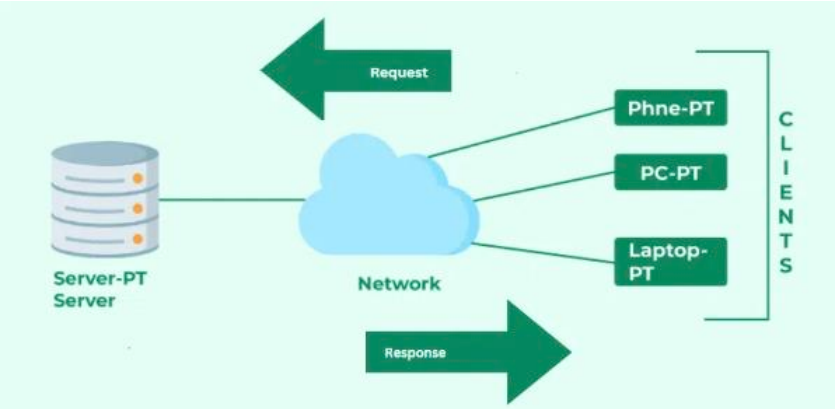
Στο μοντέλο πελάτη-εξυπηρετητή, το σύστημα οργανώνεται γύρω από διάφορες υπηρεσίες που φιλοξενούνται σε εξειδικευμένους εξυπηρετητές, ενώ οι πελάτες έχουν πρόσβαση και αξιοποιούν αυτές τις υπηρεσίες [31]. Τα κύρια στοιχεία αυτής της αρχιτεκτονικής περιλαμβάνουν:

- **Εξυπηρετητές (Servers):** Παρέχουν υπηρεσίες σε άλλα μέρη του συστήματος. Για παράδειγμα, εξυπηρετητές εκτύπωσης χειρίζονται αιτήματα εκτύπωσης, εξυπηρετητές αρχείων διαχειρίζονται

την αποθήκευση και ανάκτηση αρχείων, ενώ εξυπηρετητές μεταγλώττισης εκτελούν εργασίες μεταγλώττισης γλωσσών προγραμματισμού.

- **Πελάτες (Clients):** Αιτούνται υπηρεσίες από τους εξυπηρετητές. Συνήθως, πολλαπλά αντίγραφα εφαρμογών πελατών λειτουργούν ταυτόχρονα σε διαφορετικές συσκευές.

- **Δικτυακή Υποδομή:** Επιτρέπει στους πελάτες να συνδέονται με τις υπηρεσίες. Οι περισσότερες αρχιτεκτονικές πελάτη-εξυπηρετητή λειτουργούν ως κατανεμημένα συστήματα, αξιοποιώντας πρωτόκολλα Διαδικτύου για τη διευκόλυνση της συνδεσιμότητας.

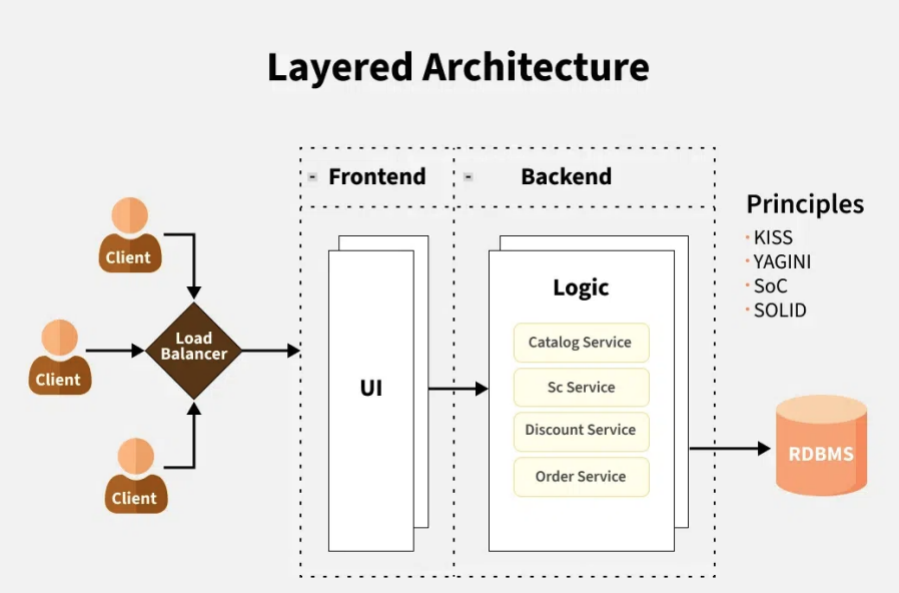


Εικόνα 2.1 Μοντέλο πελάτη-εξυπηρετητή (πηγή)

2.2.2 Αρχιτεκτονική στρωμάτων (layered architecture)

Στην αρχιτεκτονική στρωμάτων, η λειτουργικότητα του συστήματος διαιρείται σε διακριτά επίπεδα, με κάθε επίπεδο να εξαρτάται αποκλειστικά από τις υπηρεσίες που παρέχει το αμέσως κατώτερο επίπεδο. Αυτή η δομή υποστηρίζει την σταδιακή ανάπτυξη του συστήματος, καθιστώντας διαθέσιμες στους χρήστες τις υπηρεσίες κάθε επιπέδου μόλις ολοκληρωθούν. Το σχέδιο αυτό είναι ιδιαίτερα ευέλικτο, καθώς εφόσον η διεπαφή παραμένει η ίδια, οποιοδήποτε επίπεδο μπορεί να αντικατασταθεί με άλλο που προσφέρει παρόμοια λειτουργικότητα. Επιπλέον, οι αλλαγές ή οι νέες λειτουργίες που εισάγονται σε ένα επίπεδο επηρεάζουν μόνο το αμέσως συνδεδεμένο επίπεδο, διατηρώντας τη σταθερότητα του υπόλοιπου συστήματος. Με τον περιορισμό των εξαρτήσεων από βαθύτερα επίπεδα, αυτή η αρχιτεκτονική διευκολύνει τη συμβατότητα μεταξύ διαφορετικών πλατφορμών, καθώς μόνο τα εσωτερικά επίπεδα χρειάζονται προσαρμογές για να λειτουργήσουν με διαφορετικά λειτουργικά συστήματα ή βάσεις δεδομένων [31].

Συνοπτικά, το αρχιτεκτονικό πρότυπο των στρωμάτων οργανώνει το σύστημα σε ιεραρχία επιπέδων, το καθένα με σχετική λειτουργικότητα. Κάθε επίπεδο παρέχει υπηρεσίες στο ανώτερο, ενώ τα κατώτερα επίπεδα προσφέρουν βασικές υπηρεσίες που υποστηρίζουν ολόκληρο το σύστημα.



Εικόνα 2.2 Αρχιτεκτονική στρωμάτων (πηγή)

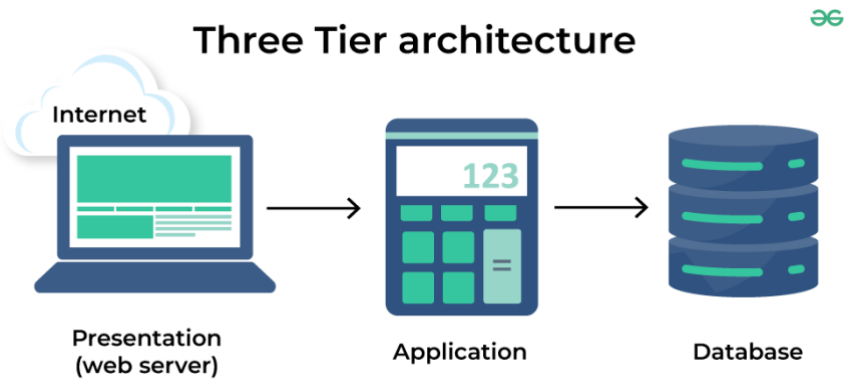
2.2.3 Αρχιτεκτονική Τριών Επιπέδων (Three-Tier Architecture)

Η αρχιτεκτονική τριών επιπέδων συνιστά υποκατηγορία της αρχιτεκτονικής στρωμάτων και αποτελείται από τρία διακριτά επίπεδα :

- **Επίπεδο Παρουσίασης (Presentation Layer):** Περιλαμβάνει τις φόρμες ή τις σελίδες του εξυπηρετητή που παρέχουν το περιβάλλον χρήστη της εφαρμογής. Σε αυτό το επίπεδο, εμφανίζονται τα δεδομένα, συλλέγονται οι είσοδοι του χρήστη και προωθούνται αιτήματα στο επόμενο επίπεδο.

- **Επίπεδο Επιχειρησιακής Λογικής (Business Layer):** Αντιμετωπίζει τα αιτήματα από το επίπεδο παρουσίασης, εφαρμόζει επιχειρησιακούς κανόνες στα δεδομένα και προωθεί αιτήματα στο επίπεδο δεδομένων. Περιλαμβάνει τη λογική που διέπει τη λειτουργία της εφαρμογής.

- **Επίπεδο Δεδομένων (Data Layer):** Περιλαμβάνει τη λογική πρόσβασης στα δεδομένα, τους οδηγούς βάσεων δεδομένων και τους μηχανισμούς ερωτημάτων, επιτρέποντας την άμεση αλληλεπίδραση με τη βάση δεδομένων για την αποθήκευση ή την ανάκτηση δεδομένων. [32]



Εικόνα 2.3 Αρχιτεκτονική τριών επιπέδων (πηγή)

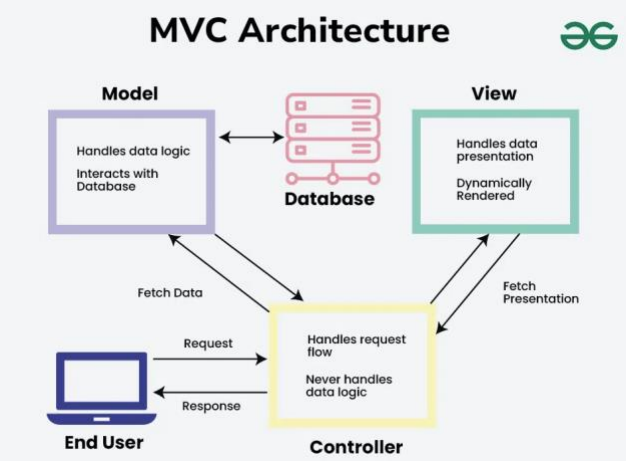
2.2.4 Αρχιτεκτονική Model-View-Controller (MVC)

Το πρότυπο Model-View-Controller (MVC) διαχωρίζει το σύστημα σε τρία μέρη:

- **Model (Μοντέλο):** Αντιπροσωπεύει τα δεδομένα της εφαρμογής και τη σχετική επιχειρησιακή λογική. Μπορεί να αποτελείται από ένα αντικείμενο ή από ένα δίκτυο σχετιζόμενων αντικειμένων. Σε εφαρμογές Java EE, τα δεδομένα αποθηκεύονται σε domain objects, ενώ για την επικοινωνία με τη βάση δεδομένων χρησιμοποιούνται αντικείμενα μεταφοράς δεδομένων (Data Transfer Objects - DTOs) και αντικείμενα πρόσβασης δεδομένων (Data Access Objects - DAOs).

- **View (Προβολή):** Παρέχει μια οπτική απεικόνιση των δεδομένων του μοντέλου. Κάθε προβολή παρουσιάζει ένα συγκεκριμένο υποσύνολο του μοντέλου, επιτρέποντας στους χρήστες να αλληλεπιδρούν με τα δεδομένα και να πράττουν ενεργοποιώντας την επιχειρησιακή λογική.

- **Controller (Ελεγκτής):** Συνδέει την προβολή με το μοντέλο και διαχειρίζεται τη ροή της εφαρμογής. Με βάση την εισαγωγή του χρήστη, καθορίζει ποια προβολή θα εμφανιστεί και ποια επιχειρησιακή λογική θα εκτελεστεί. Ο ελεγκτής λαμβάνει την είσοδο από το View, το προωθεί στο μοντέλο για επεξεργασία και στη συνέχεια επιλέγει το κατάλληλο View που θα παρουσιαστεί στο χρήστη. [33]



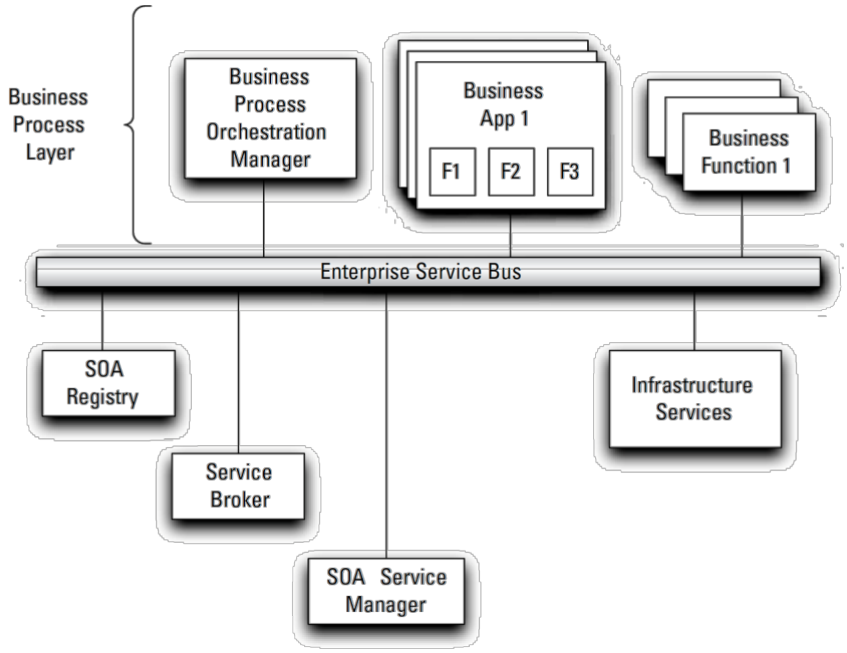
Εικόνα 2.4 Αρχιτεκτονική MVC (πηγή)

2.2.5 Αρχιτεκτονική Προσανατολισμένη στις Υπηρεσίες (SOA)

Η Αρχιτεκτονική Προσανατολισμένη στις Υπηρεσίες (Service-Oriented Architecture - SOA) είναι ένα αρχιτεκτονικό πρότυπο όπου τα λογισμικά στοιχεία, γνωστά ως υπηρεσίες, σχεδιάζονται ώστε να παρέχουν συγκεκριμένες, επαναχρησιμοποιήσιμες λειτουργίες. Κάθε υπηρεσία μέσα σε ένα σύστημα SOA ενσωματώνει μια ξεχωριστή επιχειρησιακή λειτουργία και αλληλεπιδρά με άλλες υπηρεσίες μέσω τυποποιημένων διεπαφών και πρωτοκόλλων, συνήθως μέσω ανταλλαγής μηνυμάτων σε ένα δίκτυο. Το SOA δίνει έμφαση στη χαλαρή σύζευξη μεταξύ υπηρεσιών, επιτρέποντας την ανεξάρτητη λειτουργία κάθε υπηρεσίας καθώς και τη δυνατότητα τροποποίησης, αντικατάστασης ή αναβάθμισής της χωρίς να επηρεάζονται οι υπόλοιπες υπηρεσίες στο σύστημα [34].

Το SOA επιτρέπει την ομαλή ενσωμάτωση διαφορετικών εφαρμογών, καθώς οι υπηρεσίες είναι τεχνολογικά ανεξάρτητες και επικοινωνούν χρησιμοποιώντας τυποποιημένα πρωτόκολλα, όπως SOAP ή REST. Αυτή η ευελιξία δίνει τη δυνατότητα ανάπτυξης και διάθεσης υπηρεσιών σε διαφορετικές πλατφόρμες και γλώσσες, καθιστώντας το ιδανικό για μεγάλα, σύνθετα συστήματα που απαιτούν διαλειτουργικότητα μεταξύ διαφόρων εφαρμογών και πηγών δεδομένων. Επιπλέον, το SOA διευκολύνει την κλιμάκωση, καθώς οι υπηρεσίες μπορούν να αναπτυχθούν σε πολλαπλούς διακομιστές και να κλιμακωθούν ανάλογα με τη ζήτηση.

Ο αρθρωτός σχεδιασμός του SOA βελτιώνει επίσης τη συντηρησιμότητα των συστημάτων, καθώς οι αλλαγές σε μια υπηρεσία δεν απαιτούν αλλαγές στις υπόλοιπες. Ευθυγραμμίζοντας τις υπηρεσίες με τις επιχειρησιακές διαδικασίες, το SOA επιτρέπει στους οργανισμούς να απλοποιούν τις λειτουργίες τους, να ανταποκρίνονται γρήγορα σε μεταβαλλόμενες απαιτήσεις και να επαναχρησιμοποιούν βασικές υπηρεσίες σε διάφορες εφαρμογές. Ως εκ τούτου, το SOA υιοθετείται ευρέως σε επιχειρησιακά περιβάλλοντα, όπου η ενσωμάτωση συστημάτων, η επαναχρησιμοποίηση και η προσαρμοστικότητα είναι ζωτικής σημασίας.



Εικόνα 2.5 Αρχιτεκτονική SOA (πηγή)

2.2.6 Μικροϋπηρεσίες (Microservices Architectural Pattern)

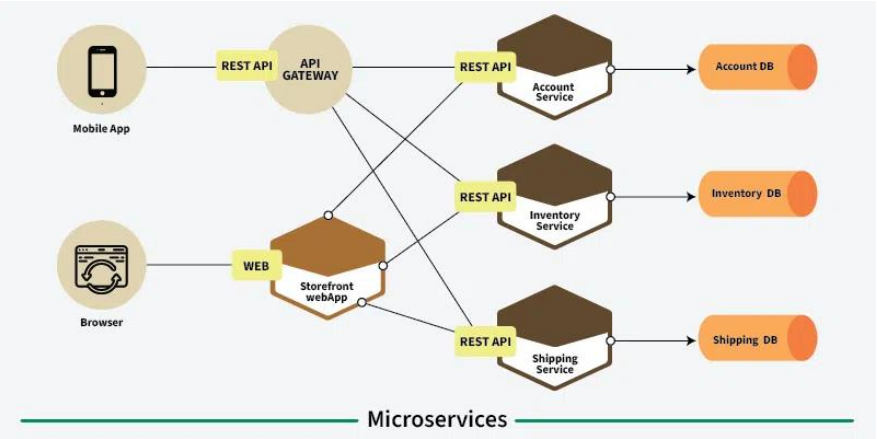
Το πρότυπο μικροϋπηρεσιών έχει υιοθετηθεί ευρέως από εταιρείες, από κολοσσούς όπως οι Amazon, Netflix και Spotify, μέχρι μικρομεσαίες επιχειρήσεις. Χαρακτηρίζεται από την ανάπτυξη ανεξάρτητων, αυτόνομων υπηρεσιών που επικεντρώνονται σε συγκεκριμένες και σαφώς ορισμένες λειτουργίες.

Σε αντίθεση με τις παραδοσιακές μονολιθικές αρχιτεκτονικές, οι μικροϋπηρεσίες προωθούν την κάθετη αποδόμηση των εφαρμογών σε διακριτές, επιχειρησιακά προσανατολισμένες υπηρεσίες. Κάθε υπηρεσία μπορεί να αναπτυχθεί, να δοκιμαστεί και να εγκατασταθεί ανεξάρτητα, χρησιμοποιώντας διαφορετικές τεχνολογίες. Αυτή η προσέγγιση προσφέρει:

- Ευελιξία στην επιλογή των τεχνολογιών.

- Στοχευμένη κλιμάκωση ανά υπηρεσία.
- Αυξημένη ανθεκτικότητα, καθώς η αποτυχία μιας υπηρεσίας δεν επηρεάζει ολόκληρη την εφαρμογή. [35]

Οι αρχιτεκτονική μικροϋπηρεσιών είναι μία εξέλιξη της αρχιτεκτονικής SOA. Ενώ στην αρχιτεκτονική SOA, κάθε υπηρεσία συνιστά ένα ολοκληρωμένο επιχειρησιακό κομμάτι, η αρχιτεκτονική μικροϋπηρεσιών εστιάζει σε πιο μικρά και εξειδικευμένα κομμάτια λογισμικού που επιτελούν ένα μονάχα ρόλο [34].



Εικόνα 2.6 Αρχιτεκτονική μικροϋπηρεσιών (πηγή)

2.3 Επιλογή θεμελιωδών αρχιτεκτονικών προτύπων στα πλαίσια της εργασίας

Στην παρούσα εργασία, θα εστιάσουμε σε ένα επιλεγμένο σύνολο γνωστών και καθιερωμένων αρχιτεκτονικών προτύπων: **Πελάτη-Εξυπηρετητή (Client-Server)**, **Τριών Επιπέδων (Three-Tier)** και **Model-View-Controller (MVC)**. Η επιλογή αυτών των συγκεκριμένων προτύπων βασίζεται στην εκτίμησή μας ότι η χρήση του AI στην αρχιτεκτονική λογισμικού βρίσκεται ακόμη σε πρώιμο στάδιο, με τα αποτελέσματά της να μην έχουν αξιολογηθεί επαρκώς.

Με αυτό το δεδομένο, στοχεύουμε να εξετάσουμε και να πειραματιστούμε με απλούστερες αρχιτεκτονικές, οι οποίες παρέχουν μια σαφή βάση και ένα ελεγχόμενο περιβάλλον για την αξιολόγηση της επίδρασης του AI. Η ενασχόληση με πιο σύνθετα πρότυπα, όπως οι Μικροϋπηρεσίες (Microservices) ή η Αρχιτεκτονική Προσανατολισμένη στις Υπηρεσίες (SOA), θα μπορούσε να εισαγάγει πολυπλοκότητες που πιθανώς να δυσχεράνουν την ανάλυση της συμβολής του AI στην παραγωγή της αρχιτεκτονικής.

Επιπλέον, η επιλογή θεμελιωδών προτύπων μας επιτρέπει να δημιουργήσουμε μια αφετηρία για μελλοντική έρευνα. Αυτό θα διευκολύνει τη σύγκριση αποτελεσμάτων και θα επιτρέψει τη σταδιακή μετάβαση σε πιο σύνθετες αρχιτεκτονικές σε επόμενες μελέτες, όταν τα αρχικά ευρήματα θα έχουν επαληθευτεί.

Ένας ακόμη λόγος για την επιλογή αυτών των προτύπων είναι η πρακτικότητα της μελέτης τους. Τα συγκεκριμένα πρότυπα είναι ευρέως χρησιμοποιούμενα, σχετικά απλά και κατανοητά από ένα μεγάλο εύρος περιβαλλόντων ανάπτυξης λογισμικού. Έτσι, η εστίασή μας στις αρχιτεκτονικές Πελάτη-Εξυπηρετητή, Τριών Στρωμάτων και MVC προσφέρει μια ισορροπημένη προσέγγιση για τη

διερεύνηση των δυνατοτήτων του AI στην αρχιτεκτονική λογισμικού, ενώ παράλληλα θέτει τις βάσεις για μελλοντικές εξελίξεις στον τομέα.

2.4 Κίνητρα και διαφοροποιήσεις της προσέγγισης μας

Τα συμπεράσματα από τις μελετηθείσες έρευνες στην ενότητα 2.1 αναδεικνύουν βασικούς τομείς που παραμένουν ανεξερεύνητοι ή ανεπαρκώς αντιμετωπισμένοι. Η προσέγγισή μας διαφοροποιείται και διαμορφώνεται από τους παρακάτω άξονες:

1. **Κλιμακωσιμότητα και Προσβασιμότητα των Μοντέλων**
   Παρόλο που η υπάρχουσα βιβλιογραφία συχνά εστιάζει σε μεγάλα, βασισμένα στο cloud LLMs όπως το GPT-3.5 ή σε συστήματα με έντονη εξειδίκευση όπως το ClassDiagGen, αυτές οι προσεγγίσεις μπορεί να μην είναι βιώσιμες για επιχειρήσεις ή ερευνητές που ανησυχούν για την ιδιωτικότητα, το κόστος ή την κλιμάκωση. Η δική μας εργασία δίνει έμφαση στη διερεύνηση τοπικών μοντέλων, όπως τα **LLaMA** και **Mixtral**, τα οποία μπορούν να λειτουργούν εντός των εγκαταστάσεων (on-premises). Αυτή η κατεύθυνση ανταποκρίνεται στη συνεχώς αυξανόμενη ζήτηση για λύσεις AI που δίνουν προτεραιότητα στην ασφάλεια δεδομένων και την ανεξαρτησία από υποδομές cloud.

2. **Αξιολόγηση του Ρόλου των Μικρότερων Μοντέλων**
   Πολλές σχετικές μελέτες επικεντρώνονται σε μεγάλα, απαιτητικά σε πόρους μοντέλα, ενώ μικρότερα ή κβαντισμένα μοντέλα (π.χ., με κβαντοποίηση Q4) συχνά παραβλέπονται. Η προσέγγισή μας στοχεύει να καλύψει αυτό το κενό, δοκιμάζοντας συστηματικά μοντέλα σε ένα φάσμα μεγεθών, από μικρής κλίμακας έως σχετικά υψηλής απόδοσης μοντέλα. Αυτό όχι μόνο παρέχει μια πιο πρακτική οπτική για περιβάλλοντα με περιορισμένους πόρους, αλλά εξετάζει επίσης κατά πόσο τα μικρότερα μοντέλα μπορούν να επιτύχουν συγκρίσιμα αποτελέσματα στη δημιουργία διαγραμμάτων κλάσεων.

3. **Συνδυασμός Μεθόδου Επαυξημένης Παραγωγικής Ανάκτησης (RAG) με LLMs**
   Σε αντίθεση με τις αναφερθείσες προσεγγίσεις στο 2.1, η έρευνά μας ενσωματώνει τεχνικές RAG για τη βελτίωση της ικανότητας των LLMs να διαχειρίζονται ασάφειες στις κειμενικές απαιτήσεις. Με την παροχή εξωτερικής γνώσης, όπως αρχεία RAG προσαρμοσμένα στον γνωστικό αντικείμενο, στοχεύουμε στη μείωση κοινών προβλημάτων κατανόησης και στη βελτίωση της συνολικής συνέπειας και ακρίβειας των παραγόμενων διαγραμμάτων.

4. **Διερεύνηση επίδρασης των Αρχιτεκτονικών Προτύπων**
   Οι προηγούμενες μελέτες δεν έχουν εξετάσει συστηματικά τον αντίκτυπο των αρχιτεκτονικών προτύπων (π.χ., Πελάτης-Εξυπηρετητής ,Τριών Επιπέδων , MVC) στην ποιότητα των παραγόμενων διαγραμμάτων UML. Η δική μας εργασία καλύπτει αυτό το κενό, αξιολογώντας κατά πόσο διαφορετικά μοντέλα σέβονται αυτές τις αρχιτεκτονικές αρχές και πώς η επιλογή των συνόλων λειτουργικών και μη λειτουργικών απαιτήσεων (FR/NFR) επηρεάζει τα αποτελέσματα.

2.5 Ερευνητικά ερωτήματα της εργασίας

Η ενσωμάτωση της τεχνητής νοημοσύνης στο πεδίο της αρχιτεκτονικής λογισμικού δημιουργεί ερωτήματα σχετικά με τις δυνατότητες, τους περιορισμούς και τη βέλτιστη χρήση της. Αυτή η έρευνα στοχεύει να διερευνήσει την προοπτική και τα όρια της συνδρομής του AI στην

αρχιτεκτονική λογισμικού, εστιάζοντας στη δημιουργία αρχιτεκτονικών μοντέλων, στη βελτίωση των απαντήσεων των μοντέλων, στη χρήση κατάλληλων μοντέλων και εκπαιδευτικού υλικού και στην αξιολόγηση της απόδοσης των μοντέλων αυτών. Τα ακόλουθα ερευνητικά ερωτήματα καθοδηγούν τη μελέτη μας:

1. **Μπορεί η τεχνητή νοημοσύνη να δημιουργήσει αρχιτεκτονικές που ενσωματώνουν συγκεκριμένα αρχιτεκτονικά πρότυπα από μια κειμενική περιγραφή του προβλήματος;** Αυτό το ερώτημα εξετάζει την ικανότητα του AI να ερμηνεύει περιγραφές προβλημάτων ή απαιτήσεις και να τις μεταφράζει σε καλά δομημένες αρχιτεκτονικές που ακολουθούν τα ζητούμενα αρχιτεκτονικά πρότυπα. Μέσω αυτής της ανάλυσης, στοχεύουμε να αξιολογήσουμε την ικανότητα της τεχνητής νοημοσύνης να δημιουργεί λειτουργικές και σχετικές αρχιτεκτονικές χωρίς εκτεταμένη ανθρώπινη καθοδήγηση, βελτιώνοντας έτσι τη διαδικασία σχεδίασης αρχιτεκτονικών.

2. **Ποια θα μπορούσε να είναι η μορφή μιας κειμενικής περιγραφής προβλήματος ώστε να διασφαλίζεται η δημιουργία ορθών αρχιτεκτονικών;**
   Η διερεύνηση της ιδανικής δομής, του περιεχομένου και της ακρίβειας των εισόδων περιγραφής είναι ουσιαστική για τη βελτίωση της ικανότητας του AI να παράγει ακριβείς και αποτελεσματικές αρχιτεκτονικές. Αυτό το ερώτημα επιδιώκει να προσδιορίσει ποια στοιχεία μιας περιγραφής είναι πιο χρήσιμα για την καθοδήγηση του AI στην παραγωγή δομικά σωστών αποτελεσμάτων.

3. **Μπορούν τα αποτελέσματα να βελτιωθούν μέσω της μεθόδου Retrieval-Augmented Generation (RAG) και παρόμοιων τεχνικών;**
   Η μέθοδος RAG και συναφείς τεχνικές προσφέρουν τη δυνατότητα εμπλουτισμού των απαντήσεων που παράγονται από το AI με σχετική στο αντικείμενο γνώση. Αυτό το ερώτημα εξετάζει εάν η χρήση του RAG μπορεί να βελτιώσει την ακρίβεια, τη συνάφεια και την ποιότητα των διαγραμμάτων κλάσεων που δημιουργούν τα μοντέλα, καθώς και τους περιορισμούς αυτής της μεθόδου.

4. **Ποια είναι η επίδραση διαφορετικών πηγών υλικού στη μέθοδο RAG;**
   Αυτό το ερώτημα εξετάζει την επίδραση ποικίλων εκπαιδευτικών υλικών στην ποιότητα των απαντήσεων του AI στην αρχιτεκτονική λογισμικού. Η κατανόηση του πώς διαφορετικές πηγές RAG επηρεάζουν τις ικανότητες των μοντέλων μπορεί να συμβάλει στη βελτίωση των μεθόδων εκπαίδευσης και στην παραγωγή πιο αξιόπιστων, περιεκτικών απαντήσεων.

5. **Είναι αξιόπιστα τα τοπικά μοντέλα ανοιχτού κώδικα για την χρήση τους στην αρχιτεκτονική λογισμικού; Ποιο είναι το ελάχιστο αποδεκτό μέγεθος/τύπος μοντέλου;**
   Η απόδοση των μοντέλων ανοιχτού κώδικα στην υποβοηθούμενη αρχιτεκτονική αποτελεί αντικείμενο ανάλυσης, με στόχο την αναγνώριση του μικρότερου και πιο αποδοτικού μοντέλου που μπορεί να παρέχει αποδεκτά αποτελέσματα. Αυτό το ερώτημα αποσκοπεί στην παροχή προτάσεων για οικονομικά αποδοτικές, προσβάσιμες και ασφαλείς για την ιδιωτικότητα λύσεις AI.

6. **Πώς μπορούμε να αξιολογήσουμε την αποτελεσματικότητα της τεχνητής νοημοσύνης στη σχεδίαση αρχιτεκτονικής λογισμικού;**
   Η αξιολόγηση της συμβολή του AI στη σχεδίαση αρχιτεκτονικής απαιτεί καλά καθορισμένα κριτήρια και μετρικές. Αυτό το ερώτημα διερευνά πιθανούς τρόπους αξιολόγησης για τη

μέτρηση της ποιότητας, της ακρίβειας και της συνολικής χρησιμότητας των αρχιτεκτονικών προτάσεων που δημιουργεί η τεχνητή νοημοσύνη.

Κάθε ερώτημα είναι κρίσιμο για την κατανόηση της ενσωμάτωσης του AI στις διαδικασίες της αρχιτεκτονικής λογισμικού, από τη δημιουργία έως την αξιολόγηση. Μέσω της διερεύνησης αυτών των πτυχών, η έρευνα αποσκοπεί στην παροχή μιας θεμελιώδους κατανόησης του ρόλου της τεχνητής νοημοσύνης στην αρχιτεκτονική λογισμικού και στον εντοπισμό μεθόδων για τη βελτιστοποίηση της απόδοσης και της αξιοπιστίας της σε πραγματικές εφαρμογές.

3 Προσέγγιση

3.1 Παράμετροι

Για να εξετάσουμε διεξοδικά τα ερευνητικά μας ερωτήματα, καθορίσαμε συγκεκριμένες παραμέτρους που θα κατευθύνουν τη διαδικασία και θα διασφαλίσουν σαφήνεια στον πειραματικό μας σχεδιασμό.

3.1.1 Αρχιτεκτονικά πρότυπα υπό εξέταση

Όπως έχει ήδη συζητηθεί, επιλέξαμε να εστιάσουμε σε μια υποσύνολο ευρέως χρησιμοποιούμενων αρχιτεκτονικών προτύπων: Πελάτης-Εξυπηρετητής, Τριών Επιπέδων και Model-View-Controller.

Η επιλογή αυτή, όπως ήδη έχουμε αναλύσει βασίζεται στην απλότητα αυτών των προτύπων, τη διαδεδομένη υιοθέτησή τους και την ικανότητά τους να παρέχουν ένα ελεγχόμενο περιβάλλον για τα πειράματά μας. Εστιάζοντας σε αυτά τα θεμελιώδη πρότυπα, μπορούμε να αναλύσουμε την ικανότητα του AI να συμμορφώνεται με αναγνωρισμένες αρχιτεκτονικές, αποφεύγοντας την πολυπλοκότητα που θα μπορούσαν να εισάγουν πιο σύνθετα πρότυπα, όπως οι Μικροϋπηρεσίες ή η Αρχιτεκτονική Προσανατολισμένη στις Υπηρεσίες.

3.1.2 Μελέτη Περίπτωσης : Η εφαρμογή DCC

Η **Εφαρμογή Μετατροπής Συντεταγμένων (Dummy Coordination Conversion - DCC)** είναι ένα σύστημα λογισμικού που διαχειρίζεται ομάδες συντεταγμένων σε Καρτεσιανή και πολική μορφή. Το σύστημα επιτρέπει στους χρήστες να μετατρέπουν, αποθηκεύουν, ανακτούν, τροποποιούν και διαγράφουν ομάδες συντεταγμένων.

Η επιλογή αυτής της σχετικά απλής εφαρμογής μας επιτρέπει να επικεντρωθούμε σε βασικά στοιχεία της έρευνας, όπως η ικανότητα του AI να ακολουθεί το αρχιτεκτονικό πρότυπο, η ακρίβεια στη δημιουργία διαγραμμάτων κλάσεων και η συμμόρφωση με τις απαιτήσεις που έχουν οριστεί. Η απλότητα της εφαρμογής μειώνει την περιττή πολυπλοκότητα, διευκολύνοντας την αξιολόγηση της απόδοσης της τεχνητής νοημοσύνης με σαφή έμφαση στην αρχιτεκτονική και τη σχεδιαστική ακρίβεια. Η περιγραφή του συστήματος θα περιλαμβάνει τη συνολική επισκόπηση της εφαρμογής, καθώς επίσης και τις Λειτουργικές και Μη Λειτουργικές Απαιτήσεις της.

Για να αποκτήσουμε μια ολοκληρωμένη κατανόηση του πώς κάθε αρχιτεκτονικό πρότυπο επηρεάζει την εφαρμογή μας, αναπτύξαμε τρεις διαφορετικές εκδόσεις της εφαρμογής σε Java, με κάθε έκδοση να ακολουθεί ένα από τα επιλεγμένα πρότυπα: Πελάτη-Εξυπηρετητή, Τριών Επιπέδων και Model-View-Controller. Επιλέξαμε τη Java ως γλώσσα προγραμματισμού λόγω του αυστηρού συστήματος τύπων που διαθέτει και της ευθυγράμμισης της στις αντικειμενοστραφείς αρχές, οι οποίες, κατά την άποψή μας, καθιστούν ευκολότερη τη δημιουργία, κατανόηση και υλοποίηση της αρχιτεκτονικής.

Σύνολο λειτουργικών απαιτήσεων

Αρχικά, θα επικεντρωθούμε στις λειτουργικές απαιτήσεις. Για να παρατηρήσουμε και να αξιολογήσουμε τις διαφορές στις εξόδους που παράγονται από τα μοντέλα, έχουμε ετοιμάσει δύο διαφορετικές εκδόσεις του συνόλου των λειτουργικών απαιτήσεων.

Κάθε σύνολο έχει σχεδιαστεί ώστε να αναδεικνύει διαφορετικές πτυχές της λειτουργικότητας του λογισμικού. Μετά την παρουσίαση, θα πραγματοποιηθεί λεπτομερής σύγκριση και ανάλυση των δύο συνόλων, προκειμένου να τονιστούν οι διαφορές τους, οι οποίες πιθανότατα θα επηρεάσουν τη δημιουργία διαγραμμάτων κλάσεων και τη συμμόρφωση με τα αρχιτεκτονικά πρότυπα.

Αυτή η προσέγγιση—με τη χρήση διαφορετικών εκδόσεων απαιτήσεων—στοχεύει να φωτίσει τον τρόπο με τον οποίο η λεπτομέρεια, η ακρίβεια και η δομή των απαιτήσεων επηρεάζουν την ποιότητα και την ακρίβεια των εξόδων που παράγονται από το AI. Με αυτόν τον τρόπο, μπορούμε να συγκεντρώσουμε πολύτιμες γνώσεις σχετικά με το βέλτιστο επίπεδο λεπτομέρειας που απαιτείται για την αυτόματη δημιουργία διαγραμμάτων κλάσεων. Η Εικόνα 3.1 απεικονίζει το πρώτο σύνολο λειτουργικών απαιτήσεων ενώ η Εικόνα 3.2 το δεύτερο.

1. Δημιουργία Ομάδας Συντεταγμένων:
   - Οι χρήστες μπορούν να δημιουργήσουν μια ομάδα συντεταγμένων που αποτελείται από Καρτεσιανές και πολικές συντεταγμένες.
   - Οι χρήστες εισάγουν έναν τύπο συντεταγμένων (είτε Καρτεσιανές είτε πολικές) και το λογισμικό αναλαμβάνει τη μετατροπή στον άλλο τύπο.
   - Μια ομάδα συντεταγμένων περιλαμβάνει:
     - Καρτεσιανές: (x, y)
     - Πολικές: (r, θ)

2. Αποθήκευση Ομάδας Συντεταγμένων:
   - Οι ομάδες συντεταγμένων αποθηκεύονται σε βάση δεδομένων MySQL με τα παρακάτω χαρακτηριστικά:
     - Μοναδικός αυτόματα παραγόμενος αναγνωριστικός αριθμός (ακέραιος).
     - Ετικέτα καθοριζόμενη από τον χρήστη για αναγνώριση.
     - Αυτόματα καταχωριζόμενη χρονική σήμανση.

3. Ανάκτηση Ομάδας Συντεταγμένων:
   - Ανάκτηση ομάδων συντεταγμένων μέσω:
     - Ετικέτας: Αναζήτηση μέσω μιας ετικέτας που έχει ορίσει ο χρήστης.
     - Προβολή όλων: Ανάκτηση και εμφάνιση όλων των αποθηκευμένων ομάδων συντεταγμένων.

4. Τροποποίηση Ομάδας Συντεταγμένων:
   - Επιλογή και επεξεργασία λεπτομερειών μιας επιλεγμένης ομάδας συντεταγμένων (τιμές Καρτεσιανών ή πολικών συντεταγμένων, ετικέτα) και αποθήκευση της ενημερωμένης έκδοσης.

5. Διαγραφή Ομάδας Συντεταγμένων:
   - Διαγραφή μιας ομάδας συντεταγμένων από τη βάση δεδομένων.

Εικόνα 3.1 Πρώτο Σύνολο Λειτουργικών Απαιτήσεων

1.Λογική:
1.1 Ορισμός Καρτεσιανών συντεταγμένων ως ζεύγος (x, y) αριθμών υψηλής ακρίβειας.
1.2 Ορισμός πολικών συντεταγμένων ως ζεύγος (r, θ) αριθμών υψηλής ακρίβειας.
1.3 Μετατροπή Καρτεσιανών συντεταγμένων (x, y) σε πολικές (r, θ).
1.4 Μετατροπή πολικών συντεταγμένων (r, θ) σε Καρτεσιανές (x, y).
1.5 Μια ομάδα συντεταγμένων περιλαμβάνει Καρτεσιανές (x, y) και πολικές (r, θ) συντεταγμένες.

2. Λειτουργίες Δεδομένων:
2.1 CreateOp: Δημιουργία μοναδικού αυτόματα παραγόμενου αριθμητικού αναγνωριστικού (id) για την ομάδα συντεταγμένων, προσθήκη χρονικής σήμανσης, και αποθήκευση της ετικέτας και των τιμών της ομάδας συντεταγμένων ως νέα εγγραφή στη βάση δεδομένων.
2.2 ReadOp: Δημιουργία λίστας όλων των αποθηκευμένων εγγραφών ομάδων συντεταγμένων. Για κάθε εγγραφή εμφανίζονται: id, ετικέτα, τιμές Καρτεσιανών συντεταγμένων, τιμές πολικών συντεταγμένων, χρονική στιγμή προσθήκης ή τροποποίησης, ταξινομημένα κατά id.
2.3 UpdateOp: Ενημέρωση μιας εγγραφής ομάδας συντεταγμένων με νέες τιμές που εισάγει ο χρήστης και αποθήκευση της ενημερωμένης εγγραφής στη βάση δεδομένων.
2.4 DeleteOp: Διαγραφή μιας εγγραφής ομάδας συντεταγμένων από τη βάση δεδομένων.

3. Αλληλεπιδράσεις Χρήστη:
3.1 Εισαγωγή νέας ομάδας συντεταγμένων και μιας ετικέτας για την ονομασία της ομάδας.
3.2 Όταν ο χρήστης εισάγει Καρτεσιανές συντεταγμένες (x, y), υπολογισμός των αντίστοιχων πολικών συντεταγμένων (r, θ) μέσω μετατροπής.
3.3 Όταν ο χρήστης εισάγει πολικές συντεταγμένες (r, θ), υπολογισμός των αντίστοιχων Καρτεσιανών συντεταγμένων (x, y) μέσω μετατροπής.
3.4 Όταν επιλέγεται "προσθήκη", εκτέλεση της CreateOp για τις τρέχουσες τιμές της ομάδας συντεταγμένων και την ετικέτα.
3.5 Όταν επιλέγεται "ενημέρωση", εκτέλεση της UpdateOp για την επιλεγμένη εγγραφή από τη λίστα που δημιουργείται μέσω της ReadOp.
3.6 Όταν επιλέγεται "διαγραφή", εκτέλεση της DeleteOp για την επιλεγμένη εγγραφή από τη λίστα που δημιουργείται μέσω της ReadOp.
3.7 Κατά την αρχικοποίηση ή όταν εκτελείται "προσθήκη", "ενημέρωση" ή "διαγραφή", ανανέωση και εμφάνιση της λίστας που δημιουργείται μέσω της ReadOp.
3.8 Φιλτράρισμα της λίστας των εμφανιζόμενων ομάδων συντεταγμένων με βάση μια ετικέτα που εισάγει ο χρήστης.
3.9 Αφαίρεση ενεργού φίλτρου και εμφάνιση όλων των αποθηκευμένων εγγραφών ομάδων συντεταγμένων, όπως στη διαδικασία της ReadOp.

Εικόνα 3.2 Δεύτερο Σύνολο Λειτουργικών Απαιτήσεων

Τα δύο σύνολα λειτουργικών απαιτήσεων περιγράφουν τις ίδιες βασικές λειτουργίες, αλλά διαφέρουν στη δομή, τη λεπτομέρεια και τον προσανατολισμό τους. Ακολουθούν οι βασικές διαφορές:

Δομή και Οργάνωση

- **Πρώτο Σύνολο:** Οργανώνεται με βάση τις διάφορες λειτουργίες του συστήματος (Δημιουργία, Αποθήκευση, Ανάκτηση, Τροποποίηση, Διαγραφή). Κάθε λειτουργία παρουσιάζεται ως χαρακτηριστικό που απευθύνεται στον χρήστη.
- **Δεύτερο Σύνολο:** Χωρίζεται σε τρεις βασικές κατηγορίες—Λογική, Λειτουργίες Δεδομένων και Αλληλεπιδράσεις Χρήστη—με έμφαση σε λεπτομέρειες υλοποίησης χαμηλότερου επιπέδου, αναλύοντας κάθε λειτουργική περιοχή σε μικρότερες, συγκεκριμένες λειτουργίες.

Λεπτομέρεια και Επίπεδο Ανάλυσης

- **Πρώτο Σύνολο:** Παρουσιάζει τις απαιτήσεις με απλή, επικεντρωμένη στον χρήστη μορφή, περιγράφοντας βασικές ενέργειες χωρίς να εμβαθύνει σε λεπτομέρειες λειτουργίας. Για παράδειγμα, περιγράφει την "Ανάκτηση Ομάδας Συντεταγμένων" γενικά, αναφέροντας πώς οι χρήστες μπορούν να αναζητήσουν μέσω ετικέτας ή να δουν όλες τις εγγραφές.
- **Δεύτερο Σύνολο:** Εμβαθύνει σε λεπτομέρειες για κάθε λειτουργία, ορίζοντας συγκεκριμένες ενέργειες για διαφορετικά στάδια (π.χ. “CreateOp,” “ReadOp”) και τα χαρακτηριστικά τους. Περιλαμβάνει ακόμη και εσωτερικές λειτουργίες του συστήματος, όπως ταξινόμηση εγγραφών κατά ID ή ανανέωση προβολής μετά από κάθε ενέργεια.

Τεχνική Γλώσσα έναντι Προσέγγισης Εστιασμένης στον Χρήστη

- **Πρώτο Σύνολο:** Περιγράφει τις λειτουργίες με γλώσσα που ευθυγραμμίζεται με τις εργασίες του χρήστη και τους υψηλού επιπέδου (high-level) στόχους , καθιστώντας το πιο κατανοητό για μη τεχνικούς ενδιαφερόμενους.
- **Δεύτερο Σύνολο:** Χρησιμοποιεί περισσότερο τεχνική γλώσσα και όρους, όπως “CreateOp,” “ReadOp,” και “αριθμοί υψηλής ακρίβειας,” απευθυνόμενο κυρίως σε προγραμματιστές ή τεχνικά κοινά. Δίνει έμφαση στη διαχείριση δεδομένων και την εσωτερική λογική, παρά στις ενέργειες του χρήστη.

Προδιαγραφή Αλληλεπιδράσεων Χρήστη

- **Πρώτο Σύνολο:** Υποθέτει ότι οι χρήστες εκτελούν ενέργειες όπως ανάκτηση και τροποποίηση, χωρίς να αναλύει τα βήματα της αλληλεπίδρασης.
- **Δεύτερο Σύνολο:** Αναλύει λεπτομερώς τις αλληλεπιδράσεις του χρήστη, συμπεριλαμβάνοντας συγκεκριμένες ενέργειες όπως “προσθήκη,” “ενημέρωση,” και “διαγραφή,” με εξηγήσεις για κάθε βήμα (π.χ. μετατροπή συντεταγμένων κατά την εισαγωγή δεδομένων, εμφάνιση λίστας μετά από κάθε ενέργεια).

Έμφαση στη Διαχείριση Δεδομένων και στην Ανανέωση

- **Πρώτο Σύνολο:** Αναφέρει τη δυνατότητα αποθήκευσης, ανάκτησης και διαγραφής ομάδων συντεταγμένων, χωρίς να επεκτείνεται σε μηχανισμούς ανανέωσης δεδομένων ή ενημέρωσης προβολής.
- **Δεύτερο Σύνολο:** Εστιάζει στην ανανέωση της εμφανιζόμενης λίστας μετά από κάθε λειτουργία δημιουργίας, ενημέρωσης ή διαγραφής, υπογραμμίζοντας τη σημασία της προβολής δεδομένων για τον χρήστη. Παρέχει επίσης επιλογές φιλτραρίσματος ή κατάργησης φίλτρων, βελτιώνοντας τη διαχείριση δεδομένων.

Εστίαση στη Μετατροπή και Αποθήκευση Δεδομένων

- **Πρώτο Σύνολο:** Υπονοεί την ανάγκη για μετατροπή συντεταγμένων, χωρίς να εξηγεί τη λογική πίσω από αυτές τις μετατροπές.
- **Δεύτερο Σύνολο:** Καθορίζει ρητά τις μετατροπές συντεταγμένων (Καρτεσιανές σε πολικές και αντίστροφα) στην ενότητα "Λογική," εξηγώντας τις εσωτερικές λειτουργίες που απαιτούνται για την υποστήριξη εισαγωγών και μετατροπών από τον χρήστη.

Συνοψίζοντας το πρώτο σύνολο είναι προσανατολισμένο στον χρήστη, περιγράφοντας λειτουργίες υψηλού επιπέδου με κατανοητή γλώσσα, κατάλληλη για μη τεχνικούς ενδιαφερόμενους. Αντίθετα, το δεύτερο σύνολο είναι πιο τεχνικό και εστιάζει στην υλοποίηση, παρέχοντας λεπτομέρειες για λειτουργίες, αλληλεπιδράσεις χρήστη και εσωτερικές διαδικασίες, καθιστώντας το καταλληλότερο για προγραμματιστές που χρειάζονται ακριβείς οδηγίες για κάθε λειτουργία.

Η ανάλυση των διαφορών δείχνει ότι το δεύτερο σύνολο, με την έμφαση στη σαφήνεια της επιχειρησιακής λογικής και τη λεπτομερή δομή, είναι πιθανότερο να οδηγήσει σε καλύτερα παραγόμενα διαγράμματα κλάσεων από τα LLMs. Η καλά δομημένη και λεπτομερής φύση του δεύτερου συνόλου μπορεί να καθοδηγήσει τα LLMs να παράγουν ακριβέστερα διαγράμματα, προσαρμοσμένα στο επιχειρησιακό πλαίσιο της εφαρμογής και συμβατά με τα αρχιτεκτονικά πρότυπα.

Ωστόσο, αυτή η υπόθεση παραμένει ανεπιβεβαίωτη. Στα πειράματά μας, στοχεύουμε να αξιολογήσουμε αν αυτές οι υποθέσεις ισχύουν, εξετάζοντας την ποιότητα και την ακρίβεια των διαγραμμάτων κλάσεων που παράγονται από κάθε σύνολο απαιτήσεων. Μέσα από αυτή τη διαδικασία, φιλοδοξούμε να αντλήσουμε πολύτιμες γνώσεις σχετικά με την επίδραση της λεπτομέρειας και της ακρίβειας των απαιτήσεων στη δημιουργία αρχιτεκτονικής με τη βοήθεια της τεχνητής νοημοσύνης, παρέχοντας τεκμηριωμένα συμπεράσματα για τη βέλτιστη μορφή απαιτήσεων σε σχεδιασμό που βασίζεται σε LLMs.

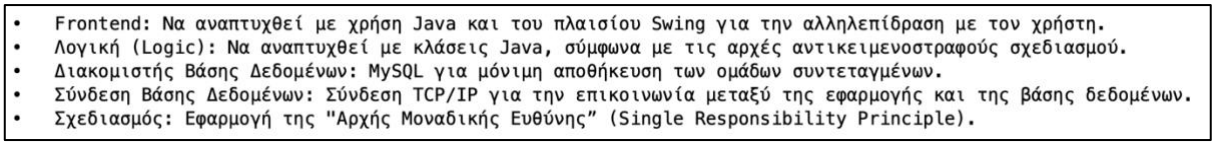
Σύνολα Μη Λειτουργικών Απαιτήσεων

Η Εικόνα 3.3 απεικονίζει το πρώτο σύνολο μη λειτουργικών απαιτήσεων.

```
•   Frontend: Ανάπτυξη με χρήση Java και του πλαισίου Swing για την αλληλεπίδραση με τον χρήστη.
•   Βάση Δεδομένων: MySQL για μόνιμη αποθήκευση των ομάδων συντεταγμένων.
•   Σύνδεση TCP/IP: Για τη διαχείριση της επικοινωνίας μεταξύ της εφαρμογής και της βάσης δεδομένων.
```

Εικόνα 3.3 Πρώτο Σύνολο Μη Λειτουργικών Απαιτήσεων

Η Εικόνα 3.4 απεικονίζει το δεύτερο σύνολο μη λειτουργικών απαιτήσεων.



Εικόνα 3.4 Δεύτερο Σύνολο Μη Λειτουργικών Απαιτήσεων

Η σύγκριση των δύο συνόλων αποκαλύπτει ότι το δεύτερο σύνολο παρέχει μια πιο λεπτομερή και δομημένη προσέγγιση, η οποία μπορεί να οδηγήσει σε βελτιωμένη σαφήνεια και τήρηση των αρχών σχεδιασμού.

Το πρώτο σύνολο προσφέρει μια συνοπτική επισκόπηση των μη λειτουργικών απαιτήσεων, εστιάζοντας σε βασικά στοιχεία όπως το πλαίσιο του frontend (Java με Swing), την επιλογή της βάσης δεδομένων (MySQL) και το πρωτόκολλο επικοινωνίας (TCP/IP). Ωστόσο, απουσιάζει η εξειδίκευση σε αρχές σχεδιασμού ή πρακτικές αντικειμενοστραφούς προγραμματισμού, αφήνοντας ορισμένες λεπτομέρειες υλοποίησης ανοιχτές σε ερμηνείες.

Αντίθετα, το δεύτερο σύνολο ενισχύει τη σαφήνεια διαιρώντας τις απαιτήσεις σε πιο συγκεκριμένες κατηγορίες, όπως “Λογική” και “Σχεδιασμός,” και εισάγει οδηγίες για τη δομή του κώδικα. Για παράδειγμα, προσδιορίζει ότι το λογικό επίπεδο πρέπει να αναπτυχθεί με κλάσεις Java που ακολουθούν τις αρχές αντικειμενοστραφούς προγραμματισμού, εξασφαλίζοντας μια αρθρωτή και εύκολα συντηρήσιμη δομή. Επιπλέον, η ενσωμάτωση της “Αρχής Μοναδικής Ευθύνης” στην ενότητα σχεδιασμού παρέχει σαφή κατεύθυνση για την οργάνωση των κλάσεων, προωθώντας καθαρό και στοχευμένο κώδικα.

Η έμφαση του δεύτερου συνόλου στον αντικειμενοστραφή σχεδιασμό και τις κατευθυντήριες αρχές, όπως η “Αρχής Μοναδικής Ευθύνης” μπορεί να οδηγήσει σε ένα πιο συντηρήσιμο και επεκτάσιμο σύστημα, όπως θα αποτυπωθεί στα παραγόμενα διαγράμματα κλάσεων.

Συνοψίζοντας το πρώτο σύνολο παρέχει μια σύντομη επισκόπηση των τεχνικών απαιτήσεων, ενώ το δεύτερο σύνολο αποτελεί έναν πιο αναλυτικό οδηγό, ενσωματώνοντας αρχές αντικειμενοστραφούς σχεδιασμού και σαφείς οδηγίες, καθιστώντας το ιδανικό για σύνθετα έργα όπου η σαφήνεια και η δυνατότητα συντήρησης είναι καίριας σημασίας. Τα πειράματα που θα ακολουθήσουν θα αξιολογήσουν αν αυτές οι διαφορές στα μη λειτουργικά σύνολα απαιτήσεων οδηγούν σε παρατηρήσιμες βελτιώσεις στα διαγράμματα κλάσεων σε υλοποιήσεις που υποστηρίζονται από AI.

Αξίζει να σημειωθεί ότι το πρώτο σύνολο λειτουργικών απαιτήσεων θα συνδυαστεί με το πρώτο σύνολο μη λειτουργικών απαιτήσεων, και το ίδιο ισχύει για τα δεύτερα σύνολα. Αυτή η ευθυγράμμιση ενισχύει την προσδοκία μας ότι η χρήση των δεύτερων συνόλων μαζί μπορεί να οδηγήσει σε καλύτερα αποτελέσματα.

3.1.3 Είσοδος Prompt

Συνθέτοντας όλα τα παραπάνω καταλήγουμε στο τελικό prompt που θα δοθεί στα μεγάλα γλωσσικά μοντέλα. Για το πείραμά μας, επιλέξαμε ένα ενιαίο και περιεκτικό prompt:

```
Περιγραφή Εργασίας:
Σας ανατίθεται η επεξεργασία της περιγραφής μιας εφαρμογής λογισμικού και των απαιτήσεών της, προκειμένου να δημιουργήσετε
ένα διάγραμμα κλάσεων χρησιμοποιώντας το PlantUML.
Το διάγραμμα κλάσεων πρέπει να σέβεται την καθορισμένη αρχιτεκτονική λογισμικού, να ορίζει όλες τις απαραίτητες κλάσεις
και να αναπαριστά με ακρίβεια τις σχέσεις μεταξύ τους.

Επισκόπηση: {Software_app_overview}

Αρχιτεκτονική Λογισμικού: {Architecture}

Λειτουργικές Απαιτήσεις: {FR}

Μη Λειτουργικές Απαιτήσεις: {NFR}

Δημιουργήστε ένα διάγραμμα κλάσεων χρησιμοποιώντας το PlantUML που να ορίζει όλες τις απαραίτητες κλάσεις και σχέσεις
βάσει της περιγραφόμενης αρχιτεκτονικής, των απαιτήσεων και της λειτουργικότητας. Διασφαλίστε ότι:
•   Το διάγραμμα αντικατοπτρίζει τον διαχωρισμό των ευθυνών σύμφωνα με την καθορισμένη αρχιτεκτονική.
•   Κάθε κλάση περιλαμβάνει κατάλληλα χαρακτηριστικά και μεθόδους.
•   Όλες οι σχέσεις (π.χ., σύνθεση, συγκέντρωση, κληρονομικότητα) αναπαρίστανται στο PlantUML και είναι σαφώς ορισμένες.
•   Δημιουργήστε κατάλληλα πακέτα για την οργάνωση των κλάσεων
```

Εικόνα 3.5 Prompt

Οφείλουμε σε αυτό το σημείο να διευκρινίσουμε ότι όλες οι είσοδοι στα μοντέλα δόθηκαν στην Αγγλική Γλώσσα, ωστόσο για τις ανάγκες της παρούσας εργασίας, τόσο τα σύνολα απαιτήσεων όσο και το prompt έχουν αποδοθεί στα Ελληνικά.

Το prompt όπως φαίνεται στην Εικόνα 3.5 σχεδιάστηκε ώστε να περιλαμβάνει όλες τις σχετικές πληροφορίες, από την επισκόπηση της εφαρμογής και την αρχιτεκτονική της, μέχρι τις λειτουργικές και μη λειτουργικές απαιτήσεις, καθοδηγώντας το LLM στη δημιουργία ενός πλήρους διαγράμματος κλάσεων στο PlantUML. Περιλαμβάνει τόσο γενικές κατευθυντήριες γραμμές όσο και συγκεκριμένες οδηγίες σχετικά με χαρακτηριστικά, μεθόδους, σχέσεις και την οργάνωση σε πακέτα, παρέχοντας το απαραίτητο πλαίσιο για την παραγωγή ακριβών και οργανωμένων διαγραμμάτων κλάσεων.

Η απόφαση να χρησιμοποιηθεί μία μόνο εκδοχή του prompt, αντί για τη δημιουργία πολλαπλών εκδοχών, βασίζεται σε τρεις κύριους λόγους:

1. **Αποφυγή Επανάληψης:**
   Σε αντίθεση με τις λειτουργικές απαιτήσεις, όπου διαφορετικά επίπεδα λεπτομέρειας και εξειδίκευσης μπορούν να οδηγήσουν σε σημαντικά διαφοροποιημένες αποκρίσεις, το prompt εδώ σχεδιάστηκε ώστε να είναι πλήρες, καλύπτοντας όλες τις απαραίτητες πτυχές (επισκόπηση, αρχιτεκτονική, λειτουργικές και μη λειτουργικές απαιτήσεις). Η δημιουργία επιπλέον εκδοχών του prompt θα οδηγούσε σε επαναλήψεις, καθώς κάθε εκδοχή θα έπρεπε να επαναλαμβάνει παρόμοιες οδηγίες για να παρέχει το ίδιο πλήρες πλαίσιο στο LLM.

2. **Διατήρηση Συνοχής στην Αξιολόγηση:**
   Με πολλαπλές εκδοχές των λειτουργικών και μη λειτουργικών απαιτήσεων, καθώς και παραλλαγές στα RAG αρχεία (όπως θα συζητηθεί αργότερα), η εστίασή μας είναι να αξιολογήσουμε πώς οι αλλαγές στη λεπτομέρεια των εισόδων επηρεάζουν την ερμηνεία και την έξοδο των μοντέλων. Ένα συνεπές prompt διασφαλίζει ότι η επίδραση των υπόλοιπων παραμέτρων αξιολογείται με ακρίβεια, χωρίς να εισάγονται μεταβλητές που σχετίζονται με διαφορετικές εκδοχές του prompt.

3. **Εστίαση στα Κύρια Ερευνητικά Ερωτήματα:**
   Ο κύριος στόχος μας είναι να εξετάσουμε πώς το AI ανταποκρίνεται στις διαφορές στα σύνολα λειτουργικών/μη λειτουργικών απαιτήσεων, στα αρχεία RAG και στην απόδοση των μοντέλων. Στο πλαίσιο της παρούσας εργασίας και του συγκεκριμένου πειράματος-εφαρμογής, δεν

κρίνεται απαραίτητη η ενασχόληση με μικρές διαφοροποιήσεις στη διαμόρφωση παραλλαγών του συγκεκριμένου prompt. Η διατήρηση ενός σταθερού prompt μας επιτρέπει να απομονώσουμε την επίδραση των υπόλοιπων μεταβλητών στην ποιότητα της εξόδου, χωρίς να συγχέουμε τα αποτελέσματα με επιπλέον αμελητέες διαφορές.

3.1.4 Επιλογή Μεγάλων Γλωσσικών Μοντέλων

Για να αποκτήσουμε σαφή εικόνα σχετικά με την απόδοση διαφορετικών μοντέλων, επιλέξαμε μια ποικιλία μοντέλων, η οποία βασίζεται στην εξειδίκευση και το μέγεθος των παραμέτρων, που επηρεάζουν τις απαιτήσεις μνήμης. Στον Πίνακας 3.1, ακολουθούν τα μοντέλα που επιλέξαμε για αξιολόγηση:

Όνομα Μοντέλου	**Πηγή**	**Μέγεθος Παραμέτρων**	**Quantization**	**Εξειδίκευση Μοντέλου**
llama3.1:latest	Local	8B	Q4_0	Πολυγλωσσική Υποστήριξη Κειμένου και Κώδικα
phi3:medium-128k	Local	14B	Q4_0	• Ερευνητική Χρήση στα Αγγλικά: • Κατάλληλο για Εφαρμογές που Απαιτούν: 1) Περιβάλλοντα με περιορισμούς σε μνήμη ή υπολογιστική ισχύ. 2) Σενάρια με περιορισμούς καθυστέρησης. 3) Ισχυρή λογική και μαθηματική συλλογιστική. 4) Επεξεργασία μεγάλων συμφραζομένων.
gemma2:27b	Local	27.2B	Q4_0	• Πολλαπλές Μορφές Δεδομένων • Ποικιλία Εργασιών Παραγωγής Κειμένου: Συμπεριλαμβανομένων ερωτήσεων-απαντήσεων, περίληψης και λογικής επεξεργασίας.
command-r	Local	32.3B	Q4_0	• Βελτιστοποίηση για Εργασίες Μεγάλου Συμφραζομένου: Όπως η δημιουργία περιεχομένου με ενίσχυση ανάκτησης (RAG) και η χρήση εξωτερικών APIs και εργαλείων.
mixtral:8x7b	Local	46.7B	Q4_0	• Ικανότητα Διαχείρισης Συμφραζομένων 32k Tokens • Υποστήριξη Γλωσσών: Αγγλικά, Γαλλικά, Ιταλικά, Γερμανικά και Ισπανικά. • Ισχυρή Απόδοση στη Δημιουργία Κώδικα

Όνομα Μοντέλου	Πηγή	Μέγεθος Παραμέτρων	Quantization	Εξειδίκευση Μοντέλου
gpt-4o (chatGPT4o)	online	200B	N/A	-
gpt-4o (Software Architecture Visualiser)	online	N/A	N/A	Πρόκειται για εργαλείο που δημιουργεί διαδραστικά και σε πραγματικό χρόνο διαγράμματα, όπως το PlantUML. Το εργαλείο αυτό διευκολύνει την κατανόηση και τον σχεδιασμό συστημάτων λογισμικού, ενισχύοντας την οπτικοποίηση και την ανάλυση της αρχιτεκτονικής τους.
gemini	online	N/A	N/A	-
copilot	online	N/A	N/A	-
claude3.5 Sonnet	online	175B	N/A	-

Πίνακας 3.1 Σύνολο επιλεγμένων μοντέλων

Αυτή η ποικιλία μας επιτρέπει να αξιολογήσουμε μοντέλα με διαφορετικά μεγέθη παραμέτρων, από μικρότερα μοντέλα, όπως το **llama3.1**, έως μεγαλύτερα μοντέλα, όπως το **command-r**, προσφέροντας πολύτιμες πληροφορίες για το κατά πόσο το μέγεθος και οι δυνατότητες του μοντέλου επηρεάζουν την έξοδο. Επιπλέον, επιλέξαμε μοντέλα από ένα ευρύ φάσμα εξειδικεύσεων, συμπεριλάβαμε μοντέλα με συγκεκριμένες βελτιστοποιήσεις, όπως το **gpt-4o (Software Architecture Visualizer)**, το οποίο έχει σχεδιαστεί ειδικά για αρχιτεκτονικές εργασίες, αλλά και μοντέλα γενικής χρήσης όπως το **Copilot.** Δοκιμάζοντας αυτή τη αύξουσα σειρά μοντέλων, αφουγκραζόμαστε επίσης το υπαρκτό ενδιαφέρον για τη χρήση μικρότερων, τοπικά φιλοξενούμενων μοντέλων ως εναλλακτική στα μεγαλύτερα, βασισμένα σε cloud, μοντέλα—μια προσέγγιση που ευθυγραμμίζεται με την ανάγκη για ασφάλεια και ιδιωτικότητα δεδομένων, ειδικά σε επιχειρηματικά περιβάλλοντα που διαχειρίζονται ευαίσθητες πληροφορίες.

Όλα τα μοντέλα που επιλέχθηκαν για τη μελέτη αυτή χρησιμοποιούν την παράμετρο κβαντοποίησης Q4, η οποία μειώνει την ακρίβεια των βαρών του μοντέλου σε 4 bits. Η τυποποίηση αυτής της παραμέτρου σε όλα τα μοντέλα αποτέλεσε συνειδητή επιλογή, ιδιαίτερα στα αρχικά στάδια της έρευνάς μας για την AI-υποβοηθούμενη αρχιτεκτονική λογισμικού. Αυτή η προσέγγιση εξασφαλίζει δίκαιη σύγκριση των δυνατοτήτων των μοντέλων, ενώ μας επιτρέπει να εξετάσουμε τα πρακτικά πλεονεκτήματα της χρήσης κβαντισμένων LLMs σε περιβάλλοντα με περιορισμένους πόρους.

Με τη δοκιμή αυτής της ποικιλίας μοντέλων, μπορούμε να παρατηρήσουμε πως η εξειδίκευση ενός μοντέλου (π.χ., γενικού σκοπού μοντέλα έναντι αυτών που έχουν βελτιστοποιηθεί για συγκεκριμένες εφαρμογές) ή το μέγεθος παραμέτρων ενός μοντέλου επηρεάζουν τη διαδικασία δημιουργίας διαγραμμάτων κλάσεων. Αυτή η προσέγγιση επιτρέπει μια ολοκληρωμένη αξιολόγηση, δίνοντάς μας τη δυνατότητα να κάνουμε συγκρίσεις και να εντοπίσουμε ποιοι τύποι μοντέλων ανταποκρίνονται στη δημιουργία υψηλής ποιότητας διαγραμμάτων κλάσεων που ευθυγραμμίζονται με τα καθορισμένα αρχιτεκτονικά πρότυπα.

3.1.5 Retrieval Augmented Generation (RAG)

Τι είναι το RAG;

Τα μεγάλα γλωσσικά μοντέλα (LLMs) συχνά εμφανίζουν απρόβλεπτη συμπεριφορά. Παρόλο που μπορούν να παρέχουν ακριβείς απαντήσεις σε ορισμένες περιπτώσεις, ενδέχεται επίσης να παράγουν άσχετες ή ανακριβείς πληροφορίες, κυρίως επειδή βασίζονται σε στατιστικά μοτίβα της γλώσσας και όχι σε πραγματική κατανόηση. Τα LLMs είναι εκπαιδευμένα να αναγνωρίζουν στατιστικές σχέσεις μεταξύ λέξεων, αλλά τους λείπει η ουσιαστική κατανόηση του νοήματος.

Το **Retrieval-Augmented Generation (RAG)** είναι μία μέθοδος που έχει σχεδιαστεί για να βελτιώνει την ακρίβεια και τη συνάφεια των αποκρίσεων που παράγουν τα LLMs. Βασίζοντας το μοντέλο σε εξωτερικές, ενημερωμένες πηγές γνώσης, το RAG συμπληρώνει την εσωτερική γνώση του LLM με επαληθευμένες πληροφορίες, διασφαλίζοντας ότι οι απαντήσεις είναι ακριβείς και αξιόπιστες.

Η εφαρμογή του RAG σε συστήματα ερωτήσεων-απαντήσεων που βασίζονται σε LLM προσφέρει δύο κύρια πλεονεκτήματα:

1. Παρέχει στο μοντέλο πρόσβαση σε τρέχοντα και επαληθευμένα δεδομένα.
2. Επιτρέπει στους χρήστες να δουν τις πηγές που στηρίζουν τις απαντήσεις του μοντέλου, ενισχύοντας τη διαφάνεια και την επαληθευσιμότητα.

Αυτή η ιχνηλασιμότητα δημιουργεί εμπιστοσύνη, καθώς οι χρήστες μπορούν να ελέγξουν την εγκυρότητα των παρεχόμενων πληροφοριών.

Το RAG προσφέρει και άλλα σημαντικά πλεονεκτήματα:

- Βασίζοντας την έξοδο του LLM σε εξωτερικά, τεκμηριωμένα δεδομένα, μειώνει την πιθανότητα το μοντέλο να βασιστεί αποκλειστικά στις εσωτερικές του παραμέτρους.
- Μειώνει κινδύνους όπως η ακούσια αποκάλυψη ευαίσθητων δεδομένων ή η παραγωγή "παραπλανητικών" (λανθασμένων ή μη αξιόπιστων) πληροφοριών.
- Ελαχιστοποιεί την ανάγκη για συχνή επανεκπαίδευση ή ενημέρωση παραμέτρων του μοντέλου, καθιστώντας τη συντήρηση συστημάτων LLM πιο οικονομική και αποδοτική, ειδικά σε εταιρικά περιβάλλοντα.

Στην ουσία, το RAG λειτουργεί με την ανάκτηση σχετικών δεδομένων από μια εξωτερική πηγή και την εισαγωγή τους στο μεγάλο γλωσσικό μοντέλο μαζί με την προτροπή του χρήστη. Αυτή η εξωτερική πληροφορία εμπλουτίζει τις αποκρίσεις του μοντέλου, ενισχύοντας τη βάση γνώσεών του για την παραγωγή πιο ακριβών και κατάλληλων αποτελεσμάτων, είτε πρόκειται για κείμενο είτε για εικόνες.

Το RAG αποτελεί μια ουσιαστική μέθοδο για τη βελτίωση της ακρίβειας, της διαφάνειας και της αποδοτικότητας των συστημάτων που βασίζονται σε LLMs, προσφέροντας ένα πρακτικό και οικονομικά βιώσιμο τρόπο για την ενίσχυση της αξιοπιστίας τους.

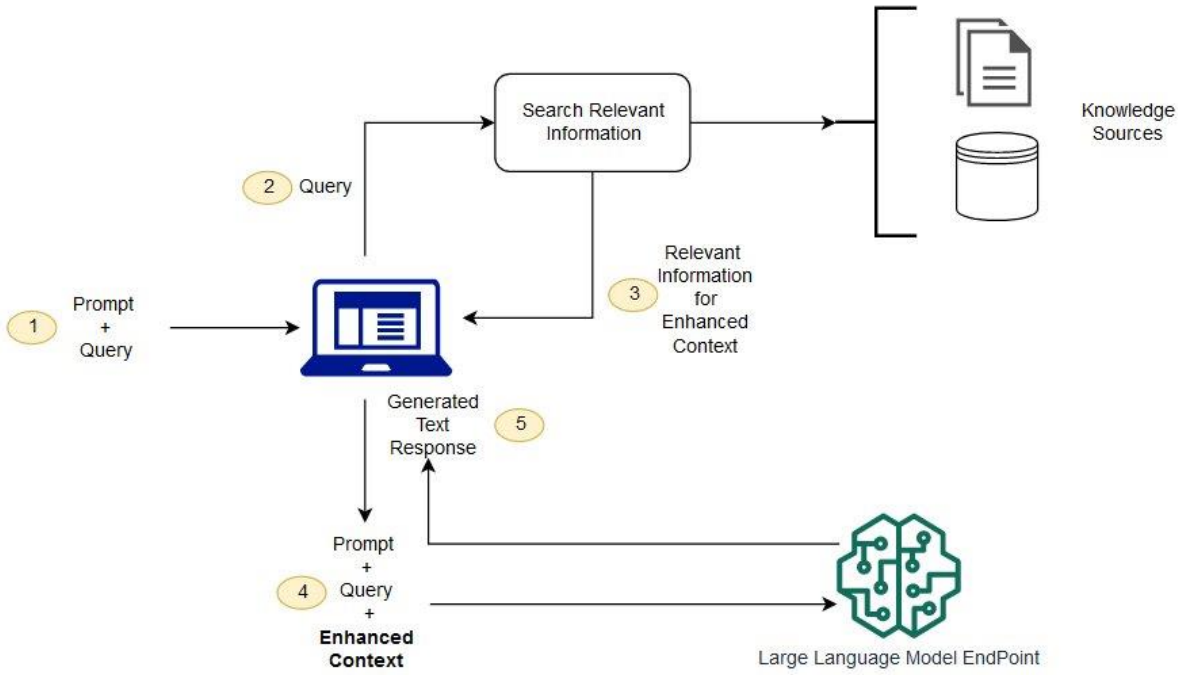
Τι είναι τα Embeddings και πώς χρησιμοποιούνται στο RAG;

Τα embeddings είναι μαθηματικές αναπαραστάσεις που αποτυπώνουν τη σημασιολογική έννοια λέξεων, προτάσεων ή ολόκληρων εγγράφων, κωδικοποιώντας τα ως διανύσματα σε έναν χώρο υψηλών διαστάσεων. Ο σκοπός των embeddings είναι να τοποθετούν κείμενα με παρόμοιες σημασίες κοντά το ένα στο άλλο στον διανυσματικό χώρο, διευκολύνοντας την αποδοτική σύγκριση εννοιών με βάση τη σημασία τους και όχι την ακριβή διατύπωση.

Στο πλαίσιο του **Retrieval-Augmented Generation (RAG)**, τα embeddings διαδραματίζουν κρίσιμο ρόλο στην ανάκτηση σχετικών πληροφοριών από ένα μεγάλο σύνολο δεδομένων βάσει του ερωτήματος του χρήστη. Οι πληροφορίες που ανακτώνται χρησιμοποιούνται για να βοηθήσουν το μοντέλο να δημιουργήσει απαντήσεις με πλούσιο και ακριβές περιεχόμενο. Τα στάδια του RAG για τη χρήση embeddings είναι τα ακόλουθα:

1. **Δημιουργία Embeddings για το Σύνολο Δεδομένων:**
   Πριν από την επεξεργασία οποιουδήποτε ερωτήματος, δημιουργούνται embeddings για κάθε έγγραφο ή τμήμα κειμένου του συνόλου δεδομένων. Αυτή η διαδικασία χαρτογραφεί ολόκληρο το σύνολο δεδομένων σε έναν διανυσματικό χώρο υψηλών διαστάσεων, επιτρέποντας την ομαδοποίηση παρόμοιων εγγράφων με βάση τη σημασία τους.
2. **Δημιουργία Embedding του Ερωτήματος:**
   Όταν ένας χρήστης υποβάλλει ένα ερώτημα, το κείμενο του ερωτήματος μετατρέπεται επίσης σε ένα embedding. Αυτό το embedding του ερωτήματος τοποθετείται στον ίδιο διανυσματικό χώρο με τα embeddings του συνόλου δεδομένων, αντιπροσωπεύοντας τη σημασιολογική πρόθεση της εισόδου του χρήστη.
3. **Ανάκτηση βάσει Ομοιότητας:**
   Το σύστημα συγκρίνει το embedding του ερωτήματος με όλα τα embeddings των εγγράφων στον διανυσματικό χώρο. Χρησιμοποιώντας μετρήσεις ομοιότητας (όπως η συσχέτιση συνημιτόνου - cosine similarity), εντοπίζει και ανακτά τα έγγραφα που ευθυγραμμίζονται περισσότερο με τη σημασία του ερωτήματος.
4. **Ενίσχυση prompt με Ανακτημένες Πληροφορίες:**
   Τα έγγραφα που ανακτήθηκαν λειτουργούν ως πρόσθετη γνώση ή πλαίσιο για το μοντέλο, το οποίο στη συνέχεια δημιουργεί μια απάντηση βασισμένη σε πραγματικά δεδομένα. Αυτή η ενίσχυση επιτρέπει στο μοντέλο να παρέχει πιο ακριβείς, λεπτομερείς και συμφραζόμενες απαντήσεις.

Μέσω των embeddings, το RAG μπορεί να ανακτά και να αξιοποιεί αποτελεσματικά σημασιολογικά σχετικές πληροφορίες, βελτιώνοντας την ποιότητα και την αξιοπιστία των απαντήσεων του μοντέλου. Αυτή η διαδικασία εξασφαλίζει ότι οι αποκρίσεις του μοντέλου βασίζονται σε εξωτερικό, ουσιαστικό περιεχόμενο, ενισχύοντας τη συνάφεια και την ακρίβειά τους.



Εικόνα 3.6 Conceptual flow of using RAG with LLMs (πηγή)

Τεχνικές για την Ενίσχυση του Retrieval Augmented Generation (RAG)

Καθώς το παραγωγικό AI εξελίσσεται ταχύτατα, ερευνητές και επαγγελματίες εξετάζουν διάφορες τεχνικές για τη βελτίωση της απόδοσης των μεγάλων γλωσσικών μοντέλων (LLMs) σε εργασίες **Retrieval-Augmented Generation (RAG)**. Συνδυάζοντας τα πλεονεκτήματα των γλωσσικών μοντέλων με συστήματα ανάκτησης πληροφορίας (IR), το RAG επιτρέπει στα LLMs να παράγουν πιο τεκμηριωμένες και συμφραζόμενες απαντήσεις. Ακολουθούν ορισμένες βασικές τεχνικές για τη βελτίωση της αποτελεσματικότητας των RAG συστημάτων που βασίζονται σε LLM:

1. Επιλογή του Κατάλληλου Μοντέλου Embedding

Η αποτελεσματικότητα της ανάκτησης μέσω αναζήτησης ομοιότητας επηρεάζεται σημαντικά από την ποιότητα των embeddings που αναπαριστούν τα έγγραφα και τα ερωτήματα. Η σωστή επιλογή ενός μοντέλου embedding είναι κρίσιμη, καθώς επηρεάζει άμεσα την ακρίβεια ανάκτησης και, κατά συνέπεια, τη συνάφεια των αποκρίσεων που παράγονται από το LLM. Διαθέσιμες επιλογές για embedding μοντέλα περιλαμβάνουν:

- Self-hosted ενός μοντέλου ανοιχτού κώδικα.
- Χρήση μιας cloud υπηρεσίας που προσφέρει μοντέλα ανοιχτού κώδικα.
- Επιλογή μιας λύσης cloud (π.χ. OpenAI).
- Χρήση μιας ολοκληρωμένης λύσης, όπως το Enterprise Bot, που περιλαμβάνει υπηρεσίες embeddings.

Άξονες για την αξιολόγηση του κατάλληλου embedding μοντέλου:

- **Μέσος όρος απόδοση ανάκτησης:** Μέτρηση της ακρίβειας του μοντέλου σε benchmarks που εστιάζουν στην ανάκτηση για εργασίες RAG.

- **Μέγεθος μοντέλου και χρήση μνήμης:** Τα μεγαλύτερα μοντέλα συχνά προσφέρουν καλύτερη ακρίβεια αλλά απαιτούν περισσότερους πόρους.
- **Διαστάσεις embedding:** Τα embeddings με περισσότερες διαστάσεις αποτυπώνουν περισσότερες σημασιολογικές λεπτομέρειες, αλλά αυξάνουν τις απαιτήσεις μνήμης.
- **Μέγιστος αριθμός tokens:** Δείχνει το μέγιστο μήκος κειμένου που μπορεί να χειριστεί ένα embedding, κρίσιμο για εκτενή ερωτήματα.

2. Βελτιστοποίηση Στρατηγικής Τμηματοποίησης (Chunking)

Η τμηματοποίηση (chunking) αφορά τη διάσπαση των εγγράφων ή δεδομένων σε μικρότερες μονάδες, ενισχύοντας την αποδοτικότητα και ακρίβεια της ανάκτησης. Αποτελεσματικές στρατηγικές chunking βοηθούν τα RAG συστήματα να διαχειρίζονται μεγαλύτερες εισόδους κειμένου και να παράγουν πιο συνεκτικές, συμφραζόμενες απαντήσεις. Συνήθεις στρατηγικές περιλαμβάνουν:

- **Τμηματοποίηση με ολίσθηση παραθύρου (Sliding Window):** Διαιρεί το κείμενο σε επικαλυπτόμενα παράθυρα για διατήρηση του συμφραζομένου.
- **Τμηματοποίηση βάσει εγγράφου (Document-Based):** Διαιρεί το κείμενο με βάση τα όρια του εγγράφου.
- **Σημασιολογική Τμηματοποίηση (Semantic Chunking):** Διαχωρίζει το κείμενο με βάση τη σημασία, ομαδοποιώντας σχετικές πληροφορίες.
- **Τμηματοποίηση βάσει πρακτόρων (Agent-Based):** Σχεδιασμένη για αλληλεπιδράσεις πολλαπλών πρακτόρων.

Πειραματιζόμενοι με αυτές τις στρατηγικές, μπορεί να επιτευχθεί η ισορροπία μεταξύ ακρίβειας ανάκτησης, υπολογιστικής αποδοτικότητας και ποιότητας αποκρίσεων.

3. Εφαρμογή Φίλτρων Μεταδεδομένων

Σε πολλές εφαρμογές RAG, τα έγγραφα στη βάση γνώσεων περιέχουν χρήσιμα μεταδεδομένα, όπως αξιοπιστία πηγών, χρονικές σημάνσεις ή θεματική συνάφεια. Η αξιοποίηση αυτών των μεταδεδομένων βελτιώνει την ποιότητα και τη συνάφεια των αποκρίσεων, επιτρέποντας στο σύστημα να φιλτράρει ή να δίνει προτεραιότητα σε πληροφορίες με βάση προκαθορισμένα κριτήρια. Για παράδειγμα:

- Προτεραιότητα σε πρόσφατες ή αξιόπιστες πηγές.
- Φιλτράρισμα βάσει θεμάτων για ειδικά πεδία, όπως η υγειονομική περίθαλψη ή τα νομικά ζητήματα.

Η χρήση φίλτρων μεταδεδομένων βοηθά στην επιλογή των πιο αξιόπιστων και συναφών πληροφοριών, ενισχύοντας την ακρίβεια των αποκρίσεων.

4. Εφαρμογή Τεχνικών Μετασχηματισμού Ερωτημάτων (Query Transformation)

Ορισμένες φορές, η εισαγωγή του χρήστη δεν ευθυγραμμίζεται με τη δομή της βάσης γνώσεων ή τις προσδοκίες του LLM. Οι τεχνικές μετασχηματισμού ερωτημάτων προσαρμόζουν και βελτιστοποιούν την εισαγωγή του χρήστη για καλύτερη συνάφεια και ακρίβεια ανάκτησης. Παραδείγματα περιλαμβάνουν:

- Επαναδιατύπωση ή επέκταση του ερωτήματος για καλύτερη σύλληψη του συμφραζομένου.
- Εξαγωγή λέξεων-κλειδιών ή οντοτήτων για ακριβέστερη ανάκτηση.

- Μετάφραση όρων σε συγκεκριμένη γλώσσα ή τεχνική ορολογία.
- Προσθήκη περιορισμών ανάκτησης που σχετίζονται με το έργο.

5. Βελτίωση Κατάταξης (ReRanking)

Η αρχική ανάκτηση σε ένα σύστημα RAG δεν διασφαλίζει πάντα ότι τα πιο σχετικά έγγραφα εμφανίζονται στις πρώτες θέσεις των αποτελεσμάτων. Οι τεχνικές ReRanking επιλύουν αυτό το ζήτημα αναδιατάσσοντας τα ανακτηθέντα έγγραφα για βελτίωση της συνάφειας.

- **Αξιοποίηση πρόσθετων παραμέτρων:** Χρήση σημασιολογικής ομοιότητας, αξιοπιστίας πηγών ή θεματικής συνάφειας για την αξιολόγηση και αναδιάταξη των εγγράφων.
- **Εφαρμογή Προηγμένων Μοντέλων Κατάταξης:** Χρήση προσεγγίσεων μηχανικής μάθησης, όπως νευρωνικά μοντέλα κατάταξης ή ενισχυτική μάθηση (reinforcement learning), για την εκμάθηση βέλτιστων κριτηρίων κατάταξης.
- **Ενσωμάτωση Ανατροφοδότησης Χρήστη:** Ενσωμάτωση δεδομένων αλληλεπίδρασης ή ανατροφοδότησης για βελτίωση των κριτηρίων κατάταξης με την πάροδο του χρόνου.

Κάθε μία από αυτές τις τεχνικές στοχεύει να επιλύσει συγκεκριμένες προκλήσεις στη μέθοδο RAG, όπως η ακρίβεια ανάκτησης, η βελτιστοποίηση ερωτημάτων και η κατάταξη εγγράφων. Η εφαρμογή αυτών των βελτιώσεων μπορεί να αυξήσει σημαντικά την ποιότητα των αποκρίσεων, διασφαλίζοντας ότι τα αποτελέσματα είναι πιο αξιόπιστα, συναφή και συμφραζόμενα. Καθώς η χρήση του RAG επεκτείνεται σε διάφορους κλάδους, αυτές οι βελτιώσεις συμβάλλουν στη δημιουργία πιο ανθεκτικών, αποδοτικών και αποτελεσματικών λύσεων που βασίζονται στην τεχνητή νοημοσύνη.

Παρόλο που όλες οι παραπάνω τεχνικές θα μπορούσαν να βελτιώσουν τη διαδικασία του RAG, το πείραμά μας επικεντρώνεται σε δύο βασικές τεχνικές: **την επιλογή μοντέλου embedding** και **τη στρατηγική τμηματοποίησης (chunking strategy)**. Εστιάζοντας σε αυτές τις συγκεκριμένες τεχνικές, στοχεύουμε να κατανοήσουμε καλύτερα την επίδρασή τους στην ακρίβεια ανάκτησης και στην ποιότητα των παραγόμενων αποκρίσεων.

Μοντέλα Embedding

Για τα μοντέλα embedding, επιλέξαμε τις εξής δύο επιλογές που παρουσιάζονται στον Πίνακας 3.2, οι οποίες αντιπροσωπεύουν διαφορετικές προσεγγίσεις σε απόδοση και διαστασιολόγηση embeddings:

- **nomic-embed-text-v1.5:**
  Ένα τοπικό μοντέλο βελτιστοποιημένο για γενικής χρήσης embeddings, το οποίο αποδίδει εξαιρετικά στην αποτύπωση ευρείας σημασιολογικής σχέσης τόσο για σύντομα όσο και για μακροσκελή συμφραζόμενα.
- **text-embedding-3-large:**
  Ένα μοντέλο που παράγει μεγαλύτερα διανύσματα embedding και μπορεί να προσφέρει πιο λεπτομερείς, ακριβείς αναπαραστάσεις που βελτιώνουν την ακρίβεια ανάκτησης σε πιο σύνθετα ερωτήματα.

Όνομα Embedding Μοντέλου	Provider	Μέγεθος Παραμέτρων	Context window length used to generate the next token (num_ctx)	Διαστάσεις	Εξειδικεύσεις μοντέλου
nomic-embed-text-v1.5	nomic-ai	137M	8192 tokens	768	Μπορεί να χρησιμοποιηθεί για τη δημιουργία embeddings για σύντομο και μακρύ context
text-embedding-3-large	openai	N/A	8192 tokens	3072	Υψηλής ακρίβειας εργασίες: Ιδανικό για περιπτώσεις όπου η αποτύπωση των λεπτομερειών της γλώσσας είναι κρίσιμη. Κατάλληλο για σύνθετες εφαρμογές

Πίνακας 3.2 Σύνολο επιλεγμένων embedding μοντέλων

Με τη χρήση των δύο αυτών μοντέλων embedding, στοχεύουμε να αξιολογήσουμε κατά πόσο η επιλογή μοντέλου επηρεάζει τα αποτελέσματα του RAG, ιδιαίτερα σε ό,τι αφορά τη συνάφεια της ανάκτησης και την ευθυγράμμιση με τις καθορισμένες αρχιτεκτονικές. Αυτή η σύγκριση παρέχει πολύτιμες γνώσεις σχετικά με τον ρόλο της επιλογής embedding στη βελτίωση του σχεδιασμού της αρχιτεκτονικής με τη βοήθεια των LLMs.

Μέθοδοι Τμηματοποίησης (Chunking Methods)

Για το πείραμά μας, επιλέξαμε δύο μεθόδους τμηματοποίησης με στόχο τη βελτιστοποίηση της ανάκτησης και της συνοχής των τμημάτων κειμένου κατά τη διαδικασία RAG: την **αναδρομική τμηματοποίηση (recursive chunking)** και την **σημασιολογική τμηματοποίηση (semantic chunking)**.

Μέσα από τον πειραματισμό με αυτές τις δύο διακριτές προσεγγίσεις, μπορούμε να εξερευνήσουμε πώς διαφορετικές στρατηγικές τμηματοποίησης επηρεάζουν τη συνάφεια και τη συνοχή του ανακτηθέντος περιεχομένου.

Αναδρομική Τμηματοποίηση Κατά Χαρακτήρα (Recursive Character Split)

Αυτή η μέθοδος συνιστάται ευρέως για την επεξεργασία γενικού κειμένου. Χρησιμοποιεί μια προτεραιοποιημένη λίστα χαρακτήρων, επιχειρώντας να διαχωρίσει το κείμενο με κάθε χαρακτήρα διαδοχικά, μέχρι τα τμήματα να φτάσουν στο επιθυμητό μέγεθος. Η προεπιλεγμένη λίστα χαρακτήρων είναι: ["\n\n", "\n", " ", ""]. Η μέθοδος προσπαθεί πρώτα να διατηρήσει παραγράφους, μετά προτάσεις, και τέλος λέξεις. Με τη διατήρηση αυτών των μεγαλύτερων μονάδων όπου είναι δυνατόν, η μέθοδος διατηρεί σημασιολογικά συναφή τμήματα κειμένου, βελτιώνοντας την ακρίβεια ανάκτησης μέσω της διατήρησης σχετικών εννοιών στο ίδιο τμήμα.

Σημασιολογική Τμηματοποίηση (Semantic Chunking)

Η σημασιολογική τμηματοποίηση έχει σχεδιαστεί για να εξάγει και να διατηρεί το νόημα, αξιολογώντας τις σημασιολογικές σχέσεις μεταξύ των τμημάτων. Αντί να διαχωρίζει το κείμενο αποκλειστικά με βάση χαρακτήρες, η μέθοδος κρατά μαζί τμήματα που συνδέονται στενά νοηματικά. Αυτή η προσέγγιση είναι ιδιαίτερα χρήσιμη για να διασφαλίσει ότι κάθε τμήμα αντιπροσωπεύει μια συνεκτική ιδέα ή θέμα. Ο στόχος είναι να παραχθούν πιο ακριβείς και συμφραζόμενες ανακτήσεις. Για αυτή τη διαδικασία η μέθοδος Semantic αξιοποιεί κάποιο embedding model.

Για την υλοποίηση, χρησιμοποιήθηκαν συναρτήσεις από το Langchain framework της Python:

- **RecursiveCharacterTextSplitter:** Για την αναδρομική τμηματοποίηση χαρακτήρων.
- **SemanticChunker:** Για τη σημασιολογική τμηματοποίηση.

Με τη σύγκριση αυτών των δύο μεθόδων τμηματοποίησης, στοχεύουμε να παρατηρήσουμε πώς η στρατηγική τμηματοποίησης επηρεάζει τη συνοχή και τη συνάφεια της ανακτηθείσας πληροφορίας. Τελικά, αυτά τα ευρήματα θα επηρεάσουν την ποιότητα των αποτελεσμάτων που παράγονται από τα LLMs, προσφέροντας χρήσιμες πληροφορίες για τη βελτιστοποίηση του RAG στις εφαρμογές που βασίζονται στο AI.

Πηγές Εγγράφων RAG: Περιεχόμενο και Επίδραση στα Αποτελέσματα

Για να ενισχύσουμε την απόδοση του μοντέλου στη δημιουργία διαγραμμάτων κλάσεων, παρέχουμε στο LLM καλά τεκμηριωμένες παρουσιάσεις των ζητούμενων αρχιτεκτονικών. Μέσω δομημένων RAG αρχείων, επιδιώκουμε όχι μόνο να βελτιώσουμε την ποιότητα των παραγόμενων αποτελεσμάτων, αλλά και να εξετάσουμε πώς οι διαφοροποιήσεις στην πηγή και τη δομή της πληροφορίας επηρεάζουν τα αποτελέσματα. Για τον σκοπό αυτό, δημιουργήσαμε τρεις τύπους RAG αρχείων, το καθένα με διαφορετικό περιεχόμενο ή μορφή, για να αξιολογήσουμε την επίδρασή τους στο τελικό αποτέλεσμα.

1. **RAG Αρχείο από το "Software Engineering" του Ian Sommerville**
   Το πρώτο RAG αρχείο αντλείται από μια θεμελιώδη πηγή: το βιβλίο **Software Engineering, 10th Edition** του Ian Sommerville. Το βιβλίο αυτό αποτελεί σταθμό στον τομέα της Τεχνολογίας Λογισμικού και καλύπτει εις βάθος τις βασικές πρακτικές της αρχιτεκτονικής λογισμικού. Χρησιμοποιώντας περιεχόμενο από μια τόσο έγκυρη και αξιόπιστη πηγή, αναμένουμε ότι το μοντέλο θα αποκτήσει μια πλήρη και δομημένη κατανόηση, οδηγώντας σε ακριβέστερα διαγράμματα κλάσεων. Η ενσωμάτωση υλικού από το βιβλίο αυτό μας επιτρέπει να εξετάσουμε πώς μια αυστηρά ακαδημαϊκή και δομημένη βάση γνώσεων επηρεάζει την ικανότητα του μοντέλου να παράγει διαγράμματα σύμφωνα με τις αρχιτεκτονικές απαιτήσεις.

2. **Διαχωρισμένα Έγγραφα Αρχιτεκτονικής**
   Στη δεύτερη προσέγγιση, εξακολουθούμε να χρησιμοποιούμε το βιβλίο του Sommerville ως πηγή πληροφορίας, αλλά τροποποιούμε τη μορφή. Ειδικότερα, χωρίζουμε την τεκμηρίωση για κάθε τύπο αρχιτεκτονικής σε ξεχωριστά έγγραφα. Αυτή η μέθοδος επιτρέπει τη σύνδεση του RAG αρχείου με την αρχιτεκτονική που ζητείται στο prompt, παρέχοντας στοχευμένη και εξειδικευμένη πληροφορία. Ο διαχωρισμός των αρχιτεκτονικών σε ξεχωριστά έγγραφα στοχεύει στη βελτίωση της ακρίβειας και της συνάφειας του παραγόμενου διαγράμματος, επιτρέποντας μια πιο λεπτομερή και προσαρμοσμένη προσέγγιση για κάθε αρχιτεκτονικό πρότυπο.

3. **RAG Αρχείο από Διάφορες Διαδικτυακές Πηγές**
   Το τρίτο RAG αρχείο δημιουργήθηκε από διάφορες διαδικτυακές πηγές, όπως το Wikipedia, το GeeksforGeeks, και την τεκμηρίωση της IBM. Αυτό το αρχείο προσφέρει στο μοντέλο ένα ευρύ φάσμα απόψεων και εξηγήσεων, παρέχοντας μια ποικιλία πληροφοριών που συχνά απαντώνται σε λιγότερο επίσημες πηγές. Ο συνδυασμός δεδομένων από πολλαπλές διαδικτυακές πηγές αντικατοπτρίζει τη διαφορετικότητα των εξηγήσεων και των αποχρώσεων που συναντώνται σε ανεπίσημες, αλλά εύκολα προσβάσιμες πηγές. Η δοκιμή αυτού του αρχείου μας επιτρέπει να διερευνήσουμε αν η χρήση ποικιλόμορφης πληροφορίας επηρεάζει τη συνοχή και την ακρίβεια των παραγόμενων διαγραμμάτων κλάσεων, σε σύγκριση με την αυστηρότητα της ακαδημαϊκής προσέγγισης.

Συνολικά, η πειραματική αξιολόγηση αυτών των τριών τύπων RAG αρχείων αποσκοπεί στην κατανόηση του τρόπου με τον οποίο οι διαφορές στην πηγή περιεχομένου και τη δομή του εγγράφου επηρεάζουν την ποιότητα και την πιστότητα των παραγόμενων διαγραμμάτων κλάσεων. Αυτή η ανάλυση θα μας προσφέρει πολύτιμες γνώσεις σχετικά με τη βέλτιστη μέθοδο ενίσχυσης των LLMs με εξωτερική γνώση για σύνθετες αρχιτεκτονικές εργασίες.

3.2 Σχεδιασμός Πειράματος

3.2.1 Διαγράμματα αρχιτεκτονικής ως σημείο αναφοράς

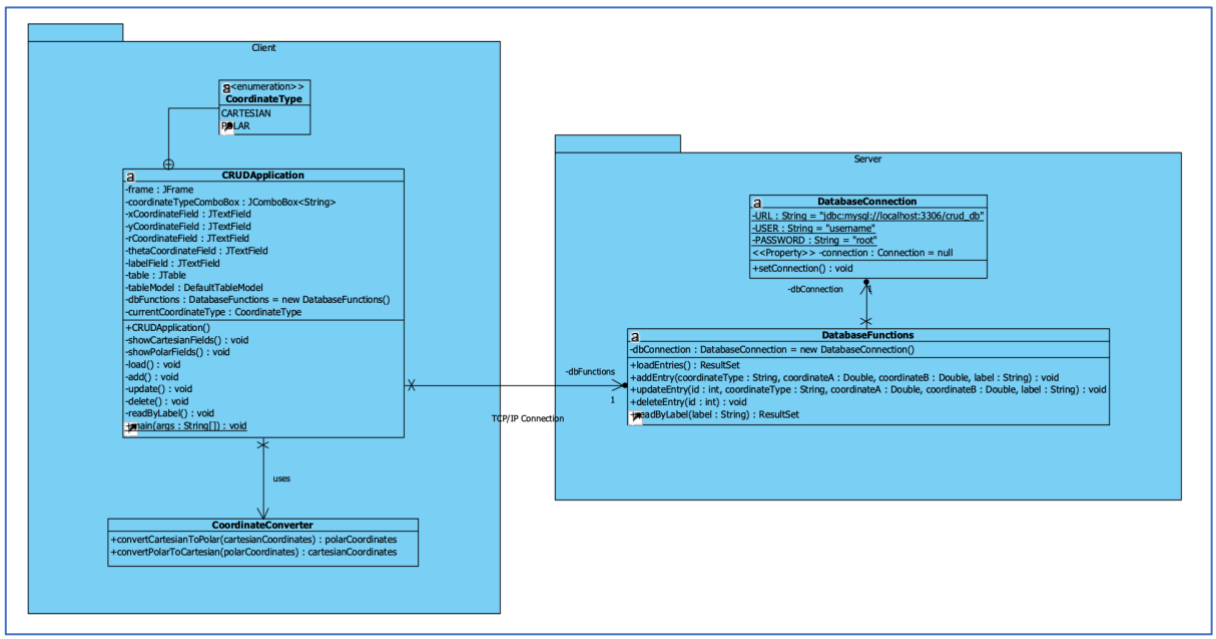
Για να αποκτήσουμε μια ολοκληρωμένη κατανόηση του τρόπου με τον οποίο κάθε αρχιτεκτονικό πρότυπο επηρεάζει την εφαρμογή μας, υλοποιήσαμε τρεις διακριτές εκδόσεις της εφαρμογής σε **Java**, καθεμία από τις οποίες βασίζεται σε ένα από τα επιλεγμένα αρχιτεκτονικά πρότυπα: **Πελάτης-Εξυπηρετητής**, **Τριών Επιπέδων** και **Model-View-Controller**. Αυτή η προσέγγιση μας επιτρέπει να παρατηρήσουμε πώς κάθε αρχιτεκτονική επηρεάζει τη δομή, τη ροή δεδομένων και τη διαμόρφωση των μονάδων εντός της εφαρμογής.

Σε κάθε έκδοση, φροντίσαμε να τηρούμε αυστηρά τις αρχές και τις κατευθυντήριες γραμμές που σχετίζονται με κάθε αρχιτεκτονικό πρότυπο, όπως περιγράφονται στην ενότητα 2.2. Αυτό περιλάμβανε την εξασφάλιση σαφούς διαχωρισμού ευθυνών και συνέπειας στη διαχείριση και την αλληλεπίδραση των δεδομένων μεταξύ των συστατικών στοιχείων. Τηρώντας αυτές τις βασικές αρχιτεκτονικές αρχές, στοχεύουμε να δημιουργήσουμε μια βάση αναφοράς, με την οποία μπορούν να συγκριθούν τα παραγόμενα αποτελέσματα που προκύπτουν από τις αποκρίσεις των μοντέλων.

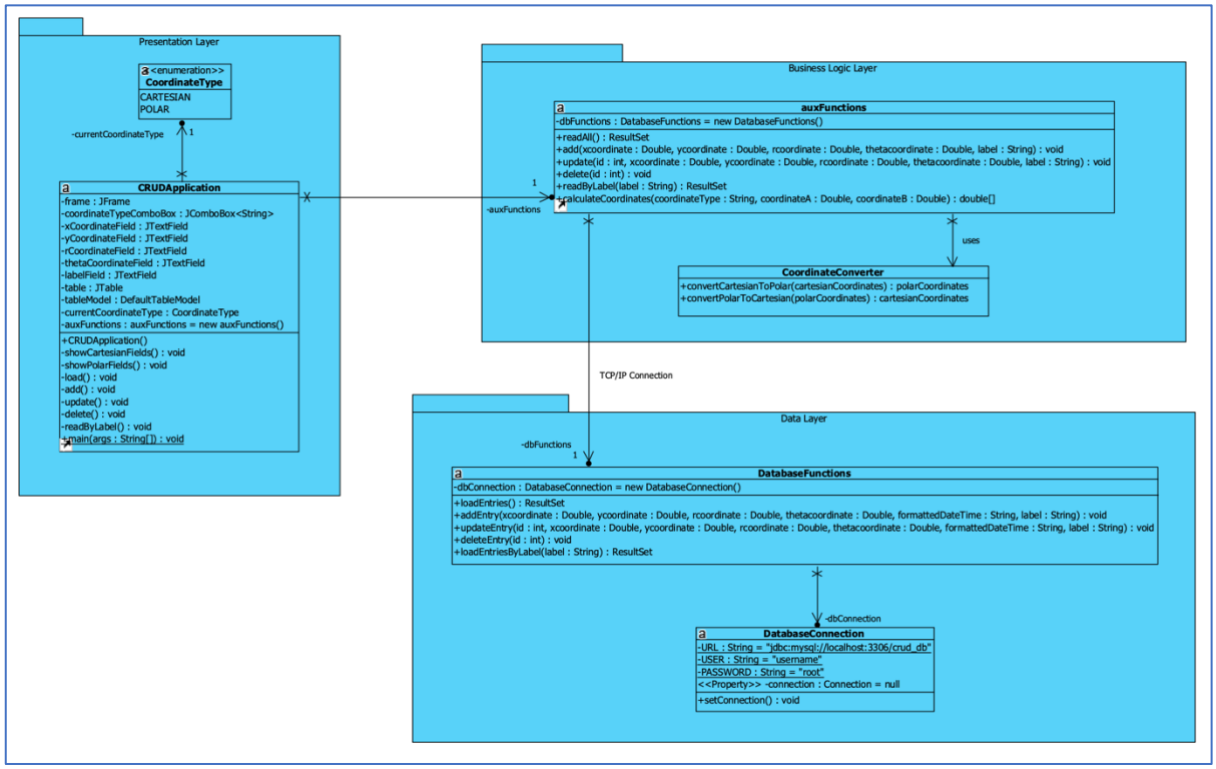
Μετά την υλοποίηση κάθε έκδοσης, χρησιμοποιήσαμε το εργαλείο **Visual Paradigm** για την αντίστροφη παραγωγή του κώδικα Java σε διαγράμματα κλάσεων UML. Αυτή η διαδικασία μας επέτρεψε να δημιουργήσουμε οπτικές αναπαραστάσεις κάθε αρχιτεκτονικής, καταγράφοντας με ακρίβεια τις σχέσεις, τις εξαρτήσεις και τις ιεραρχίες εντός του υλοποιημένου κώδικα.

Στις Εικόνα 3.7 , Εικόνα 3.8 και Εικόνα 3.9 παρουσιάζονται τα διαγράμματα κλάσεων UML που προέκυψαν από τις υλοποιήσεις μας. Αυτά τα διαγράμματα χρησιμεύουν ως σημείο αναφοράς για την αξιολόγηση των διαγραμμάτων που παράγονται από τα μοντέλα σε σύγκριση με τα ζητούμενα αρχιτεκτονικά πρότυπα. Παρέχουν μια καθαρή, δομημένη εικόνα των συστατικών στοιχείων, δείχνοντας πώς κάθε επιλογή αρχιτεκτονικής επηρεάζει την οργάνωση και την αλληλεπίδραση των κλάσεων εντός της εφαρμογής

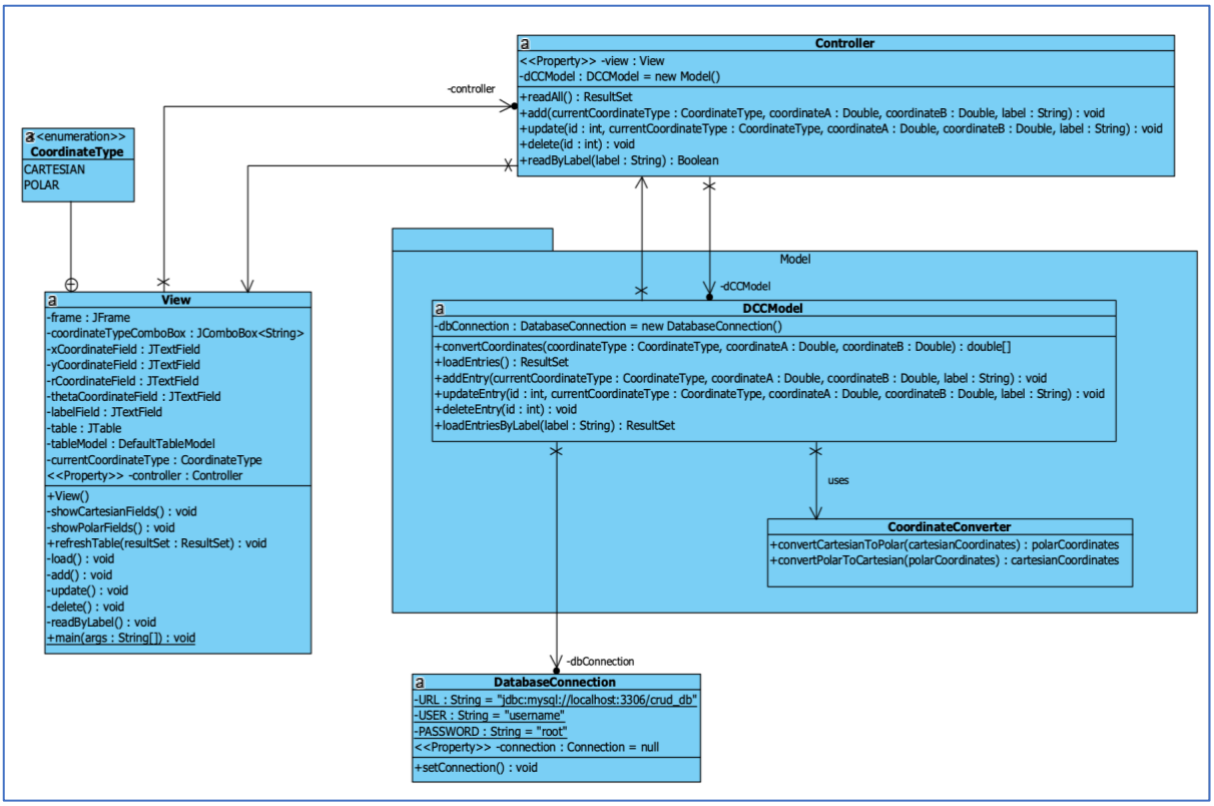
Παρόλο που δεν αποζητάμε τα διαγράμματα που δημιουργούνται από τα LLMs να είναι απόλυτα ταυτόσημα με αυτά, αποτελούν ένα σημείο αναφοράς για σύγκριση. Είμαστε ανοιχτοί σε νέες προσεγγίσεις, υπό την προϋπόθεση όμως ότι ευθυγραμμίζονται με τα καθιερωμένα αρχιτεκτονικά πρότυπα και τις αρχές σχεδίασης.



Εικόνα 3.7 Διάγραμμα κλάσεων του DCC σε Client-Server



Εικόνα 3.8 Διάγραμμα κλάσεων του DCC σε Three-Tier



Εικόνα 3.9 Διάγραμμα κλάσεων του DCC σε MVC

3.2.2 Εργαλείο PlantUML

Η χρήση ενός model-oriented εργαλείου όπως το **Visual Paradigm** για τη δημιουργία και αξιολόγηση διαγραμμάτων UML από τα LLMs παρουσίασε ορισμένες τεχνικές προκλήσεις. Συγκεκριμένα, η συνέχιση με αυτό το εργαλείο απαιτούσε δύο εναλλακτικές προσεγγίσεις, καθεμία με περιορισμούς.

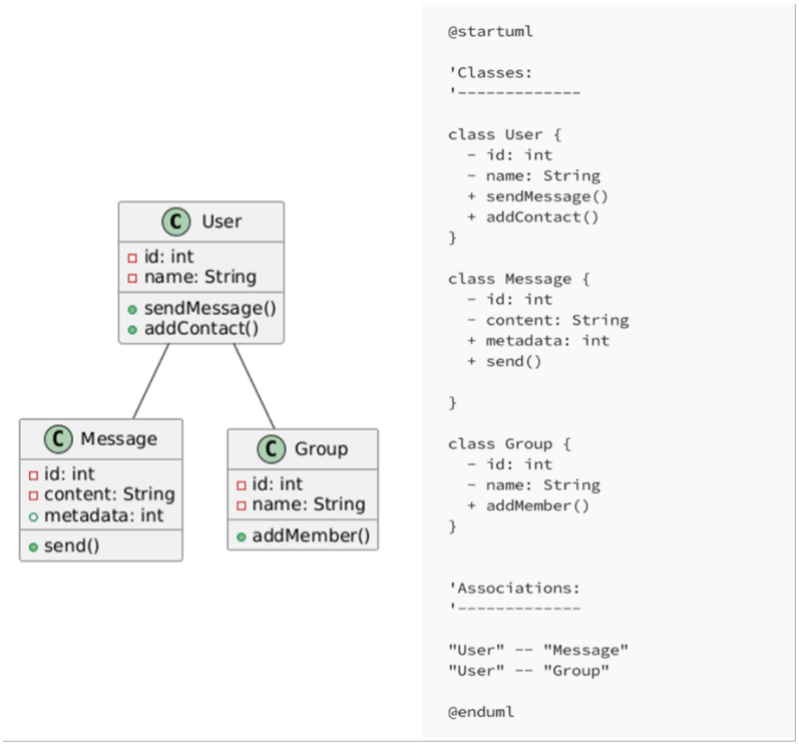
Στην πρώτη προσέγγιση, θα μπορούσαμε να εργαστούμε με διαγράμματα κλάσεων σε μορφή εικόνας. Ωστόσο, αυτό θα μας περιόριζε στη χρήση μόνο μεγάλων γλωσσικών μοντέλων (LLMs) που μπορούν να δημιουργούν και να αναγνωρίζουν εικόνες, απομακρύνοντας τη ροή εργασίας από τους αρχικούς στόχους του έργου. Επιπλέον, η εξάρτηση από την αναγνώριση εικόνων εισάγει προκλήσεις, όπως μειωμένη ακρίβεια και αυξημένη δυσκολία στην επεξεργασία και επαλήθευση των στοιχείων του διαγράμματος.

Η δεύτερη προσέγγιση αφορούσε τη δημιουργία διαγραμμάτων κλάσεων από τα LLMs σε μορφή XML, τα οποία θα μπορούσαμε να εισαγάγουμε στο Visual Paradigm. Ωστόσο, αυτή η επιλογή είχε επίσης προβλήματα. Τα διαγράμματα κλάσεων σε XML απαιτούν σύνθετη σύνταξη, η οποία μπορεί να διαφέρει μεταξύ διαφορετικών πλατφορμών, οδηγώντας σε πιθανά ζητήματα συμβατότητας και ασυνέπειες. Επιπλέον, η δημιουργία XML από LLMs μπορεί να είναι δύσκολη, ενδεχομένως οδηγώντας σε σφάλματα ή ελλιπή διαγράμματα.

Για να αντιμετωπίσουμε αυτές τις προκλήσεις, επιλέξαμε τη χρήση του PlantUML. Το PlantUML συνδυάζει τη δυνατότητα δημιουργίας λεπτομερών και καλά δομημένων διαγραμμάτων UML με την απλότητα μιας γλώσσας βασισμένης σε κώδικα, που τα LLMs μπορούν εύκολα να παράγουν και να ερμηνεύσουν. Αυτή η επιλογή μας επέτρεψε να αποφύγουμε τα προβλήματα συμβατότητας του XML, καθώς και τις προκλήσεις της αναγνώρισης εικόνων, διατηρώντας μια

ομαλή ροή εργασίας. Η απλή σύνταξη του PlantUML διασφαλίζει ότι οποιοδήποτε LLM με βασικές δυνατότητες παραγωγής κειμένου και κώδικα μπορεί να δημιουργεί και να αναγνωρίζει τα διαγράμματα, καθιστώντας το μια ιδανική λύση για τις ανάγκες μας.

Το PlantUML προσφέρει επίσης ποικιλία επιλογών για την οπτικοποίηση διαγραμμάτων UML. Υποστηρίζει πολλαπλές πλατφόρμες και εργαλεία, συμπεριλαμβανομένου ενός επίσημου web server που λειτουργεί ως κειμενογράφος, καθώς και βιβλιοθήκες για Java, Python, και React. Στα πλαίσια της δικιάς μου εργασίας, αξιοποιήσαμε το αρχείο .jar του PlantUML για Java, το οποίο μας επέτρεψε να αποδώσουμε και να προβάλουμε τα διαγράμματα απευθείας ως εικόνες. Αυτή η λειτουργικότητα παρείχε έναν αποδοτικό τρόπο οπτικοποίησης και αξιολόγησης των παραγόμενων διαγραμμάτων κλάσεων, επιταχύνοντας τη ροή εργασίας μας μέσω της γρήγορης μετατροπής από κώδικα PlantUML σε μορφή εικόνας για ανάλυση.



Εικόνα 3.10 Παράδειγμα plantUML

3.2.3 Ανάπτυξη, Εργαλεία και Στήσιμο

Για την ανάπτυξη και την εκτέλεση του πειράματός μας, χρησιμοποιήσαμε έναν server Ubuntu 22.04 εξοπλισμένο με 128 GB RAM. Αυτή η διαμόρφωση παρείχε τους απαραίτητους υπολογιστικούς πόρους για τη διαχείριση μεγάλων γλωσσικών μοντέλων (LLMs) και της εκτεταμένης επεξεργασίας που απαιτείται για τη δημιουργία και αξιολόγηση διαγραμμάτων κλάσεων UML σε διάφορα σενάρια.

Για την αλληλεπίδραση με τα τοπικά LLMs, χρησιμοποιήσαμε το **Ollama framework**, μια χρήσιμη πλατφόρμα που έχει σχεδιαστεί για τη διαχείριση και την εκτέλεση μεγάλων γλωσσικών μοντέλων, όπως το Llama 3.1 και το Mistral, σε τοπικές μηχανές. Το Ollama παρέχει ευέλικτη ενσωμάτωση προηγμένων δυνατοτήτων AI, προσφέροντας στους προγραμματιστές εργαλεία για την κλήση και τη διαχείριση μοντέλων.

Το Ollama υποστηρίζει πολλά λειτουργικά συστήματα, συμπεριλαμβανομένων των Windows, macOS, και Linux, και προσφέρει διάφορες επιλογές χρήσης, όπως:

- **Άμεση πρόσβαση μέσω γραμμής εντολών** με την εντολή ollama run <modelName>.
- **API κλήσεις** στη διεύθυνση http://localhost:11434/api/generate.
- Ενσωμάτωση μέσω προγραμματιστικών βιβλιοθηκών.

Στο πείραμά μας, επιλέξαμε να αλληλεπιδράσουμε με το Ollama μέσω δικών μας **Python scripts**, χρησιμοποιώντας τη βιβλιοθήκη **LangChain** για τη δημιουργία δομημένων ροών εργασίας με LLMs. Το LangChain διευκολύνει την ενσωμάτωση των LLMs, καθιστώντας τη διαχείριση των αποκρίσεων βάσει prompts και των RAG διαδικασιών πιο αποδοτική.

Εκτός από το Ollama, χρησιμοποιήσαμε το **OpenAI API** για την πρόσβαση στο embedding μοντέλο **text-embedding-large-3** και στο **gpt-4o**, για τη δημιουργία embeddings και την ανάκτηση ενισχυμένων αποτελεσμάτων.

Βάση Δεδομένων για Αποθήκευση Vectors

Στη διαδικασία RAG, χρειάστηκε μια βάση δεδομένων για την προσωρινή αποθήκευση embeddings vectors από τα τεμαχισμένα (chunked) κείμενα. Χρησιμοποιήσαμε το **Chroma**, μια ανοιχτού κώδικα βάση δεδομένων AI-native, προσαρμοσμένη για τη βελτίωση της παραγωγικότητας προγραμματιστών και ενσωματωμένη στενά με το LangChain. Το Chroma επέτρεψε την αποδοτική αποθήκευση και ανάκτηση δεδομένων vectorized, απαραίτητα για τη διαχείριση embeddings στη ροή εργασίας RAG.

Στήσιμο πειράματος και Εκτέλεση

Για τη δόμηση των σεναρίων και την αυτοματοποίηση της διαδικασίας δημιουργίας διαγραμμάτων κλάσεων, δημιουργήσαμε ένα πίνακα που περιέχει αναλυτικές πληροφορίες για κάθε περίπτωση, αποκαλούμενη ως "σενάριο." Το αρχείο αυτό περιλαμβάνει τις εξής στήλες:

- **ID:** Ένας μοναδικός αναγνωριστικός αριθμός για κάθε σενάριο, που συνδέεται απευθείας με το παραγόμενο διάγραμμα κλάσεων.

- **familyId:** Προσδιορίζει τη γενική κατηγορία ("οικογένεια") στην οποία ανήκει κάθε σενάριο, διευκολύνοντας τη δομή φακέλων. Η μορφή του είναι: {architecture}{fr_set_number}{nfr_set_number}{prompt_file_number}

  (π.χ., το cs111 δηλώνει Client-Server αρχιτεκτονική, FR set 1, NFR set 1, και prompt 1).

- **architecture:** Η ζητούμενη αρχιτεκτονική (π.χ. Client-Server, Three-Tier).

- **frPathfile:** Path για το αρχείο λειτουργικών απαιτήσεων (FR).

- **nfrPathfile:** Path για το αρχείο μη λειτουργικών απαιτήσεων (NFR).

- **promptPathfile:** Path για το αρχείο κειμένου του prompt.

- **modelMetadata:** Το όνομα και η έκδοση του επιλεγμένου LLM (π.χ. llama3.1:latest, gemma2:27b).

- **source:** Πληροφορίες για το αν το διάγραμμα κλάσεων δημιουργήθηκε μέσω Python scripts ή μέσω online μεθόδων (π.χ., ChatGPT).

- **ragDecision:** Καθορίζει αν το RAG χρησιμοποιείται για το σενάριο (Ναι ή Όχι).

- **ragPathfile:** Διαδρομή στο αρχείο RAG με εξωτερικά δεδομένα για τα σενάρια που χρησιμοποιούν RAG.

- **embeddingsMetadata:** JSON αντικείμενο με λεπτομέρειες για τη διαδικασία RAG, όπως:

```
{
  "agent": "openai/ollama",
  "model": "nomic-embed-text:latest / text-embedding
    -large-3",
  "chunking_info": {
    "method": "recursive/semantic",
    "chunk_size": "1000/-"
  }
}
```

- **resultPath:** Συμπληρώνεται αυτόματα από το script μετά τη δημιουργία της απόκρισης, προσδιορίζοντας το path για το αρχείο αποτελέσματος.

Αυτός ο δομημένος πίνακας επιτρέπει στα Python scripts να διατρέχουν κάθε σενάριο, αυτοματοποιώντας τη διαδικασία δημιουργίας διαγραμμάτων κλάσεων σύμφωνα με τις καθορισμένες απαιτήσεις και διαμορφώσεις.

Η ρύθμιση αυτή διευκολύνει τη συστηματική διερεύνηση κάθε μεταβλητής (π.χ. τύπος αρχιτεκτονικής, επιλογή μοντέλου, χρήση RAG) και διασφαλίζει ότι τα αποτελέσματα είναι οργανωμένα, συνεπή και εύκολα ανιχνεύσιμα σε όλα τα σενάρια του πειράματος.

3.2.4 Πειράματα

Για να αξιολογήσουμε την επίδραση κάθε παραμέτρου (τύπος αρχιτεκτονικής, σύνολα FR/NFR, επιλογή μοντέλου, χρήση RAG και στρατηγική embedding) στην ποιότητα των διαγραμμάτων κλάσεων που παράγονται από τα διάφορα μοντέλα, διεξαγάγαμε ένα ολοκληρωμένο σύνολο πειραμάτων υπό ελεγχόμενες συνθήκες. Στόχος μας ήταν να απομονώσουμε την επίδραση κάθε μεταβλητής, δημιουργώντας και αναλύοντας μια ποικιλία σεναρίων.

Ο πειραματικός μας σχεδιασμός περιλάμβανε την επανάληψη του ίδιου συνόλου σεναρίων σε τρεις διακριτές αρχιτεκτονικές: **Client-Server**, **Three-Tier**, και **Model-View-Controller**. Κάθε σύνολο αρχιτεκτονικής περιείχε **160 μοναδικά σενάρια**, με αποτέλεσμα τη δημιουργία συνολικά **480 διαγραμμάτων**.

Αυτός ο σχεδιασμός μας επέτρεψε να μετρήσουμε την απόδοση και να αξιολογήσουμε πώς κάθε παράμετρος επηρεάζει την ακρίβεια, τη συνάφεια και τη δομή των διαγραμμάτων κλάσεων που δημιουργούνται, ενισχύοντας την κατανόηση του τρόπου με τον οποίο οι διαφορετικές διαμορφώσεις επηρεάζουν τα αποτελέσματα. Τα σενάρια κατασκευάστηκαν βάσει συνδυασμών παραμέτρων, όπως:

- **Μεγάλα Γλωσσικά Μοντέλα (LLMs):** Χρησιμοποιήθηκαν διαφορετικά μοντέλα, όπως τα llama3.1, gemma2:27b, και command-r, για να αξιολογηθεί πώς η επιλογή του μοντέλου

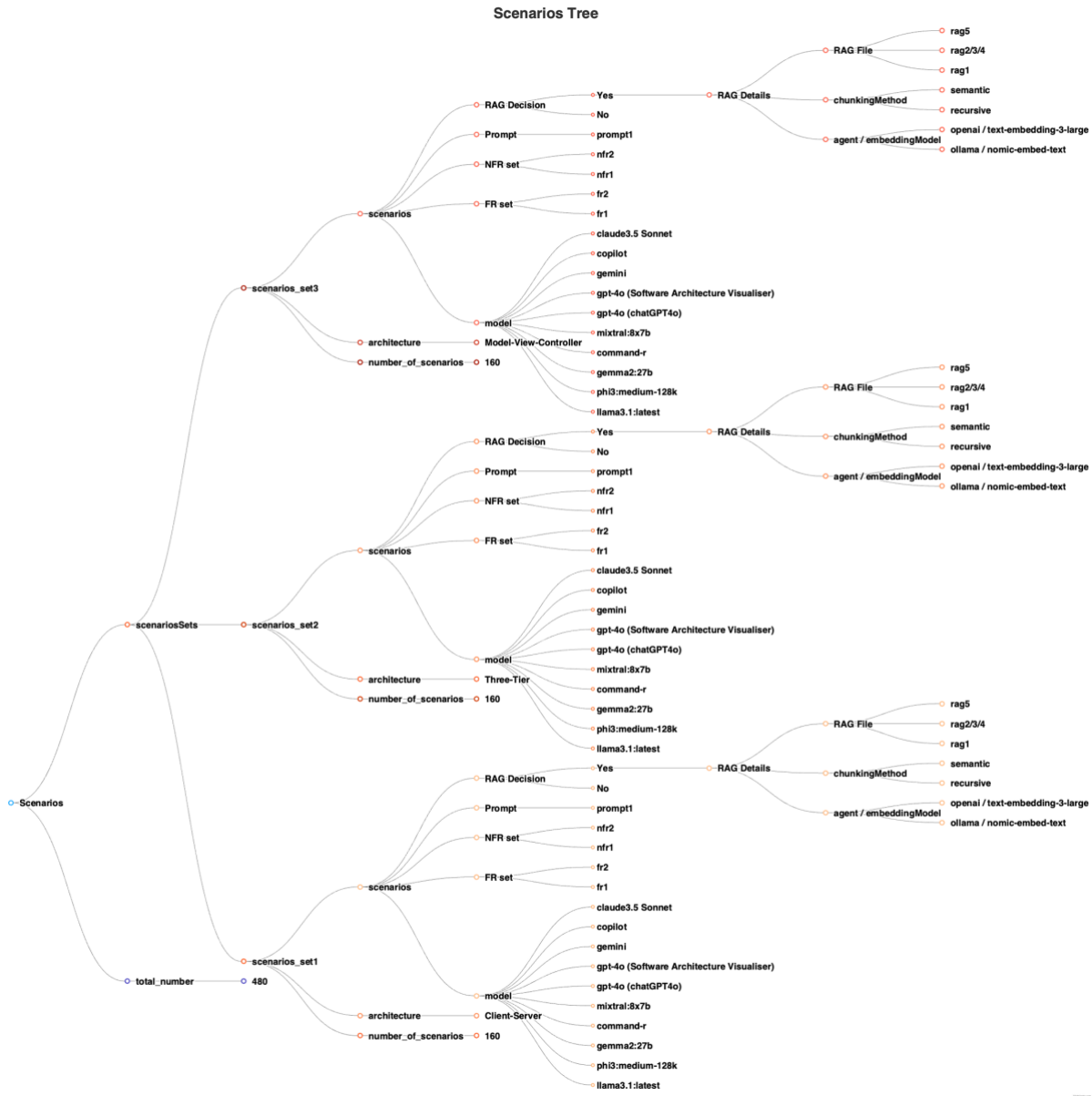
επηρεάζει τη διαδικασία δημιουργίας διαγραμμάτων.

- **RAG έναντι Μη-RAG:** Δοκιμάσαμε σενάρια με και χωρίς χρήση του **Retrieval-Augmented Generation (RAG)**, ώστε να παρατηρήσουμε πώς η παροχή συμπληρωματικών πληροφοριών επηρεάζει την ακρίβεια και τη δομή των παραγόμενων διαγραμμάτων.

- **Λειτουργικές και Μη Λειτουργικές Απαιτήσεις (FR/NFR):** Χρησιμοποιήθηκαν δύο εκδόσεις των συνόλων FR και NFR για να εξεταστεί αν η λεπτομέρεια και η δομή των απαιτήσεων επηρεάζει τα αποτελέσματα των μοντέλων.

- **Μοντέλα και Μέθοδοι Embedding:** Ενσωματώθηκαν διαφορετικά μοντέλα embeddings και μέθοδοι τμηματοποίησης (chunking) στη διαδικασία RAG, για να μελετηθεί πώς αυτοί οι παράγοντες επηρεάζουν την ανάκτηση και τη συνάφεια των αποτελεσμάτων.

Στον γράφημα που ακολουθεί στην Εικόνα 3.11, παραθέτουμε λεπτομερή απεικόνιση των σεναρίων:



Εικόνα 3.11 Γράφος σεναρίων

3.2.5 Αξιολόγηση από AI και Ανθρώπινο Δυναμικό

Η τελική φάση του πειράματός μας περιλαμβάνει μια ολοκληρωμένη αξιολόγηση των παραγόμενων διαγραμμάτων κλάσεων μέσω δύο μεθόδων την αξιολόγηση από χρήστες και την αξιολόγηση με τη χρήση AI στους παρακάτω άξονες αξιολόγησης.

Άξονες Αξιολόγησης

1. **Τήρηση της Αρχιτεκτονικής**
   Οι χρήστες αξιολόγησαν κατά πόσο η ζητούμενη αρχιτεκτονική υλοποιήθηκε σωστά σε κάθε διάγραμμα κλάσεων. Η αξιολόγηση επικεντρώθηκε στο αν τηρήθηκαν οι αρχιτεκτονικές αρχές, όπως η κατάλληλη κατανομή ευθυνών μεταξύ των κλάσεων και η ευθυγράμμιση με το καθορισμένο αρχιτεκτονικό πρότυπο.

2. **Ορθότητα των Σχέσεων μεταξύ Κλάσεων**
   Αυτή η διάσταση εξέτασε την ακρίβεια των σχέσεων μεταξύ των κλάσεων στο πλαίσιο της καθορισμένης αρχιτεκτονικής. Οι χρήστες αξιολόγησαν αν οι συσχετίσεις, οι εξαρτήσεις και οι ροές επικοινωνίας μεταξύ των κλάσεων απεικονιζόταν σωστά και αν τηρούσαν τις αρχιτεκτονικές αρχές.

3. **Συνοχή και Συζευκτικότητα (Cohesion and Coupling)**
   Η υψηλή συνοχή και η χαμηλή συζευκτικότητα αποτελούν θεμελιώδεις σχεδιαστικές αρχές για διατηρήσιμες και αποδοτικές αρχιτεκτονικές. Οι χρήστες βαθμολόγησαν κάθε διάγραμμα με βάση την εστίαση και τον μοναδικό σκοπό κάθε κλάσης (υψηλή συνοχή) και την ελαχιστοποίηση των εξαρτήσεων μεταξύ των κλάσεων (χαμηλή συζευκτικότητα).

4. **Συνέπεια με τις Απαιτήσεις Λογισμικού**
   Οι χρήστες εξέτασαν αν κάθε διάγραμμα κάλυπτε τόσο τις λειτουργικές όσο και τις μη λειτουργικές απαιτήσεις, όπως καθορίζονται στα σύνολα απαιτήσεων. Αυτή η αξιολόγηση εξασφάλισε ότι τα διαγράμματα δεν τηρούσαν μόνο τις αρχιτεκτονικές αρχές, αλλά και εκπροσωπούσαν με ακρίβεια την απαιτούμενη λειτουργικότητα.

Αξιολόγηση από Ανθρώπινο δυναμικό

Συγκροτήσαμε μια ομάδα από τέσσερα άτομα με διαφορετικά επίπεδα εμπειρίας στον τομέα της αρχιτεκτονικής λογισμικού για να αξιολογήσουν τα παραγόμενα διαγράμματα. Κάθε χρήστης πραγματοποίησε ανεξάρτητη αξιολόγηση και βαθμολόγηση των διαγραμμάτων βάσει τεσσάρων βασικών διαστάσεων όπως παρουσιάστηκαν παραπάνω.

Κάθε χρήστης έδωσε έναν ακέραιο βαθμό από 0 έως 5 σε κάθε άξονα. Αυτή η διαδικασία δημιούργησε ένα πλούσιο ποιοτικό και ποσοτικό σύνολο δεδομένων για την ανάλυση της ποιότητας των διαγραμμάτων.

Αξιολόγηση μέσω AI

Χρησιμοποιήσαμε επίσης LLMs για την αξιολόγηση των διαγραμμάτων κλάσεων. Σε κάθε LLM δόθηκε ως είσοδο το παραγόμενο διάγραμμα και η καθορισμένη αρχιτεκτονική και κλήθηκε να αξιολογήσει τα διαγράμματα με βάση τις ίδιες τέσσερις διαστάσεις που χρησιμοποιήθηκαν από τους ανθρώπινους ειδικούς.

Τα LLMs παρείχαν:

- **Ακέραιους Βαθμούς:** Όπως οι χρήστες που αξιολόγησαν, και τα LLMs βαθμολόγησαν κάθε κριτήριο με βαθμό από 0 έως 5, επιτρέποντας την άμεση σύγκριση μεταξύ των αξιολογήσεων AI και ανθρώπων.

- **Κείμενα Δικαιολόγησης:** Για κάθε βαθμό, τα LLMs παρείχαν επεξηγήσεις σχετικά με τη λογική της βαθμολόγησής τους. Αυτή η προσέγγιση ήταν κρίσιμη για την κατανόηση των διαδικασιών λογικής των LLMs, βοηθώντας μας να εκτιμήσουμε την ικανότητά τους να αναγνωρίζουν αρχιτεκτονικές αρχές, να εντοπίζουν σφάλματα και να εκτιμούν τις βέλτιστες πρακτικές στον σχεδιασμό διαγραμμάτων κλάσεων.

Ο συνδυασμός αξιολογήσεων από ανθρώπους και AI μας επιτρέπει να πραγματοποιήσουμε μια ολοκληρωμένη ανάλυση της ποιότητας κάθε διαγράμματος. Μέσω της σύγκρισης της συνέπειας και της ευθυγράμμισης των βαθμολογιών και των δικαιολογήσεων που παρέχονται από τα LLMs με εκείνες των ανθρώπινων ειδικών, επιδιώκουμε να εντοπίσουμε πιθανά πλεονεκτήματα και αδυναμίες της αξιολόγησης με βάση το AI.

Αυτή η διπλή προσέγγιση προσφέρει ένα ισχυρό πλαίσιο για την αξιολόγηση της αποτελεσματικότητας των μοντέλων στη δημιουργία και βαθμολόγηση διαγραμμάτων κλάσεων, παρέχοντας μας μια λεπτομερή κατανόηση των τρεχουσών δυνατοτήτων της τεχνητής νοημοσύνης και των τομέων όπου απαιτείται περαιτέρω βελτίωση.

4 ΕΚΤΕΛΕΣΗ – ΑΠΟΤΕΛΕΣΜΑΤΑ

4.1 Τυπικές περιπτώσεις

Σε αυτή την ενότητα θα παρουσιάσουμε ορισμένα UML διαγράμματα κλάσεων που δημιουργήθηκαν από τα LLMs.

Πλήρη και ακριβή διαγράμματα κλάσεων UML

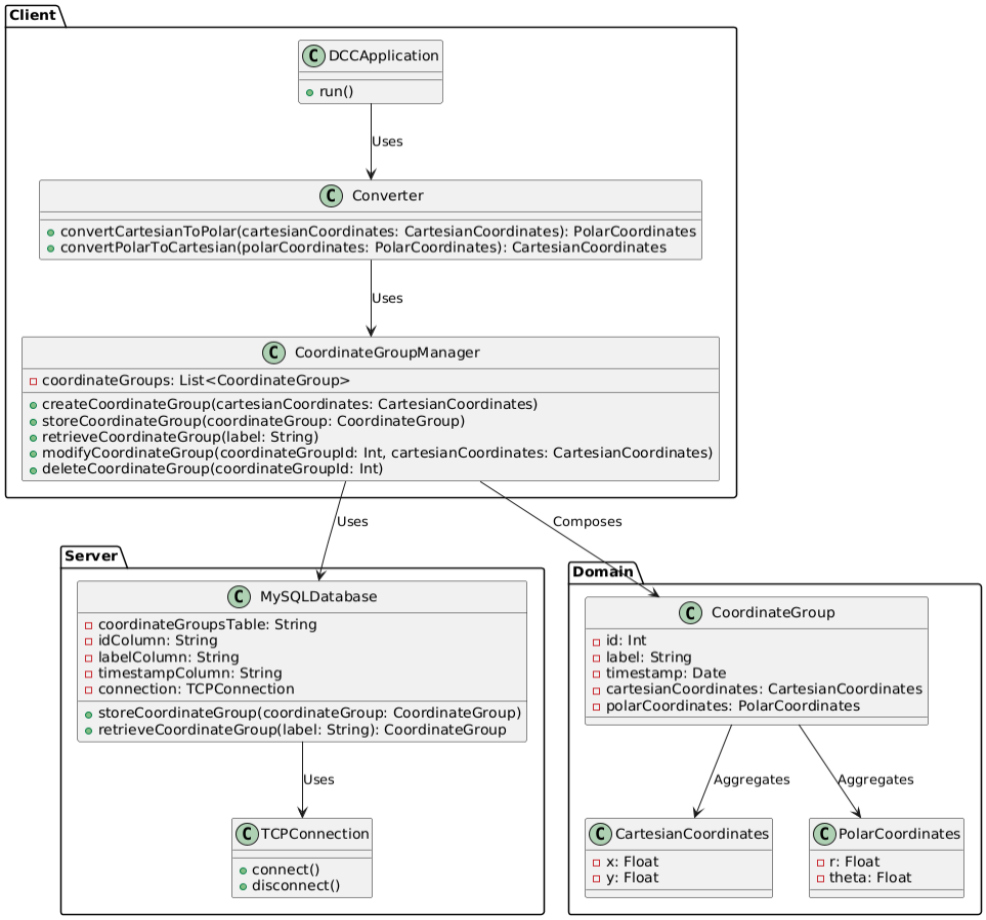
Τα παρακάτω διαγράμματα κλάσεων αναδεικνύουν ορισμένες ενδιαφέρουσες και αξιόλογες προσεγγίσεις στην αποτύπωση των αρχιτεκτονικών προτύπων. Όλα τα διαγράμματα τηρούν επιτυχώς τις βασικές αρχές των τριών συγκεκριμένων αρχιτεκτονικών. Επιπλέον, εναρμονίζονται με τις απαιτήσεις του λογισμικού, αποτυπώνουν με ακρίβεια τις σχέσεις μεταξύ των κλάσεων και παρουσιάζουν υψηλή συνοχή και χαμηλή συζευκτικότητα, αντικατοπτρίζοντας εξαιρετική απόδοση στα βασικά κριτήρια αξιολόγησης.

Client-Sever:

Llama3.1:8B | fr1 | nfr1 | rag1 | ollama | nomic-ebed-text:latest | semantic (ID = 3)

Το διάγραμμα στην Εικόνα 4.1 περιλαμβάνει τρία διακριτά πακέτα: **client**, **server** και **domain**, καθένα από τα οποία έχει σαφώς καθορισμένους ρόλους. Τα πακέτα συνδέονται μέσω λεπτομερών σχέσεων μεταξύ των κλάσεων, αποδεικνύοντας αποτελεσματική επικοινωνία μεταξύ των συστατικών στοιχείων. Το διάγραμμα ενσωματώνει με επιτυχία σχεδόν όλες τις απαιτήσεις του λογισμικού, περιλαμβάνοντας συγκεκριμένες λεπτομέρειες σχετικά με τη σύνδεση TCP/IP. Στην πλευρά του διακομιστή (server), διαχειρίζονται οι ευθύνες της βάσης δεδομένων, ενώ στην πλευρά του πελάτη (client) περιλαμβάνονται κλάσεις όπως η DCCApplication με μέθοδο run(), μια κλάση Converter, καθώς και μια κλάση Manager εξοπλισμένη με λειτουργίες CRUD για την αλληλεπίδραση με τον διακομιστή.

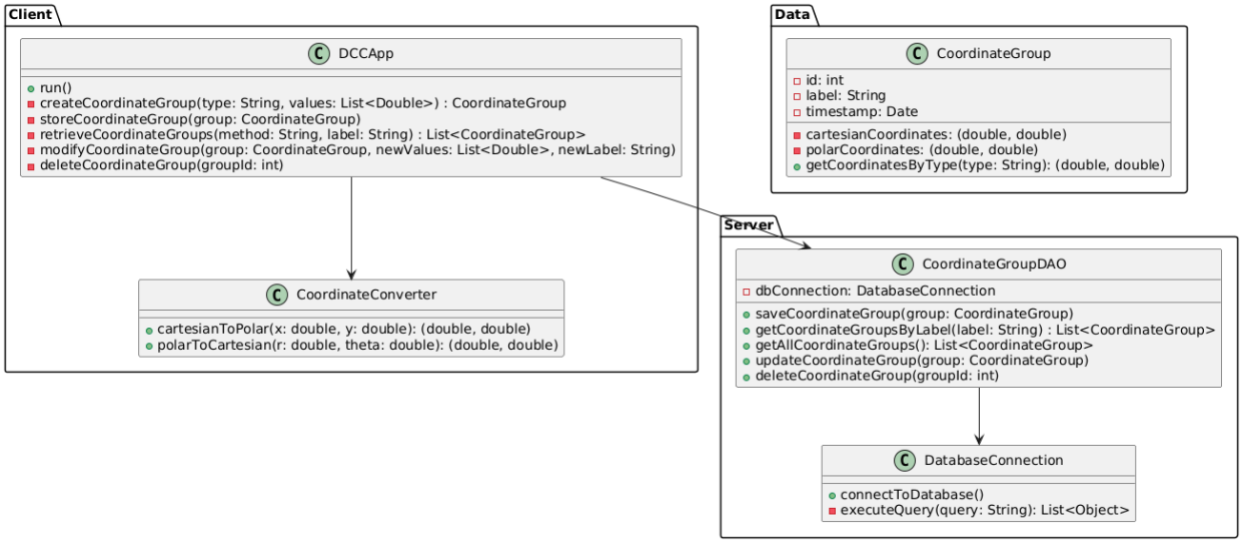
Συνολικά, αυτό το διάγραμμα είναι πλήρες και πληροί όλα τα κριτήρια, καθιστώντας το μια ισχυρή και ακριβή αναπαράσταση της αρχιτεκτονικής **πελάτη-εξυπηρετητή**.



Εικόνα 4.1 Διάγραμμα με ID = 3

gemma2:27B | fr1 | nfr1 | rag5 | ollama | nomic-ebed-text:latest | semantic (ID = 168)

Το διάγραμμα στην Εικόνα 4.2 προσφέρει μια παρόμοια και εξίσου ορθή προσέγγιση στην αρχιτεκτονική **πελάτη-εξυπηρετητή** με εκείνη που περιγράφηκε παραπάνω. Καθορίζει διακριτά πακέτα **client** και **server**, καθένα από τα οποία περιλαμβάνει τις κατάλληλες κλάσεις με επαρκή χαρακτηριστικά. Στην πλευρά του πελάτη (**client**), το διάγραμμα ενσωματώνει αποτελεσματικά τόσο το περιβάλλον χρήστη (**UI**) όσο και τη λογική της εφαρμογής, ενώ η πλευρά του διακομιστή (**server**) είναι υπεύθυνη για τη διαχείριση της βάσης δεδομένων.



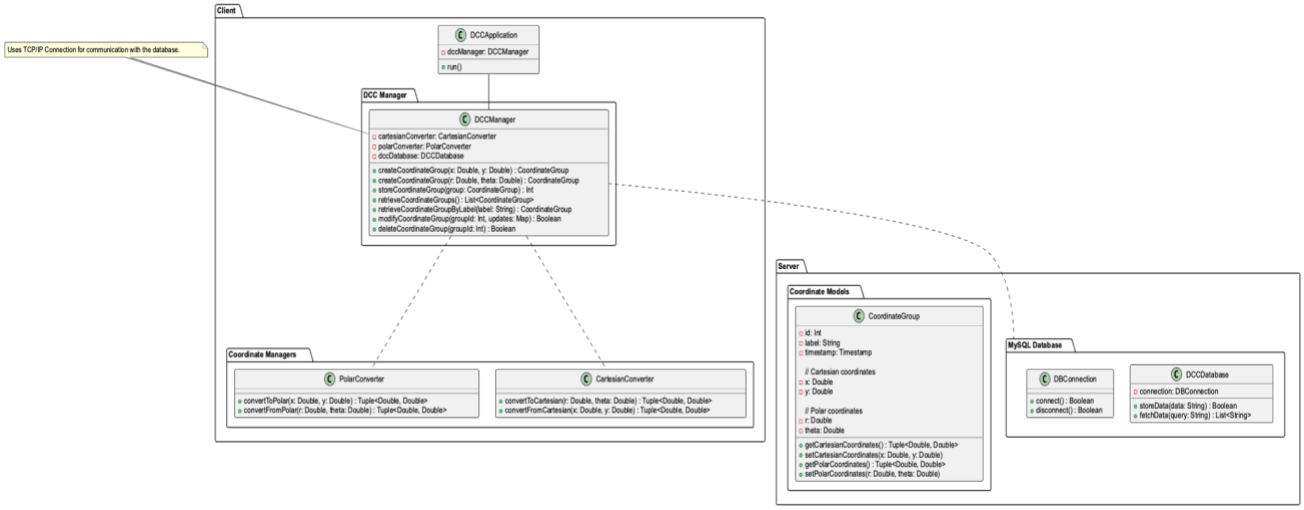
Εικόνα 4.2 Διάγραμμα με ID = 168

command-r | fr1 | nfr1 | rag5 | openai | text-embedding-3-large | semantic (ID = 185)

Το διάγραμμα στην Εικόνα 4.3 παρουσιάζει μια ακόμη αξιόλογη προσέγγιση στην αρχιτεκτονική **πελάτη-εξυπηρετητή**. Αποτελείται από δύο κύρια πακέτα: **client** και **server**.

Στην πλευρά του διακομιστή (**server**), περιλαμβάνονται κλάσεις αφιερωμένες στη διαχείριση της βάσης δεδομένων και της σύνδεσης με αυτήν. Η πλευρά του πελάτη (**client**), διαθέτει ξεχωριστές κλάσεις για τον **PolarConverter** και τον **CartesianConverter**, καθώς και μια ολοκληρωμένη κλάση **Manager** και την κλάση **DCCApplication**.

Αυτό το διάγραμμα ακολουθεί τις βασικές αρχές του μοντέλου **πελάτη-εξυπηρετητή**, ενώ παράλληλα επιδεικνύει εξαιρετική συνοχή εντός των συστατικών στοιχείων και κατάλληλα επίπεδα συζευκτικότητας μεταξύ τους.

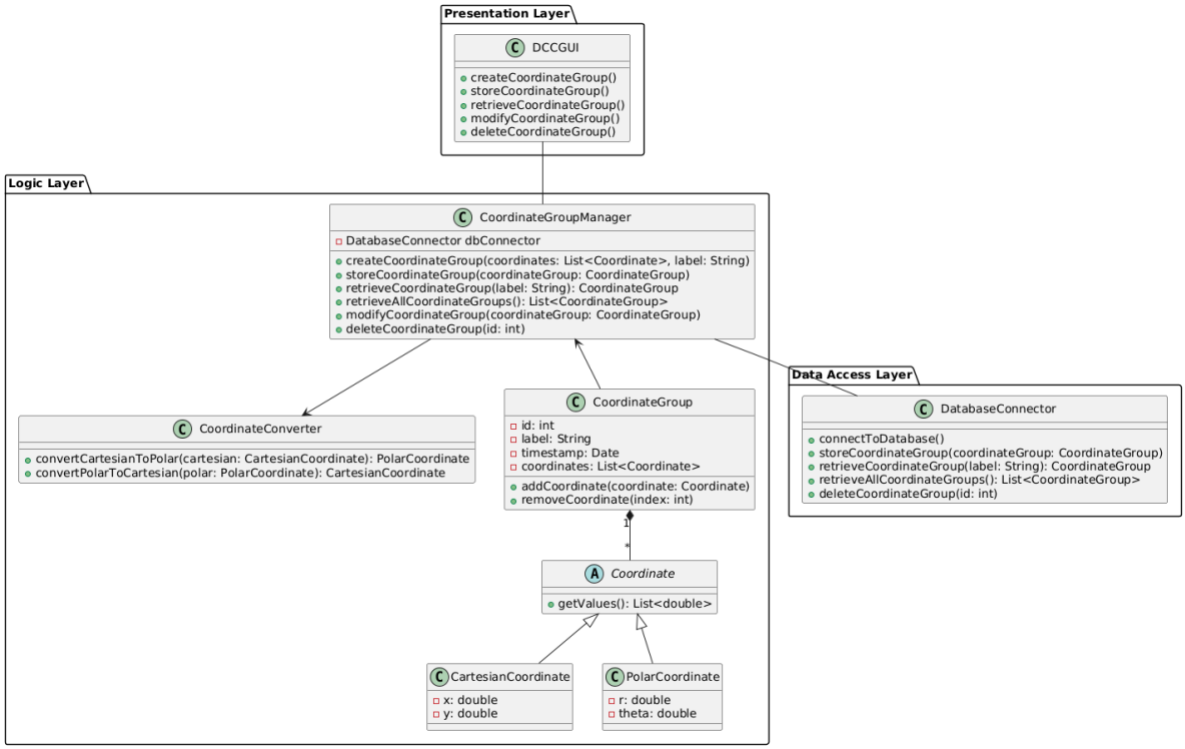


Εικόνα 4.3 Διάγραμμα με ID = 185

Three-Tier:

gemma2:27B | fr1 | nfr1 | rag1 | ollama | nomic-ebed-text:latest | semantic (ID = 23)

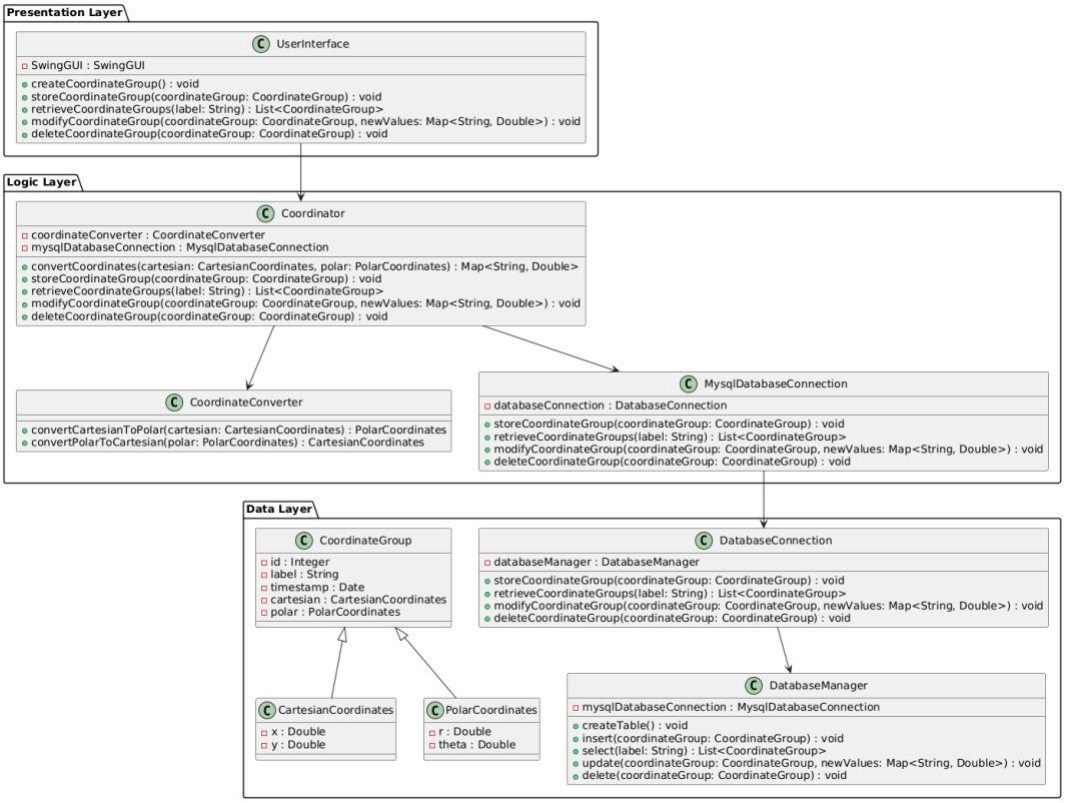
Το διάγραμμα στην Εικόνα 4.4 συμμορφώνεται με την τήρηση των βασικών αρχών της αρχιτεκτονικής **Τριών Επιπέδων (Three-Tier)**. Παρουσιάζει με σαφήνεια τον διαχωρισμό των ευθυνών ανάμεσα στα επίπεδα, όχι μόνο μέσω της χρήσης διακριτών πακέτων, αλλά και με την κατανομή της λειτουργικότητας στα κατάλληλα στρώματα. Για παράδειγμα εμφανής είναι η ενσωμάτωση της λειτουργίας μετατροπής συντεταγμένων, η οποία αντιπροσωπεύει τη βασική επιχειρησιακή λογική του λογισμικού, και η σωστή τοποθέτησή της στο αντίστοιχο στρώμα.



Εικόνα 4.4 Διάγραμμα με ID = 23

Llama3.1:8B | fr1 | nfr1 | rag3 | openai | text-embedding-3-large | semantic (ID = 85)

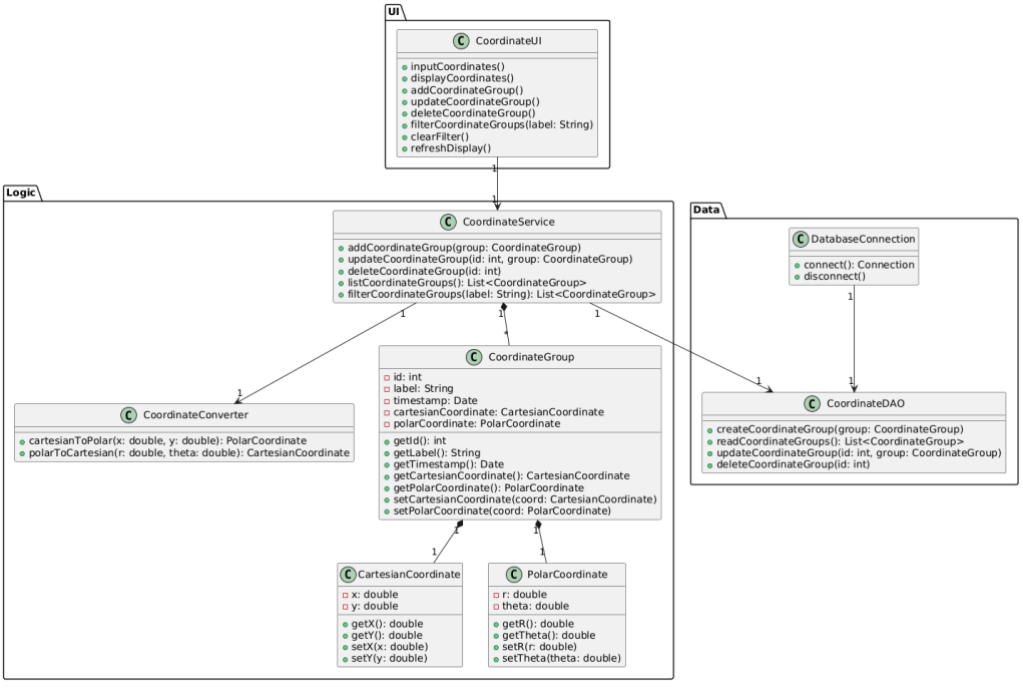
Το διάγραμμα στην Εικόνα 4.5 όχι μόνο ευθυγραμμίζεται με την αιτούμενη αρχιτεκτονική και τις αρχές της, αλλά περιλαμβάνει επίσης καλά τεκμηριωμένες κλάσεις με πλήθος ακριβών χαρακτηριστικών. Κάθε στρώμα και οι αντίστοιχες κλάσεις του έχουν σχεδιαστεί με ολοκληρωμένο και προσεγμένο τρόπο, αντανακλώντας υψηλό επίπεδο συμμόρφωσης με τα αρχιτεκτονικά πρότυπα.



Εικόνα 4.5 Διάγραμμα με ID = 85

gpt4o | fr2 | nfr2 | No RAG (ID = 467)

Παρόμοια με το προηγούμενο παράδειγμα, το διάγραμμα στην Εικόνα 4.6 ευθυγραμμίζεται πλήρως με την αρχιτεκτονική **Τριών Επιπέδων** και τις αρχές της. Περιλαμβάνει σαφώς καθορισμένα πακέτα και κλάσεις, κάθε μία από τις οποίες διαθέτει αξιοσημείωτο επίπεδο λεπτομέρειας.

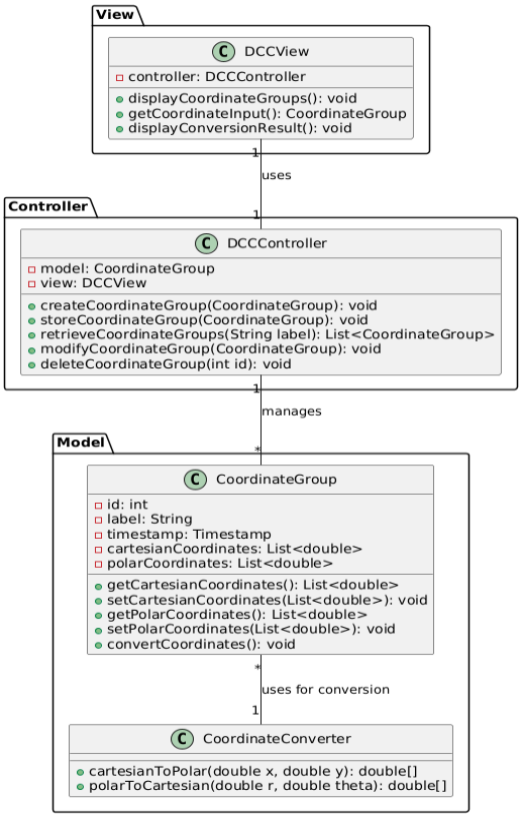


Εικόνα 4.6 Διάγραμμα με ID = 467

Model - View – Controller:

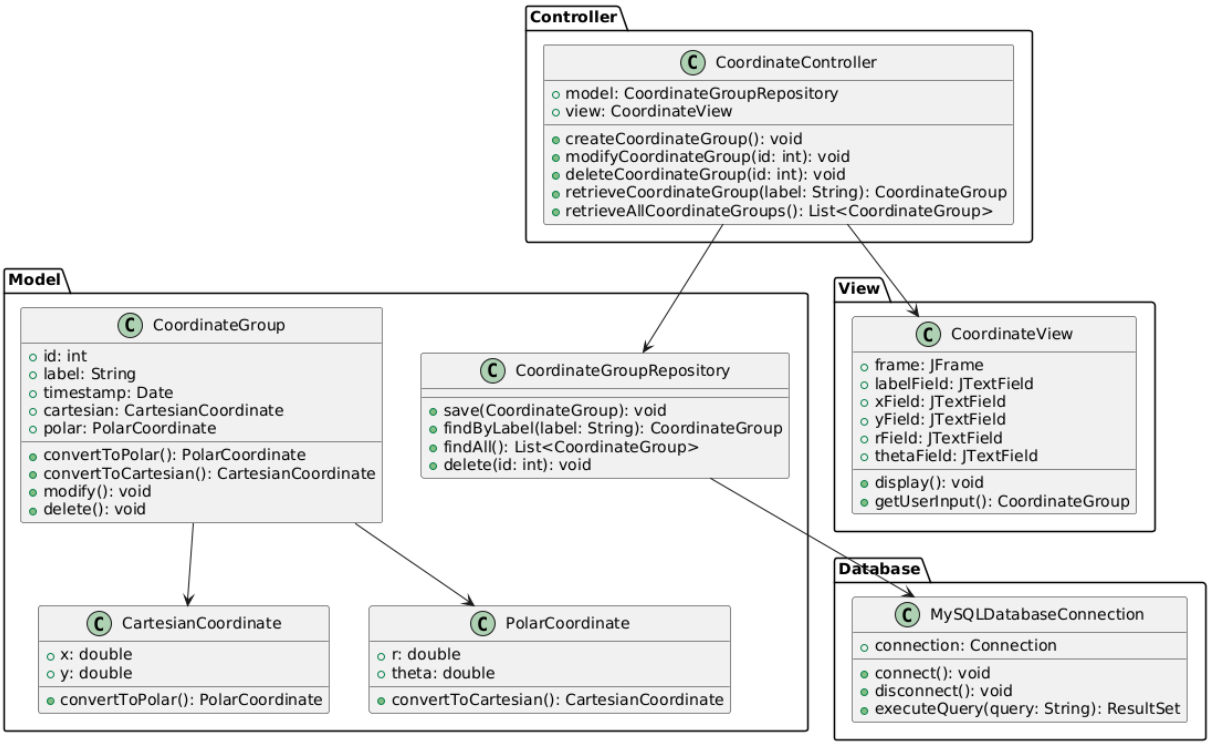
Τα παρακάτω τρία διαγράμματα παρουσιάζουν χαρακτηριστική συμφωνία με το αρχιτεκτονικό πρότυπο **Model-View-Controller (MVC)**. Κάθε διάγραμμα περιλαμβάνει τα τρία βασικά πακέτα: Model, View και Controller, τα οποία είναι σωστά οργανωμένα ώστε να αντικατοπτρίζουν τις αρχές της συγκεκριμένης αρχιτεκτονικής. Επιπλέον, και τα τρία διαγράμματα ενσωματώνουν τη λογική της επιχειρησιακής λειτουργίας για τη μετατροπή, τοποθετώντας την ορθά στο πακέτο Model, σύμφωνα με τις βέλτιστες αρχιτεκτονικές πρακτικές.

gemma2:27B | fr1 | nfr1 | rag1 | openai | text-embedding-3-large | semantic (ID = 30)



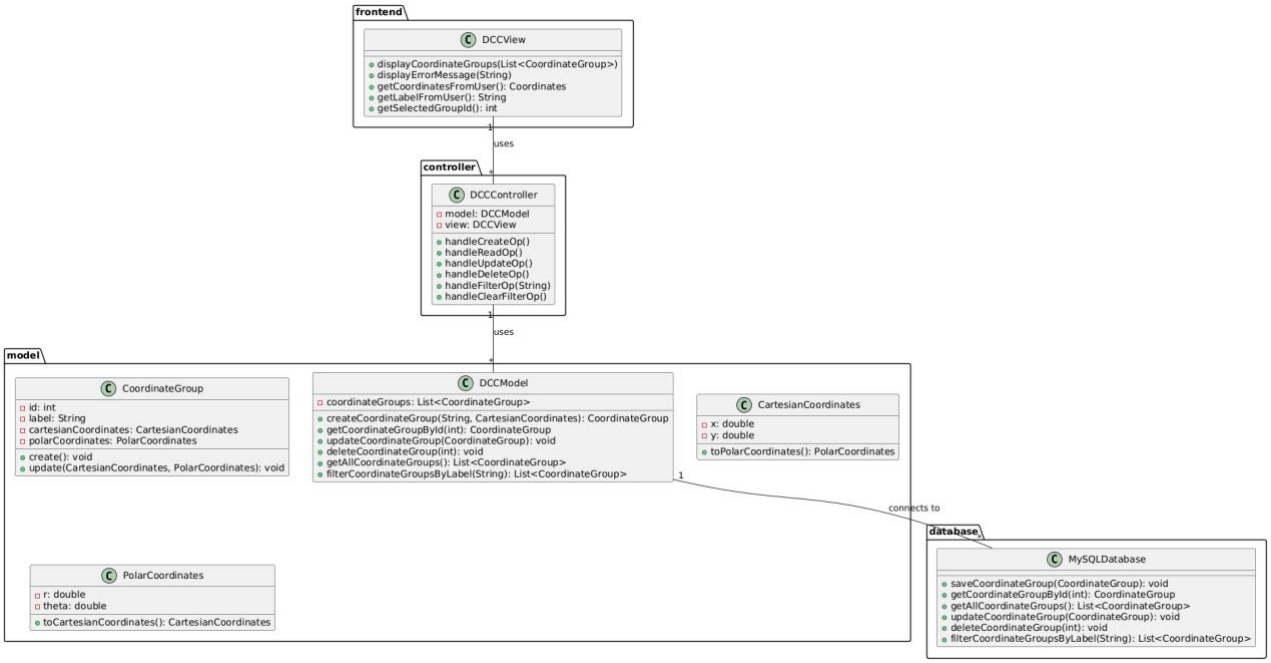
Εικόνα 4.7 Διάγραμμα με ID = 30

gpt4oSAV | fr1 | nfr1 | No RAG (ID = 231)



Εικόνα 4.8 Διάγραμμα με ID = 231

gemma2:27B | fr2 | nfr2 | rag4 | ollama | nomic-embed-text:latest |semantic (ID = 342)



Εικόνα 4.9 Διάγραμμα με ID = 342

Ελλιπή και μη δομημένα διαγράμματα UML

Δεν κατάφεραν όλα τα διαγράμματα να ανταποκριθούν στα υψηλά επίπεδα των παραδειγμάτων που παρουσιάστηκαν προηγουμένως. Παρόλο που ορισμένα διαγράμματα έλαβαν μέτριες αξιολογήσεις, επιτυγχάνοντας εν μέρει στα βασικά σημεία, άλλα απέτυχαν σημαντικά. Τα παρακάτω διαγράμματα αντιπροσωπεύουν ιδιαίτερα χαμηλής ποιότητας αποτελέσματα, τα οποία χαρακτηρίζονται από την απουσία σχέσεων μεταξύ κλάσεων, πακέτων, χαρακτηριστικών και οποιασδήποτε συμμόρφωσης με το αιτούμενο αρχιτεκτονικό πρότυπο.

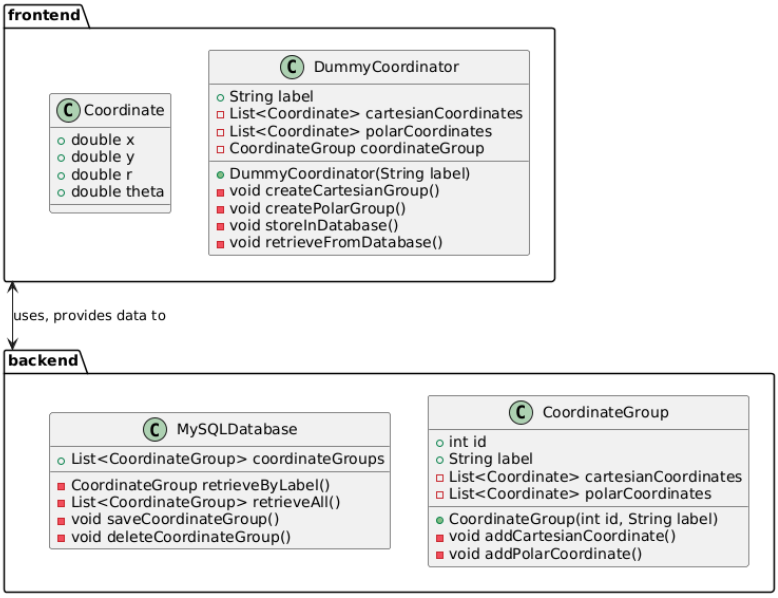
Η ενότητα αυτή παρουσιάζει μια επιλογή από τέτοια διαγράμματα UML. Αν και εμφανίζουν κάποιο επίπεδο δομικής συνοχής, είναι γενικά ελλιπή και αποτυγχάνουν να αποδώσουν πλήρως την προβλεπόμενη αρχιτεκτονική και λειτουργικότητα του συστήματος. Παρακάτω παρέχεται μια πιο λεπτομερής ανάλυση αυτών των διαγραμμάτων:

Client-Server:

Llama3.1:8B | fr1 | nfr1 | rag1 | openai | text-embedding-3-large | recursive (ID = 4)

Το παρακάτω διάγραμμα στην Εικόνα 4.10 προσπαθεί να ακολουθήσει την αρχιτεκτονική πελάτη-εξυπηρετητή , διαχωρίζοντας το frontend και το backend σε διακριτά πακέτα. Ωστόσο, παρουσιάζει σοβαρές ελλείψεις σε βασικά σημεία. Αρχικά, το διάγραμμα αποτυγχάνει να αποδώσει τον ρόλο του πελάτη (client) μέσα σε αυτήν την αρχιτεκτονική. Στο πακέτο **frontend**, οι κλάσεις **Coordinate** και **DummyCoordinator** περιλαμβάνονται, αλλά καμία δεν αντιπροσωπεύει το στοιχείο διεπαφής χρήστη (UI) του πελάτη.

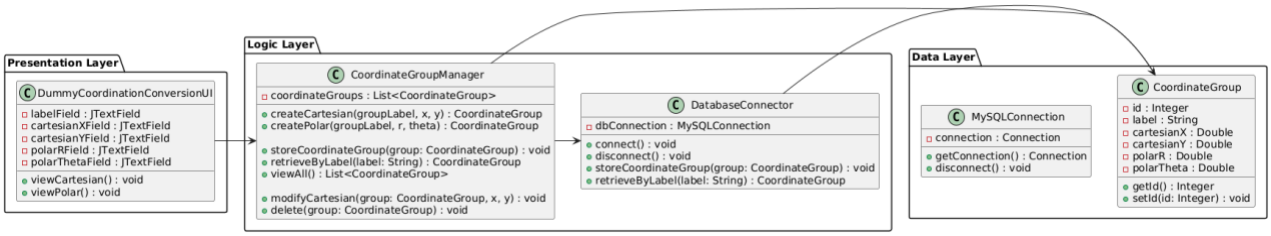
Επιπλέον, το διάγραμμα απεικονίζει μόνο μία σχέση μεταξύ των δύο πακέτων, κάτι που είναι ανεπαρκές για να αποτυπώσει την πολυπλοκότητα των αλληλεπιδράσεών. Τέλος, ένα κρίσιμο ζήτημα είναι η απουσία οποιασδήποτε αναπαράστασης της μετατροπής συντεταγμένων που είναι κομβική λειτουργική απαίτηση, είτε ως λειτουργία είτε ως ανεξάρτητη κλάση.



Εικόνα 4.10 Διάγραμμα με ID = 4

Llama3.1:8B | fr1 | nfr1 | rag5 | ollama | nomic-embed-text:latest | recursive (ID = 152)

Το διάγραμμα στην Εικόνα 4.11 παρουσιάζει σύγχυση σχετικά με τα αρχιτεκτονικά πρότυπα. Παρόλο που η ζητούμενη αρχιτεκτονική ήταν **Πελάτη-Εξυπηρετητή**, η παραγόμενη δομή ευθυγραμμίζεται περισσότερο με την αρχιτεκτονική **Τριών Στρωμάτων**. Επιπλέον, το διάγραμμα αποτυγχάνει να αποδώσει μια βασική λειτουργία επιχειρησιακής λογικής του συστήματος—τη λειτουργία μετατροπής συντεταγμένων—κάτι που μειώνει περαιτέρω την ακρίβεια και την πληρότητά του.

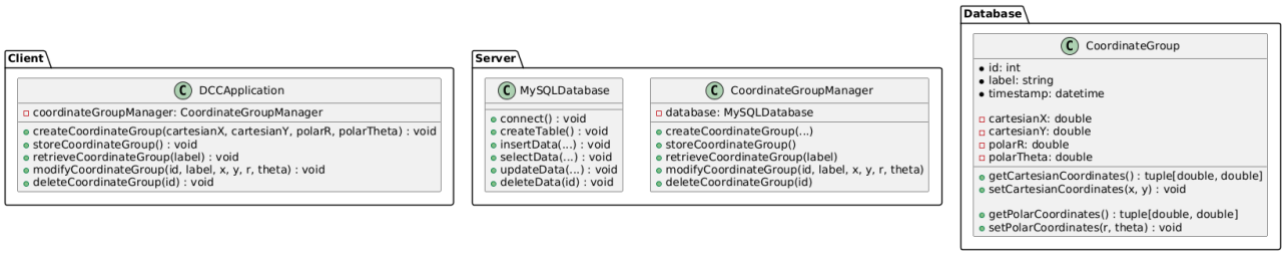


Εικόνα 4.11 Διάγραμμα με ID = 152

Three-Tier:

Command-r | fr1 | nfr1 | rag1 | openai | text-embedding-3-large | recursive (ID = 39)

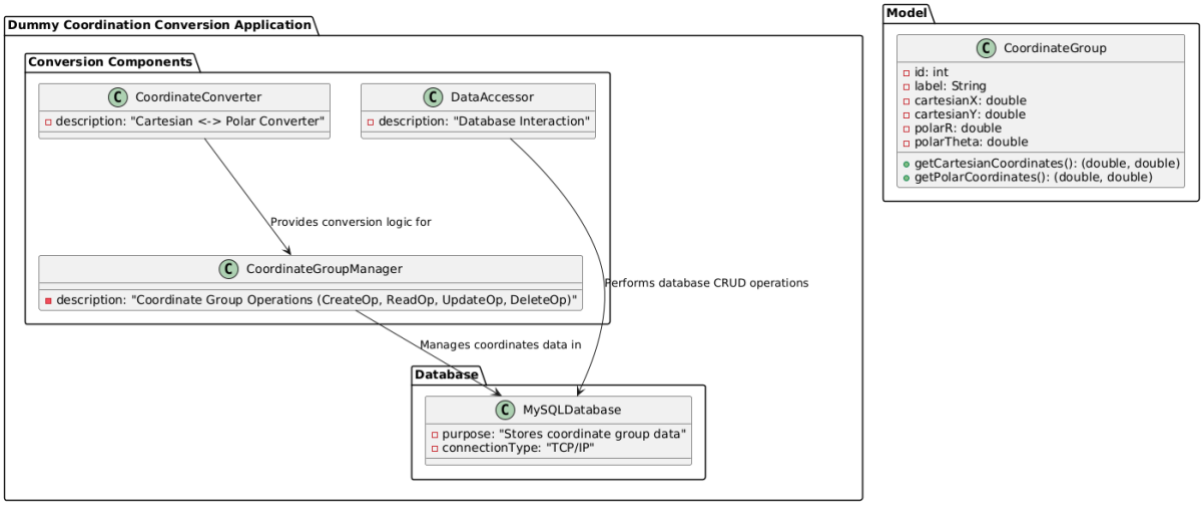
Το διάγραμμα στην Εικόνα 4.12 παρουσιάζει και πάλι σύγχυση σχετικά με τα αρχιτεκτονικά πρότυπα. Παρόλο που η ζητούμενη αρχιτεκτονική ήταν **Τριών Επιπέδων**, η παραγόμενη δομή φαίνεται να ακολουθεί τις αρχές της αρχιτεκτονικής **Πελάτη-Εξυπηρετητή**. Επιπλέον, το διάγραμμα στερείται οποιωνδήποτε σχέσεων μεταξύ κλάσεων και παραλείπει εντελώς τη βασική επιχειρησιακή λογική που σχετίζεται με τη λειτουργία μετατροπής συντεταγμένων.



Εικόνα 4.12 Διάγραμμα με ID = 39

Gemma2:27B | fr2 | nfr2 | rag3 | openai | text-embedding-3-large | semantic (ID = 340)

Το διάγραμμα στην Εικόνα 4.13 αποτυγχάνει να τηρήσει οποιαδήποτε βασική αρχή της αρχιτεκτονικής **Τριών Επιπέδων**. Επιπλέον, οι περισσότερες κλάσεις, με εξαίρεση το **Model**, στερούνται χαρακτηριστικών πέρα από ένα γενικό πεδίο περιγραφής. Συνολικά, αυτό το διάγραμμα είναι ανεπαρκώς δομημένο και ακατάλληλο για την ουσιαστική αναπαράσταση του λογισμικού της εφαρμογής.

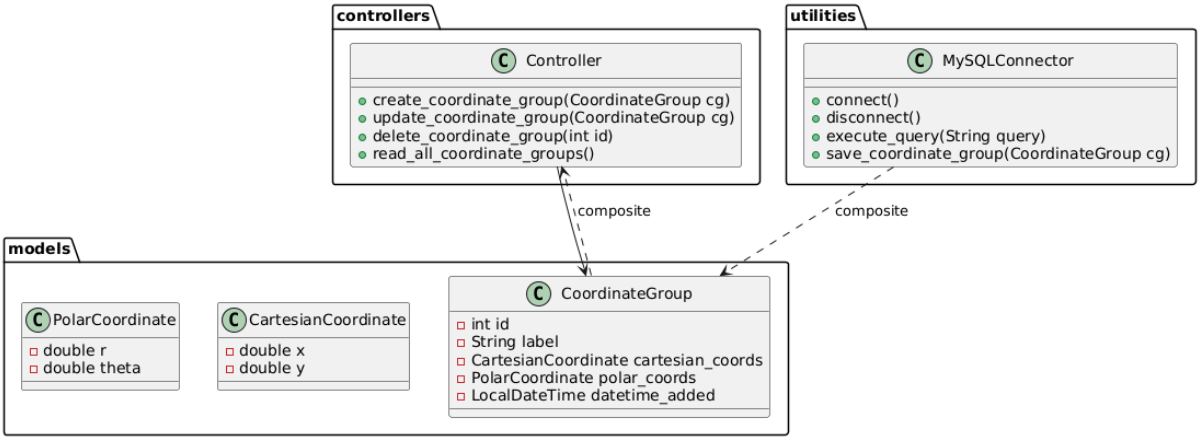


Εικόνα 4.13 Διάγραμμα με ID = 340

Model -View – Controller:

Llama3.1:8B | fr2 | nfr2 | rag1 | ollama | nomic-embed-text:latest | recursive (ID = 252)

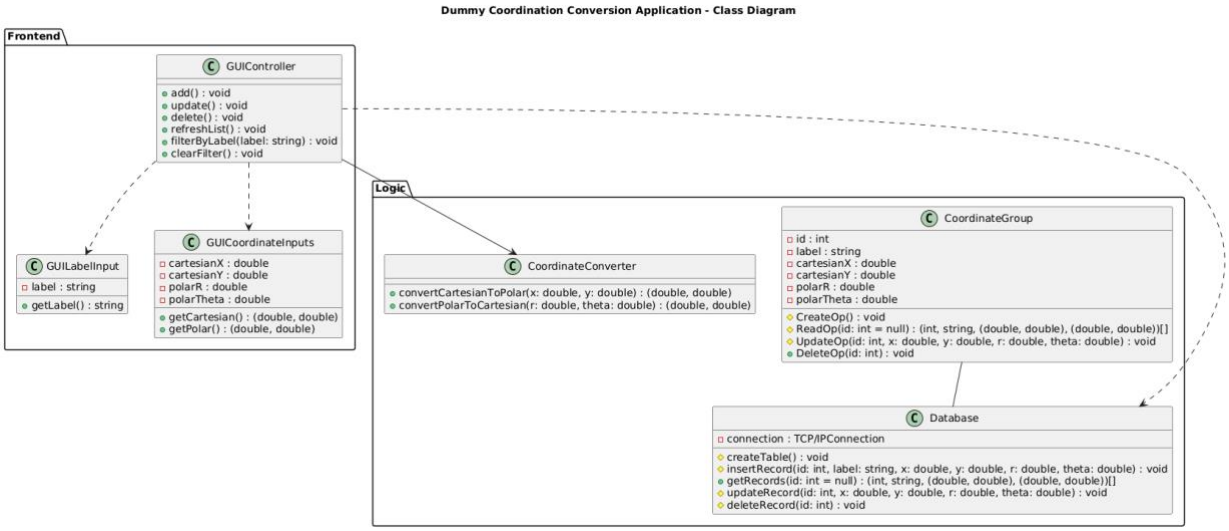
Το διάγραμμα στην Εικόνα 4.14 προσπαθεί να ακολουθήσει ορισμένες πρακτικές της αρχιτεκτονικής Model-View-Controller, όπως φαίνεται από την ύπαρξη των πακέτων Controllers και Models, με σχετικά κατάλληλες κλάσεις εντός αυτών. Ωστόσο, απουσιάζει ένα πακέτο ή μια κλάση View, ενώ δεν αποδίδει επίσης τη βασική επιχειρησιακή λειτουργία για τη μετατροπή συντεταγμένων, γεγονός που μειώνει ακόμα περισσότερο την πληρότητά του.



Εικόνα 4.14 Διάγραμμα με ID = 252

Command-r | fr2 | nfr2 | rag5 | ollama | nomic-embed-text:latest | recursive (ID = 432)

Το διάγραμμα στην Εικόνα 4.15 αποδεικνύει για άλλη μια φορά το φαινόμενο της παρερμηνείας της ζητούμενης αρχιτεκτονικής. Παρόλο που το ζητούμενο πρότυπο ήταν το **Model-View-Controller** , το διάγραμμα αυτό ευθυγραμμίζεται περισσότερο με την αρχιτεκτονική **Πελάτη-Εξυπηρετητή**. Παρά αυτό το σημαντικό ζήτημα, παρουσιάζει μια ικανοποιητική συμμόρφωση με τις λειτουργικές απαιτήσεις του λογισμικού.



Εικόνα 4.15 Διάγραμμα με ID = 432

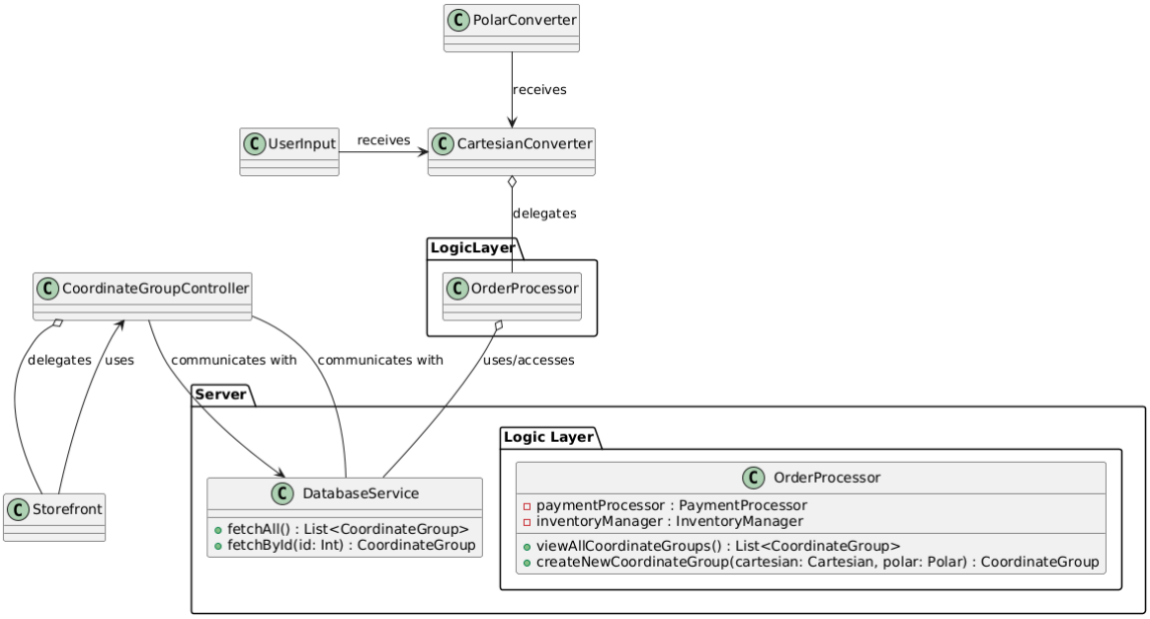
Πλήρως Ανεπαρκή Διαγράμματα

Υπάρχουν επίσης διαγράμματα που μπορούν να χαρακτηριστούν μόνο ως πλήρεις αποτυχίες σχεδόν σε όλα τα βασικά κριτήρια αξιολόγησης. Αυτά τα διαγράμματα παρουσιάζουν σημαντική σύγχυση όσον αφορά την αρχιτεκτονική, τις σχέσεις μεταξύ κλάσεων και τις απαιτήσεις του λογισμικού.

Τα προβλήματα ενδέχεται να οφείλονται στους περιορισμούς του εκάστοτε μοντέλου, σε αδυναμίες στη διαδικασία RAG ή σε συνδυασμό των δύο—όπως η αδυναμία του μοντέλου να διαχειριστεί αποτελεσματικά τη διαδικασία RAG όταν αντιμετωπίζει ένα περίπλοκο και εκτενές prompt. Παρακάτω, παρουσιάζουμε ένα παράδειγμα μιας τέτοιας περίπτωσης:

Phi3:medium-128k | fr1 | nfr1 | rag5 | openai | text-embedding-3-large | recursive (ID = 214)

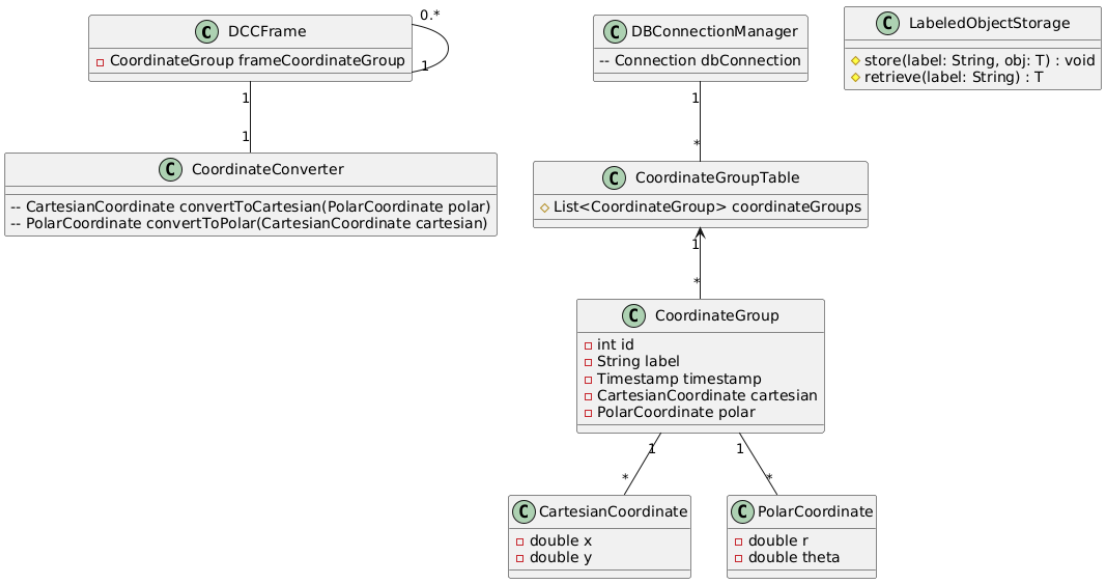
Η ζητούμενη αρχιτεκτονική για το διάγραμμα στην Εικόνα 4.16 ήταν **Πελάτη-Εξυπηρετητή (Client-Server)**. Ωστόσο, το διάγραμμα αποτυγχάνει να τηρήσει οποιοδήποτε αρχιτεκτονικό πρότυπο. Αξιοσημείωτο είναι ότι οι κλάσεις **Storefront** και **OrderProcessor** φαίνεται να προέρχονται από παράδειγμα διαγράμματος κλάσεων UML που περιλαμβανόταν στο αρχείο RAG. Αυτό υποδηλώνει ότι το διάγραμμα επηρεάστηκε σημαντικά—και τελικά μπερδεύτηκε—από τη διαδικασία RAG.



Εικόνα 4.16 Διάγραμμα με ID = 214

Mixtral:8x7B | fr1 | nfr1 | rag1 | ollama | nomic-embed-text:latest |semantic (ID = 48)

Το διάγραμμα στην Εικόνα 4.17 περιλαμβάνει ορισμένες κλάσεις, όπως οι **CoordinateGroup** και **CoordinateConverter**, που αποτυπώνουν ορισμένες από τις απαιτήσεις του λογισμικού. Ωστόσο, αποτυγχάνει πλήρως να τηρήσει οποιοδήποτε αρχιτεκτονικό πρότυπο. Επιπλέον, απουσιάζει η χρήση πακέτων, και συνολικά, πρόκειται για ένα εξαιρετικά αδύναμο αποτέλεσμα.



Εικόνα 4.17 Διάγραμμα με ID = 48

Επίδραση του συνόλου fr2 στη δομή των διαγραμμάτων

Σε αυτό το σημείο, θα θέλαμε να παρουσιάσουμε μια ενδιαφέρουσα παρατήρηση που έγινε κατά τη διάσχιση και αξιολόγηση των παραγόμενων διαγραμμάτων, συγκεκριμένα σε σχέση με τα αποτελέσματα του συνόλου **fr2**. Όπως περιγράφεται στην ενότητα 3.1.2, το σύνολο **fr2** οργανώνεται σε τμήματα: **Λογική (Logic)**, **Αλληλεπιδράσεις Χρήστη (User Interactions)** και **Λειτουργίες**

Δεδομένων (Data Operations), με διαφορετικές απαιτήσεις να περιγράφονται ξεχωριστά σε κάθε τμήμα.

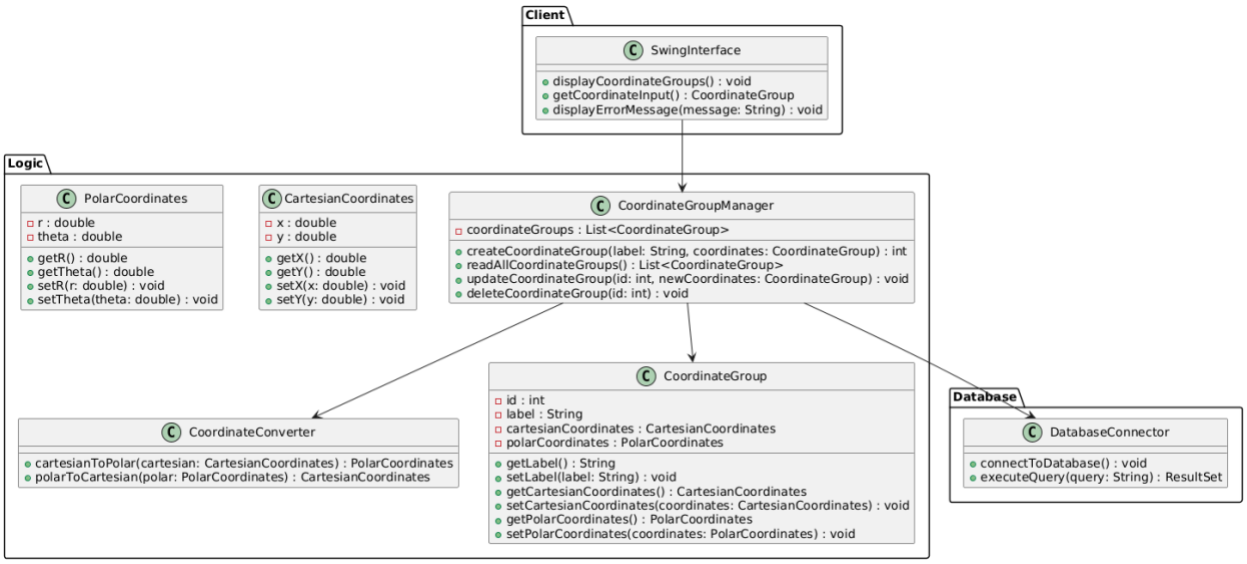
Αυτή η δομή έχει επηρεάσει τη δημιουργία των διαγραμμάτων κλάσεων, οδηγώντας πολλά από αυτά να ευθυγραμμίζονται με το πρότυπο της αρχιτεκτονικής **Τριών Επιπέδων**, ανεξαρτήτως δηλαδή της ζητούμενης αρχιτεκτονικής—είτε **Πελάτη-Εξυπηρετητή** , είτε **MVC** — τα διαγράμματα συχνά εμφανίζουν δομή που θυμίζει το μοντέλο των τριών επιπέδων.

Χαρακτηριστικό είναι ότι πολλά διαγράμματα που δημιουργήθηκαν με τη χρήση του συνόλου **fr2** περιλαμβάνουν τα πακέτα **Frontend**, **Logic** και **Data**, με σχέσεις που προσεγγίζουν την αρχιτεκτονική των τριών επιπέδων. Για την υποστήριξη αυτής της παρατήρησης, θα παρουσιάσουμε συγκεκριμένα παραδείγματα διαγραμμάτων. Αργότερα, στην ενότητα αποτελεσμάτων, θα τεκμηριώσουμε αυτή τη διαπίστωση με γραφήματα και ανάλυση δεδομένων.

Τα παρακάτω δύο διαγράμματα καταδεικνύουν την επίδραση του συνόλου **fr2**, όπως περιγράφηκε παραπάνω. Περιλαμβάνουν τρία διακριτά πακέτα: ένα που αντιπροσωπεύει το **UI/Frontend**, ένα άλλο αφιερωμένο στο **Logic Component**, που περιέχει τη λογική της εφαρμογής, και ένα τρίτο που επικεντρώνεται στη **Database**.

Αυτή η δομή αντικατοπτρίζει τις αρχές της αρχιτεκτονικής Τριών Επιπέδων, αντί να ακολουθεί το ζητούμενο αρχιτεκτονικό πρότυπο: Πελάτη-Εξυπηρετητή για το διάγραμμα στην Εικόνα 4.18 και MVC για το διάγραμμα στην Εικόνα 4.19.

Gemma2:27B | fr2 | nfr2 | rag1 | openai | text-embedding-3-large | semantic (ID = 260)



Εικόνα 4.18 Διάγραμμα με ID = 260

Mixtral:8x7B | fr2 | nfr2 | No rag (ID = 296)

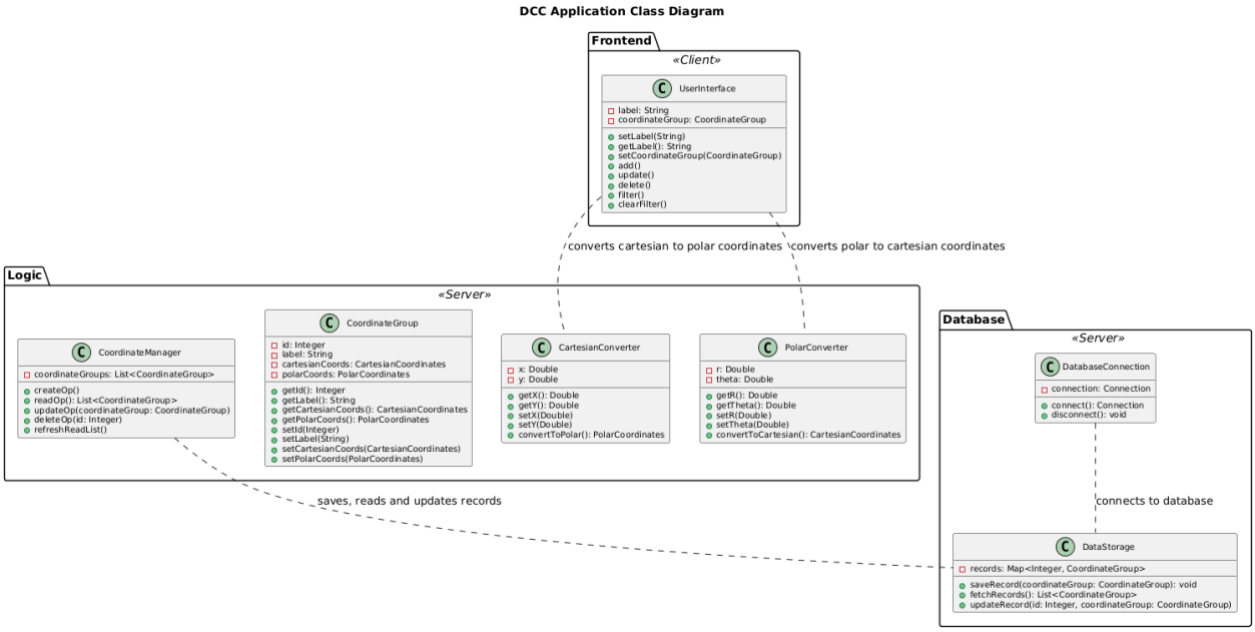
Εικόνα 4.19 Διάγραμμα με ID = 296

Command-r | fr2 | nfr2 | rag2 | openai | text-embedding-3-large | recursive (ID = 349)

Το διάγραμμα στην Εικόνα 4.20 σχεδιάστηκε με σκοπό να υλοποιήσει την ζητούμενη αρχιτεκτονική Πελάτη-Εξυπηρετητή. Ωστόσο, η επιρροή του συνόλου fr2 είναι ξανά εμφανής, καθώς το διάγραμμα οργανώνεται σε τρία πακέτα: Frontend, Logic και Database.

Αυτό που διαφοροποιεί το συγκεκριμένο διάγραμμα από άλλα που επηρεάζονται από το σύνολο fr2 είναι ο σαφής ορισμός των ρόλων του πελάτη και του εξυπηρετητή. Συγκεκριμένα, το πακέτο Frontend επισημαίνεται με το στερεότυπο <<Client>>, ενώ τα πακέτα Logic και Database φέρουν το στερεότυπο <<Server>>.

Παρά τη δομή των τριών πακέτων και τις σχέσεις τους, που συνήθως συνδέονται με την αρχιτεκτονική Τριών Στρωμάτων αυτό το διάγραμμα καταδεικνύει περισσότερη κατανόηση και συμμόρφωση με το πρότυπο Πελάτη-Εξυπηρετητή.



Εικόνα 4.20 Διάγραμμα με ID = 349

4.2 Εξατομικευμένο εργαλείο

Όπως περιγράφηκε στο προηγούμενο κεφάλαιο, χρησιμοποιήσαμε μεγάλα γλωσσικά μοντέλα για τη δημιουργία συνολικά 480 διαγραμμάτων κλάσεων σε μορφή PlantUML. Για να απλοποιήσουμε τη διαδικασία αξιολόγησης και να προσφέρουμε μια βάση για την επέκταση αυτού του έργου στο μέλλον, αναπτύξαμε μια προσαρμοσμένη εφαρμογή λογισμικού. Αυτό το εργαλείο σχεδιάστηκε για να προσφέρει μια οργανωμένη και φιλική προς τον χρήστη διεπαφή για να εξερευνήσει τα παραγόμενα διαγράμματα, ενώ παράλληλα προσφέρει λειτουργίες που διευκολύνουν την αξιολόγησή τους.

Η εφαρμογή περιλαμβάνει αρκετές κύριες λειτουργίες για την υποστήριξη της οπτικοποίησης και της αξιολόγησης των διαγραμμάτων:

1. **Οργανωμένη Εμφάνιση Διαγραμμάτων**
   Το εργαλείο παρέχει μια καθαρή και οργανωμένη διάταξη για την περιήγηση σε όλα τα παραγόμενα διαγράμματα κλάσεων. Τα διαγράμματα κατηγοριοποιούνται και είναι διαθέσιμα προς αναζήτηση βάσει βασικών παραμέτρων, όπως ο τύπος αρχιτεκτονικής (Client-Server, Three-Tier, Model-View-Controller), το επιλεγμένο LLM, ο μοναδικός αναγνωριστικός κωδικός του διαγράμματος ή το χρησιμοποιημένο RAG αρχείο. Αυτή η οργάνωση επιτρέπει στους χρήστες να εντοπίζουν και να αξιολογούν εύκολα και γρήγορα συγκεκριμένα διαγράμματα.

2. **Ενσωματωμένη Απόδοση Διαγραμμάτων**
   Χρησιμοποιώντας το PlantUML .jar αρχείο, το εργαλείο αποδίδει δυναμικά τον κώδικα PlantUML σε εικόνας διαγραμμάτων κλάσεων. Έτσι, οι χρήστες μπορούν να περιηγηθούν στα διαγράμματα απευθείας μέσα από την εφαρμογή, χωρίς να απαιτείται επιπλέον λογισμικό ή χειροκίνητη μετατροπή.

3. **Διεπαφή Αξιολόγησης**
   Το εργαλείο περιλαμβάνει ένα module αξιολόγησης που αντικατοπτρίζει τα κριτήρια που χρησιμοποιήθηκαν στο πείραμα (τήρηση αρχιτεκτονικής, ορθότητα των σχέσεων μεταξύ κλάσεων, συνοχή/συζευκτικότητα, και συνέπεια με τις απαιτήσεις). Οι αξιολογητές μπορούν να εισάγουν τις βαθμολογίες τους για κάθε διάγραμμα, αποθηκεύοντας τα αποτελέσματα σε μια βάση δεδομένων για μελλοντική ανάλυση. Μετά την ολοκλήρωση της αξιολόγησης, οι χρήστες μπορούν επίσης να δουν τις βαθμολογίες που αποδόθηκαν από κάθε LLM για κάθε βασικό κριτήριο αξιολόγησης.

4. **Σύγκριση και Δυνατότητες Φιλτραρίσματος**
   Οι χρήστες μπορούν να συγκρίνουν διαγράμματα που δημιουργήθηκαν υπό διαφορετικά σενάρια (π.χ., με και χωρίς RAG ή μεταξύ διαφορετικών LLMs) σε παράθεση. Οι επιλογές φιλτραρίσματος επιτρέπουν την εμφάνιση διαγραμμάτων βάσει συγκεκριμένων διαμορφώσεων, όπως μοντέλα embedding ή μέθοδοι chunking.

5. **Αποτελέσματα και Αναφορές**
   Η εφαρμογή υποστηρίζει την εξαγωγή δεδομένων αξιολόγησης, διευκολύνοντας τη σύνοψη των ευρημάτων για παρουσιάσεις ή περαιτέρω αναλύσεις.

6. **Επεκτασιμότητα για Μελλοντική Εργασία**
   Το εργαλείο αναπτύχθηκε με γνώμονα την επεκτασιμότητα. Η σχεδίαση επιτρέπει την

εύκολη ενσωμάτωση πρόσθετων λειτουργιών, όπως νέες αρχιτεκτονικές, κριτήρια αξιολόγησης ή modules ανάλυσης.

Παρουσίαση της Εφαρμογής

Στην επόμενη ενότητα, παρέχουμε στιγμιότυπα οθόνης της εφαρμογής μας, αναδεικνύοντας τις βασικές λειτουργίες και τη διάταξή της. Αυτές οι οπτικές παρουσιάσεις δείχνουν πώς η εφαρμογή απλοποιεί τη διαχείριση και την αξιολόγηση διαγραμμάτων, καθιστώντας την ένα απαραίτητο εργαλείο για το έργο μας.

Στα στιγμιότυπα παρακάτω, παρουσιάζονται οι διεπαφές της εξατομικευμένης εφαρμογής που σχεδιάστηκε για την εμφάνιση, αξιολόγηση και ανάλυση των παραγόμενων διαγραμμάτων κλάσεων.

Κεντρική σελίδα: Evaluate diagrams

Η κεντρική σελίδα αποτελείται από τέσσερα κύρια μέρη, καθένα εκ των οποίων εξυπηρετεί μια συγκεκριμένη λειτουργία:

1. **Μέρος Φιλτραρίσματος**

   Αυτό το τμήμα επιτρέπει στους χρήστες ή αξιολογητές να φιλτράρουν τα εμφανιζόμενα αποτελέσματα βάσει διαφόρων παραμέτρων:
   - **Μοντέλο:** Επιλογή του LLM που χρησιμοποιήθηκε για τη δημιουργία του διαγράμματος (π.χ., Llama 3.1, Gemma 2:27b).
   - **Αρχιτεκτονική:** Επιλογή του αρχιτεκτονικού προτύπου, όπως Client-Server, Three-Tier ή MVC.
   - **Απόφαση RAG:** Φιλτράρισμα διαγραμμάτων βάσει της χρήσης ή μη του RAG.
   - **Πρόσθετες Παράμετροι:** Φίλτρα για σύνολα FR/NFR, prompts, embeddings και στρατηγικές chunking.

   Αυτό το μέρος διασφαλίζει ότι οι χρήστες μπορούν γρήγορα να περιορίσουν τα αποτελέσματα και να επικεντρωθούν σε συγκεκριμένα διαγράμματα ή σενάρια ενδιαφέροντος.

2. **Μέρος Αποτελεσμάτων**

   Το τμήμα αυτό εμφανίζει έναν πίνακα με τα φιλτραρισμένα αποτελέσματα, βασισμένα στις παραμέτρους που επιλέχθηκαν στο μέρος φιλτραρίσματος:
   - Κάθε γραμμή του πίνακα αντιπροσωπεύει ένα σενάριο διαγράμματος, περιλαμβάνοντας λεπτομέρειες όπως αρχιτεκτονική, μοντέλο LLM, απόφαση RAG και διαδρομή αποτελέσματος.
   - Οι χρήστες μπορούν να επιλέξουν μια γραμμή για να προβάλλουν και να αξιολογήσουν το αντίστοιχο διάγραμμα. Αυτή η ολοκληρωμένη ενσωμάτωση διασφαλίζει μια ομαλή ροή εργασίας για την επιλογή και ανάλυση διαγραμμάτων.

3. **Φόρμα Αξιολόγησης**

   Αυτό το μέρος επικεντρώνεται στην αναθεώρηση και αξιολόγηση του επιλεγμένου διαγράμματος, περιλαμβάνοντας διάφορα διαδραστικά στοιχεία:
   - **Καρτέλα Διαγραμμάτων:** Εμφανίζει το επιλεγμένο διάγραμμα και περιλαμβάνει βελάκια πλοήγησης, επιτρέποντας στους χρήστες να περιηγούνται εύκολα σε άλλα διαγράμματα του συνόλου δεδομένων.

- **Άξονες Αξιολόγησης:** Κάτω από το διάγραμμα, οι αξιολογητές μπορούν να βαθμολογήσουν το διάγραμμα με βάση τα κύρια κριτήρια, όπως:
    - Τήρηση της αρχιτεκτονικής.
    - Ορθότητα σχέσεων των κλάσεων.
    - Συνοχή και συζευκτικότητα.
    - Συνέπεια με τις απαιτήσεις λογισμικού.

- **Κουμπιά Δράσης:** Παρέχονται πρόσθετες λειτουργίες, όπως:
    - **Επεξεργασία Διαγράμματος:** Επιτρέπει στους χρήστες να κάνουν τροποποιήσεις στο διάγραμμα.
    - **Σύγκριση Διαγραμμάτων:** Παρέχει παράθεση δύο διαγραμμάτων για συγκριτική αξιολόγηση.
    - **Προβολή Πλήρους Οθόνης:** Προσφέρει μια ανεμπόδιστη, μεγενθυμένη προβολή του διαγράμματος.
    - **Πλοήγηση μέσω Sidebar:** Ανοίγει ένα πλαϊνό μενού με μικρογραφίες όλων των διαγραμμάτων, επιτρέποντας άμεση πλοήγηση σε οποιοδήποτε διάγραμμα ενδιαφέροντος.
    - **Προβολή Γραφήματος Στηλών:** Εάν ένας εγγεγραμμένος χρήστης έχει ήδη ολοκληρώσει τη βαθμολόγηση, αυτό το κουμπί ενεργοποιεί την εμφάνιση ενός γραφήματος στηλών (bar chart). Το γράφημα παρέχει μια οπτική απεικόνιση των βαθμολογιών, διευκολύνοντας τη σύγκριση και την ανάλυση των αποτελεσμάτων για το συγκεκριμένο διάγραμμα μεταξύ των διαφορετικών LLMs και της βαθμολογίας του χρήστη.

4. **Μέρος Μεταδεδομένων**

    Αυτό το τμήμα παρέχει λεπτομερή μεταδεδομένα για το επιλεγμένο διάγραμμα, περιλαμβάνοντας:
    - **Λειτουργικές και Μη Λειτουργικές Απαιτήσεις (FR/NFR):** Εμφανίζει το κείμενο των σετ απαιτήσεων που χρησιμοποιήθηκαν ως είσοδο για τη δημιουργία του διαγράμματος.
    - **Prompt:** Παρουσιάζει το ακριβές κείμενο του prompt που δόθηκε στο LLM για τη δημιουργία του διαγράμματος.
    - **Αρχική Απόκριση LLM:** Εμφανίζει την απόκριση που παρήχθη από το LLM.
    - **RAG Αρχείο (εφόσον υπάρχει):** Δείχνει αν χρησιμοποιήθηκε RAG και, αν ναι, περιλαμβάνει το περιεχόμενο του αρχείου RAG που χρησιμοποιήθηκε για τη συμπλήρωση της εξόδου του LLM.

    Αυτό το τμήμα διασφαλίζει τη διαφάνεια και την ιχνηλασιμότητα, προσφέροντας στους αξιολογητές πλήρες πλαίσιο για κάθε διάγραμμα.

Η διεπαφή σχεδιάστηκε προσεκτικά για να ενισχύσει τη χρηστικότητα και να διασφαλίσει ότι οι αξιολογητές μπορούν να αλληλεπιδράσουν με τα διαγράμματα αποδοτικά και εύκολα.

Δευτερεύουσα σελίδα: Αξιολογήσεις από LLMs

Το δεύτερο στιγμιότυπο παρουσιάζει τη σελίδα **Αξιολογήσεων από LLMs** της εφαρμογής, η οποία είναι αφιερωμένη στην εμφάνιση των αξιολογήσεων που παρέχονται από μεγάλα γλωσσικά μοντέλα (LLMs). Αυτή η σελίδα διατηρεί αρκετά βασικά στοιχεία από τη προηγούμενη διεπαφή, όπως τα μέρη φιλτραρίσματος, αποτελεσμάτων και μεταδεδομένων, αλλά εισάγει νέες λειτουργίες για την παρουσίαση λεπτομερειών των αξιολογήσεων από τα LLMs.

Αντί της φόρμας αξιολόγησης που εμφανιζόταν στην προηγούμενη σελίδα, αυτή η ενότητα προβάλλει τις αξιολογήσεις του LLM για το επιλεγμένο σενάριο. Οι αξιολογήσεις περιλαμβάνουν:

- **Βαθμολογίες:** Αριθμητικές βαθμολογίες (σε κλίμακα από 0 έως 5) για κάθε κριτήριο αξιολόγησης:
    - Τήρηση Αρχιτεκτονικής.
    - Ορθότητα Σχέσεων των Κλάσεων.
    - Συνοχή και Συζευκτικότητα.
    - Συνέπεια με τις Απαιτήσεις Λογισμικού.

- **Επιχειρηματολογία Αξιολόγησης:** Αναλυτική επεξήγηση σε κείμενο για κάθε βαθμολογία, προσφέροντας πληροφορίες για τη λογική αξιολόγησης του LLM. Αυτές οι δικαιολογήσεις βοηθούν τους χρήστες να κατανοήσουν πώς το μοντέλο ερμηνεύει τις αρχιτεκτονικές αρχές και τις απαιτήσεις, αναδεικνύοντας δυνατά σημεία και πιθανές αδυναμίες στα διαγράμματα κλάσεων. Εκτός από τις βαθμολογίες, τα LLMs παρέχουν και πρακτική ανατροφοδότηση σχετικά με το πώς το διάγραμμα μπορεί να βελτιωθεί.

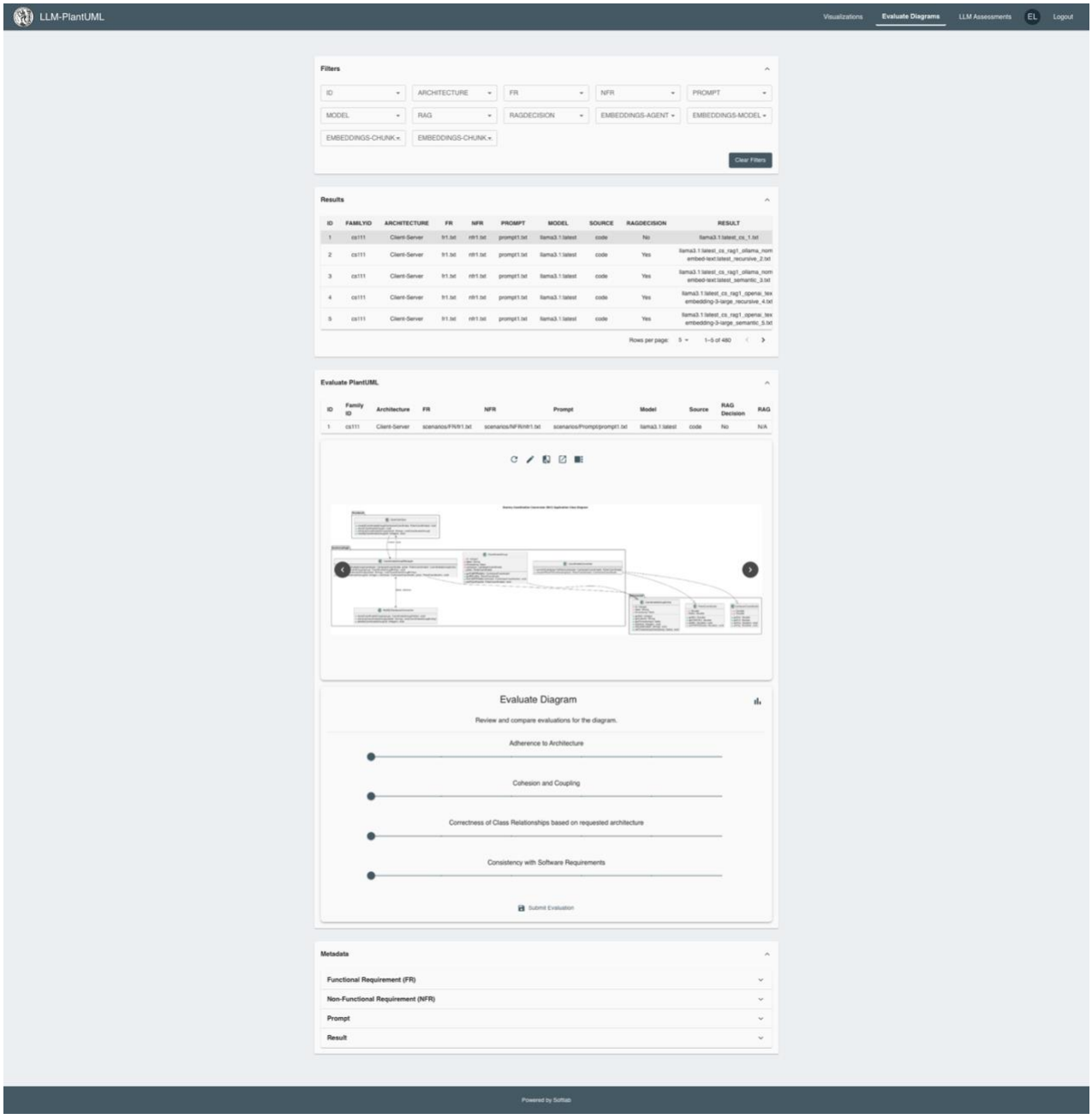
Αυτή η σελίδα προσφέρει μια διαφανή εικόνα των δυνατοτήτων των LLMs στην αξιολόγηση διαγραμμάτων κλάσεων. Με την προβολή τόσο ποσοτικών βαθμολογιών όσο και ποιοτικής λογικής, δίνεται η δυνατότητα άμεσης σύγκρισης μεταξύ των αξιολογήσεων ανθρώπων και AI.

Η διεπαφή αυτή απλοποιεί την ανάλυση των αξιολογήσεων από LLMs και διευκολύνει μια βαθύτερη κατανόηση του τρόπου με τον οποίο διαφορετικά μοντέλα ερμηνεύουν και αξιολογούν αρχιτεκτονικά διαγράμματα. Αποτελεί κρίσιμο στοιχείο της συνολικής ροής εργασίας της εφαρμογής.

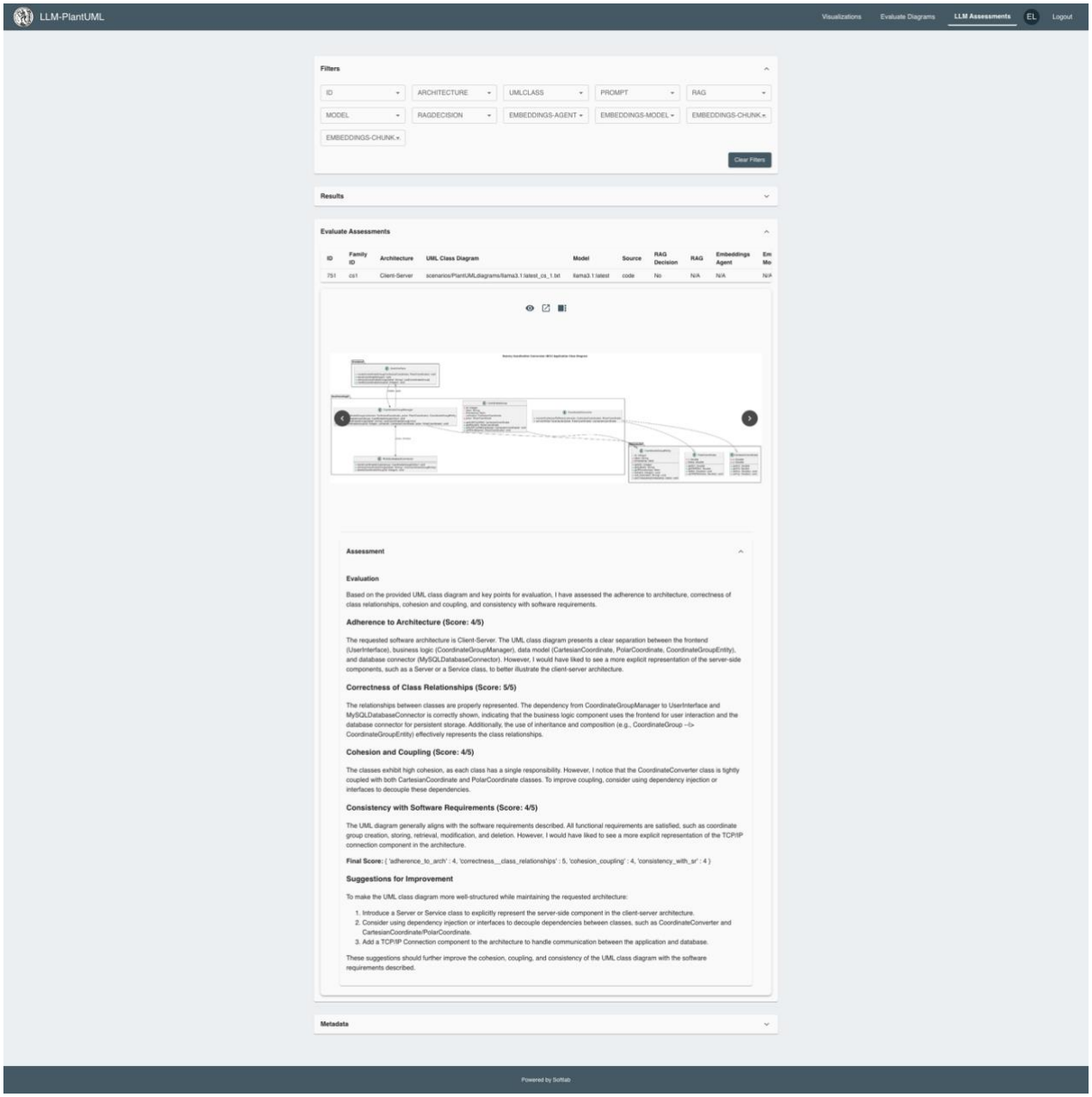
Επιπλέον Λειτουργικότητα: Στατιστική Ανάλυση

Η εφαρμογή διαθέτει επίσης μια τρίτη σελίδα, αφιερωμένη στην οπτικοποίηση στατιστικών αναλύσεων των δεδομένων αξιολόγησης μέσω γραφημάτων. Αυτά τα γραφήματα προσφέρουν πληροφορίες για πρότυπα και τάσεις στα αποτελέσματα των αξιολογήσεων. Οι λεπτομέρειες αυτών των γραφημάτων θα συζητηθούν εκτενώς στην ενότητα αποτελεσμάτων.

Η Εικόνα 4.21 και η Εικόνα 4.22 απεικονίζουν τις δύο κεντρικές σελίδες της εφαρμογής μας.



Εικόνα 4.21 Evaluate diagrams page



Εικόνα 4.22 LLM assessments page

4.3 Αποτελέσματα αξιολόγησης

Σε αυτή την ενότητα, παρουσιάζουμε γραφήματα που δημιουργήθηκαν από τα δεδομένα αξιολόγησης, εστιάζοντας στις πληροφορίες που προκύπτουν από τις αξιολογήσεις κυρίως από ανθρώπους αλλά και από LLMs. Όλα τα γραφήματα που παρουσιάζονται δημιουργήθηκαν χρησιμοποιώντας τη βιβλιοθήκη Highcharts, για καθαρή και διαδραστική απεικόνιση.

4.3.1 Επισκόπηση αξιολογήσεων από LLMs

Ξεκινάμε με την ανάλυση των βαθμολογιών που δόθηκαν από τα LLMs. Τα παρακάτω γραφήματα απεικονίζουν τις βαθμολογίες που δόθηκαν κατά μέσο όρο από όλα τα LLMs σε κάθε κατηγορία αξιολόγησης, καθώς και μια ανάλυση των βαθμολογιών που δόθηκαν από κάθε επιμέρους μοντέλο. Επιπλέον, συγκρίνουμε αυτές τις βαθμολογίες με εκείνες που αποδόθηκαν από τους ανθρώπους, ώστε να εντοπιστούν τάσεις και αποκλίσεις.

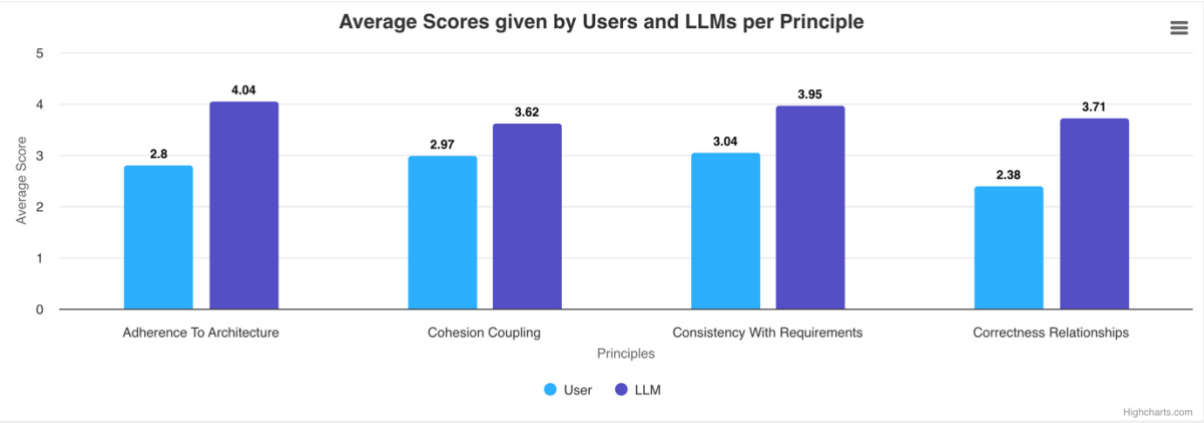
Κατά τη διάρκεια της αξιολόγησης μας, παρατηρήθηκε ότι τα LLMs δεν είναι ακόμη σε θέση να αξιολογούν διαγράμματα κλάσεων με δίκαιο και ακριβή τρόπο. Ένα αξιοσημείωτο φαινόμενο είναι ότι τα LLMs τείνουν να αποδίδουν υψηλότερες βαθμολογίες σε σύγκριση με τους ανθρώπους, ακόμη και για διαγράμματα που εμφανώς αποτυγχάνουν σε όλα τα κριτήρια αξιολόγησης.

Αυτή η τάση αποτυπώνεται στα παρακάτω γραφήματα, τα οποία δείχνουν τη γενική υπερτίμηση των διαγραμμάτων από τα LLMs, ενισχύοντας την ανάγκη για περαιτέρω βελτίωση της ικανότητάς τους να παρέχουν ακριβείς και αξιόπιστες αξιολογήσεις.

Το διάγραμμα στην Εικόνα 4.23 συγκρίνει τους μέσους όρους βαθμολογιών που αποδόθηκαν από τα LLMs με εκείνους των χρηστών, χρησιμοποιώντας μορφή γραφήματος με ράβδους. Τα δεδομένα αναδεικνύουν σημαντικές διαφορές, ιδιαίτερα στις κατηγορίες «**Συμμόρφωση με την Αρχιτεκτονική**» και «**Ορθότητα Σχέσεων Κλάσεων**».

- **Συμμόρφωση με την Αρχιτεκτονική**: Τα LLMs συχνά αποτυγχάνουν να αναλύσουν κριτικά εάν τηρούνται οι αρχές του ζητούμενου αρχιτεκτονικού προτύπου, με αποτέλεσμα να αποδίδουν αναντίστοιχα υψηλές βαθμολογίες.
- **Ορθότητα Σχέσεων Κλάσεων**: Αυτή η κατηγορία παρουσιάζει επίσης αξιοσημείωτο χάσμα, πιθανώς επειδή η ακριβής αξιολόγηση των σχέσεων μεταξύ κλάσεων απαιτεί βαθύτερη κατανόηση της προτεινόμενης λογικής σχεδιασμού του συστήματος—ένα πεδίο όπου η ανθρώπινη εμπειρία είναι πιο καθοριστική.

Αυτές οι αποκλίσεις υπογραμμίζουν τους περιορισμούς των LLMs στην παροχή λεπτομερών και ακριβών αξιολογήσεων, ιδιαίτερα για κριτήρια που απαιτούν κριτική σκέψη και αρχιτεκτονική κατανόηση.



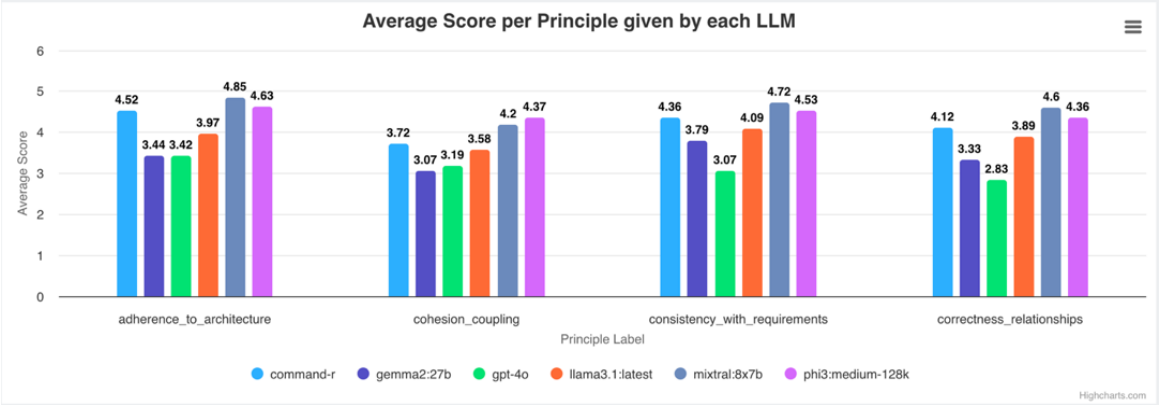
Εικόνα 4.23 Average Score by Users vs LLMs

Το γράφημα στην Εικόνα 4.24 παρουσιάζει μια σύγκριση των μέσων βαθμολογιών που αποδόθηκαν από κάθε LLM ξεχωριστά. Τα αποτελέσματα αναδεικνύουν ορισμένα μοντέλα, όπως τα **command-r** , **mixtral:8x7b** και **phi3:medium-128k**, τα οποία φαίνεται να είναι ακατάλληλα για

την αξιολόγηση διαγραμμάτων κλάσεων. Αυτά τα μοντέλα αποδίδουν συστηματικά υπερβολικά υψηλές βαθμολογίες, που συχνά ξεπερνούν το 4/5 σε σχεδόν όλες τις κατηγορίες, ανεξάρτητα από την ποιότητα των διαγραμμάτων.

Αντίθετα, υπάρχουν μοντέλα όπως τα **gemma2:27b** και **gpt-4o**, των οποίων οι μέσες βαθμολογίες φαίνεται να ευθυγραμμίζονται περισσότερο με τις αξιολογήσεις των χρηστών. Ωστόσο, αυτή η φαινομενική ευθυγράμμιση δεν αντικατοπτρίζεται με συνέπεια στην ανάλυση των μεμονωμένων διαγραμμάτων ή στις παρατηρήσεις μας σχετικά με τη συμπεριφορά βαθμολόγησης των LLMs.

Συνολικά, τα δεδομένα αλλά και οι παρατηρήσεις μας κατά τη διάρκεια της διαδικασίας αξιολόγησης ενισχύουν το συμπέρασμα ότι τα LLMs γενικά παρουσιάζουν χαμηλή απόδοση στην αξιολόγηση διαγραμμάτων κλάσεων, με έντονη τάση να υπερεκτιμούν τα διαγράμματα σε διάφορα κριτήρια. Ως εκ τούτου, για το υπόλοιπο της ανάλυσης, επικεντρωνόμαστε στις αξιολογήσεις των χρηστών, ώστε να κατανοήσουμε καλύτερα την επίδραση των πειραματικών παραμέτρων στην ποιότητα των παραγόμενων διαγραμμάτων.



Εικόνα 4.24 Average Score given by each LLM

4.3.2 Επισκόπηση αξιολογήσεων από ανθρώπους

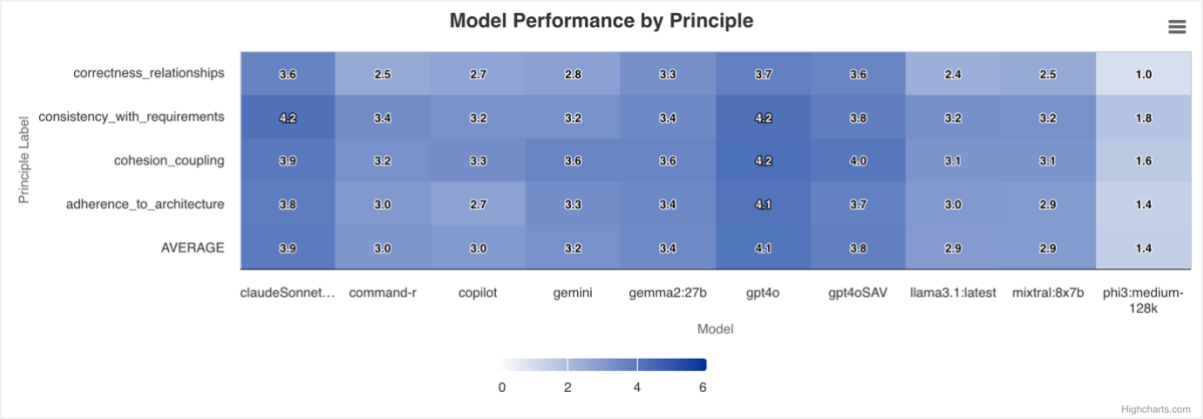
Το γράφημα στην Εικόνα 4.25 παρουσιάζει τους μέσους όρους βαθμολογιών των χρηστών για τα διαγράμματα κλάσεων που δημιουργήθηκαν από κάθε LLM, συγκεντρωμένα από όλες τις πειραματικές παραμέτρους. Οφείλουμε να σημειώσουμε ότι τα online μοντέλα (ClaudeSonnet3.5, Copilot, Gemini, GPT-4o, GPT-4o Software-Architecture-Visualizer) παρήγαγαν διαγράμματα μόνο χωρίς τη χρήση RAG. Κατά συνέπεια, οι παράμετροι που διαφοροποιήθηκαν για αυτά τα μοντέλα περιορίστηκαν στην ζητούμενη αρχιτεκτονική και στα σύνολα FR/NFR, γεγονός που είχε ως αποτέλεσμα τον μικρότερο αριθμό παραγόμενων διαγραμμάτων σε σύγκριση με τα τοπικά μοντέλα.

Παρά τον περιορισμό αυτό, τα αποτελέσματα δείχνουν ξεκάθαρα ότι τα πιο προηγμένα online μοντέλα, ιδιαίτερα τα **ClaudeSonnet3.5**, **GPT-4o** και η εξειδικευμένη έκδοσή του GPT-4o Software-Architecture-Visualizer, πέτυχαν τις υψηλότερες μέσες βαθμολογίες σε όλα τα κριτήρια αξιολόγησης. Τα μοντέλα αυτά υπερέβησαν συστηματικά τα τοπικά LLMs στη συμμόρφωση με τις αρχιτεκτονικές αρχές, στις σχέσεις κλάσεων, στη συνοχή και στην ευθυγράμμιση με τις απαιτήσεις.

Ωστόσο, η απόδοση των τοπικών μοντέλων δεν πρέπει να υποτιμηθεί. Αν και το **phi3:medium-128k** παρήγαγε απογοητευτικά αποτελέσματα, με μέσο όρο μόλις 1,4/5, άλλα τοπικά μοντέλα έδειξαν αξιοσημείωτα καλή απόδοση. Συγκεκριμένα, τα **Gemma2:27b** και

Command-r ξεχώρισαν, με βαθμολογίες που συχνά ανταγωνίζονταν ή ξεπερνούσαν αυτές των online μοντέλων σε πολλές περιπτώσεις. Επιπλέον, τα **Llama3.1:latest** και **Mixtral:8x7b** παρουσίασαν αξιοπρεπή, αν και λιγότερο εντυπωσιακή, απόδοση.

Συνολικά, το **GPT-4o** κυριάρχησε όσον αφορά τη μέση απόδοση, επιβεβαιώνοντας τη θέση του ως κορυφαίο μοντέλο. Παρ' όλα αυτά, η αξιοσημείωτη απόδοση του **Gemma2:27b**, ενός τοπικού μοντέλου, αξίζει ιδιαίτερης προσοχής και περαιτέρω διερεύνησης για τη δυναμική του σε εφαρμογές που διαχειρίζονται ευαίσθητα δεδομένα.



Εικόνα 4.25 Model performance based on human evaluations

Στη συνέχεια θα παρουσιαστούν διαγράμματα που στοχεύουν στην κατανόηση της επίδρασης των διαφορετικών πειραματικών παραμέτρων στα παραγόμενα διαγράμματα κλάσεων, ξεκινώντας με την ανάλυση της επίδρασης του **RAG (Retrieval-Augmented Generation)**.

Τα γραφήματα στην Εικόνα 4.26 και στην Εικόνα 4.27 απεικονίζουν τους μέσους όρους βαθμολογιών των διαγραμμάτων που δημιουργήθηκαν από κάθε LLM, διαχωρισμένα σε αυτά που η έξοδος ενισχύθηκε μέσω RAG και σε εκείνα που δημιουργήθηκαν χωρίς τη χρήση του. Τα αποτελέσματα είναι ενδιαφέροντα και ενθαρρυντικά, προσφέροντας κίνητρα για μελλοντικές βελτιώσεις.

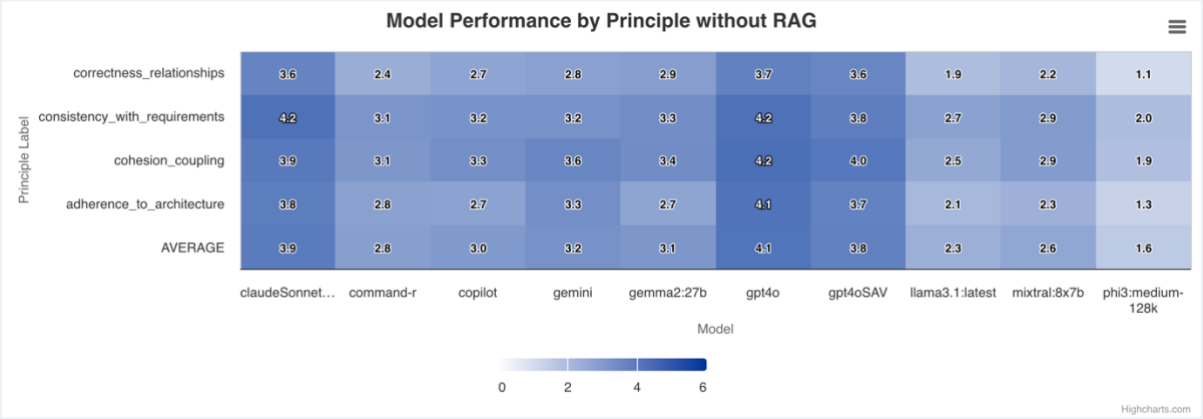
Σχεδόν όλα τα τοπικά μοντέλα παρουσιάζουν σημαντική βελτίωση στις βαθμολογίες αξιολόγησης όταν τα αποτελέσματά τους ενισχύονται μέσω RAG. Ωστόσο, το **phi3:medium-128k** αποτελεί εξαίρεση, καθώς ο μέσος όρος της βαθμολογίας του μειώνεται από 1.6 σε 1.4 όταν εφαρμόζεται το RAG. Αυτή η πτώση μπορεί να αποδοθεί στην τάση του μοντέλου να συγχέεται από το πρόσθετο περιεχόμενο που παρέχουν τα αρχεία RAG. Όπως επισημάνθηκε στην ενότητα 0 για τα αδύναμα διαγράμματα, το **phi3:medium-128k** συχνά παράγει άσχετα ή ασυνεπή διαγράμματα, επηρεαζόμενο από τα παραδείγματα και το κείμενο στη διαδικασία του RAG. Συνεπώς παρατηρούμε μια εγγενή αδυναμία του συγκεκριμένου μοντέλου να διαχειριστεί εκτεταμένα και πολύπλοκα prompts.

Για τα υπόλοιπα LLMs, η χρήση του RAG οδηγεί σε αξιοσημείωτες βελτιώσεις στην ποιότητα των διαγραμμάτων. Στα πιο χαρακτηριστικά παραδείγματα περιλαμβάνονται:

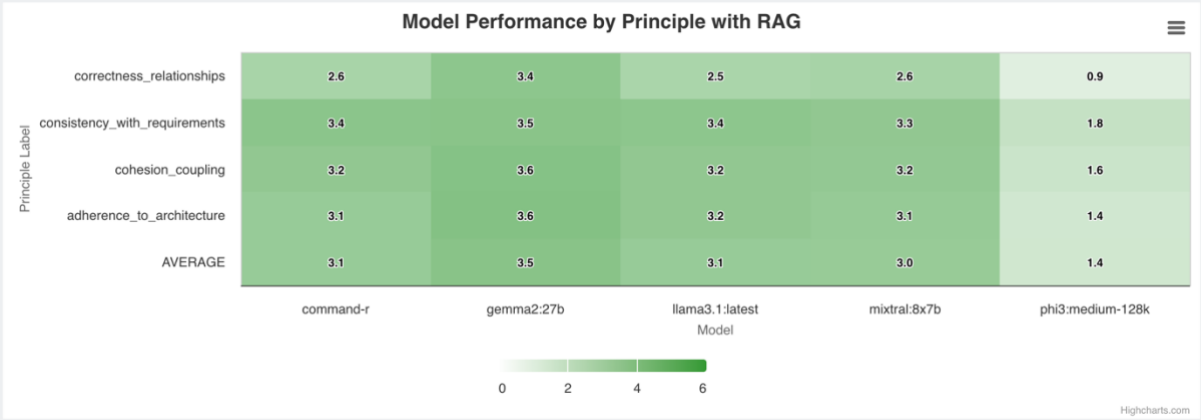
- **Command-r:** Η μέση βαθμολογία βελτιώνεται από 2.8 σε 3.1 .
- **Gemma2:27b:** Η μέση βαθμολογία αυξάνεται από 3.1 στο εντυπωσιακό 3.5 ξεπερνώντας ακόμα και τα **Gemini** και **Copilot**.
- **Llama3.1:latest:** Η μέση βαθμολογία αυξάνεται από 2.3 σε 3.1.
- **Mixtral:8x7b:** Η μέση βαθμολογία βελτιώνεται από 2.6 σε 3.0.

Τα αποτελέσματα υπογραμμίζουν τη δυναμική του RAG στη βελτίωση της απόδοσης των τοπικών LLMs στη δημιουργία διαγραμμάτων κλάσεων. Ενώ το RAG βοηθά τα μοντέλα να κατανοήσουν και να δομήσουν καλύτερα το περιεχόμενο, η αποτελεσματικότητά του εξαρτάται από την ικανότητα του εκάστοτε μοντέλου να επεξεργάζεται τις πρόσθετες πληροφορίες. Για τα περισσότερα τοπικά μοντέλα, το RAG παρέχει ένα σαφές πλεονέκτημα, με το **Gemma2:27b** να ξεχωρίζει ως κορυφαίο τοπικό μοντέλο, πετυχαίνοντας αποτελέσματα συγκρίσιμα με ορισμένα online μοντέλα.

Αυτό ενισχύει τη σημασία της περαιτέρω βελτίωσης τόσο της διαδικασίας RAG όσο και της επιλογής μοντέλων embeddings, προκειμένου να βελτιστοποιηθούν ακόμη περισσότερο τα παραγόμενα αποτελέσματα.



Εικόνα 4.26 Model Performance without RAG based on human evaluations



Εικόνα 4.27 Model Performance with RAG based on human evaluations

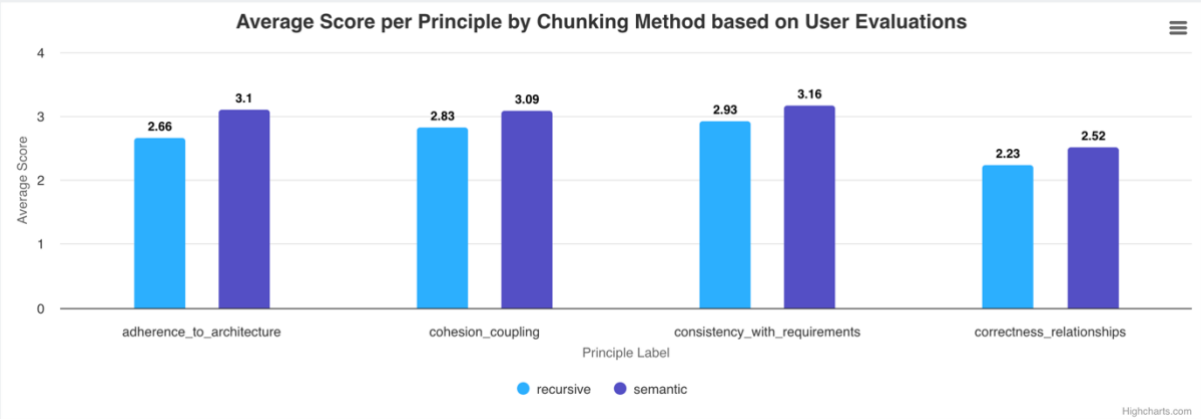
Η τεχνική **RAG (Retrieval-Augmented Generation)** περιλαμβάνει διάφορες παραμέτρους που επηρεάζουν σημαντικά την αποτελεσματικότητά της στη βελτίωση των αποκρίσεων των LLMs. Αυτές οι παράμετροι περιλαμβάνουν το μοντέλο embeddings, τη μέθοδο τμηματοποίησης (chunking), καθώς και το σχεδιασμό των εγγράφων RAG. Κάθε ένα από αυτά τα στοιχεία διαδραματίζει κρίσιμο ρόλο στο πόσο καλά ευθυγραμμίζεται το επαυξημένο περιεχόμενο που παρέχεται από το RAG με τις απαιτήσεις και στη βελτίωση των εξόδων των LLMs.

Μια σημαντική παράμετρος της τεχνικής RAG αφορά την απόδοση του μοντέλου embeddings και της μεθόδου τμηματοποίησης. Αρχικά το παρέχει μια άμεση σύγκριση των δύο μεθόδων τμηματοποίησης που χρησιμοποιήθηκαν: **Semantic Chunking** και **Recursive Chunking**.

Το γράφημα στην Εικόνα 4.28 δείχνει μια εμφανή υπεροχή της **Semantic Chunking** έναντι της **Recursive Chunking** σε όλα τα κριτήρια αξιολόγησης. Αυτό το αποτέλεσμα ευθυγραμμίζεται με τις προσδοκίες, καθώς η Semantic Chunking είναι μια πιο προηγμένη τεχνική που τμηματοποιεί δυναμικά τα δεδομένα με βάση το περιεχόμενο και τη σημασία τους. Ομαδοποιώντας σημασιολογικά σχετικές πληροφορίες σε συνεκτικά κομμάτια, διασφαλίζει ότι το ανακτημένο chunk είναι ομοιογενές και σχετικό με το ερώτημα.

Αντίθετα, η **Recursive Chunking** υιοθετεί μια πιο άκαμπτη προσέγγιση, διαχωρίζοντας το κείμενο με βάση δομικά στοιχεία όπως χαρακτήρες ή παραγράφους. Αν και είναι πιο απλή στην υλοποίηση, αυτή η μέθοδος δεν λαμβάνει υπόψη τη σημασία του κειμένου, με αποτέλεσμα τη δημιουργία λιγότερο ακριβών και σχετικών τμημάτων. Ως εκ τούτου, τα διαγράμματα που δημιουργούνται χρησιμοποιώντας τη Recursive Chunking τείνουν να λαμβάνουν χαμηλότερες βαθμολογίες.

Αυτό το εύρημα ενισχύει την ανάγκη για προτεραιοποίηση προηγμένων στρατηγικών τμηματοποίησης σε μελλοντικές υλοποιήσεις του RAG, προκειμένου να μεγιστοποιηθεί το δυναμικό του στη βελτίωση της απόδοσης των LLMs.

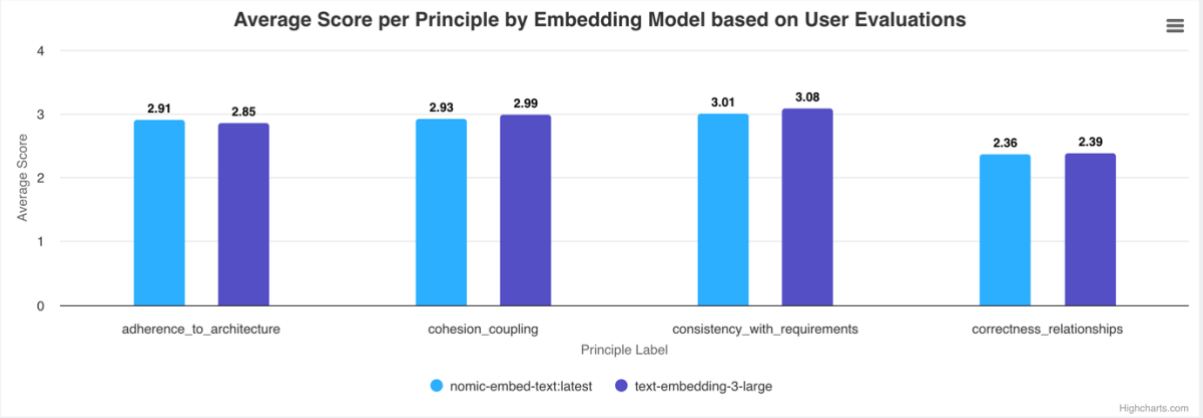


Εικόνα 4.28 Average Score by chunking method based on human evaluations

Σε αντίθεση με την διακριτή υπεροχή που παρατηρήθηκε στις μεθόδους τμηματοποίησης, η σύγκριση των μοντέλων embeddings αποκαλύπτει μια πιο ισορροπημένη απόδοση. Τα αποτελέσματα στην Εικόνα 4.29 δείχνουν ότι το **nomic-embed-text:latest** παρουσιάζει ένα εκπληκτικό επίπεδο ανταγωνιστικότητας σε σχέση με το εμπορικό, cloud-based μοντέλο **text-embedding-3-large**.

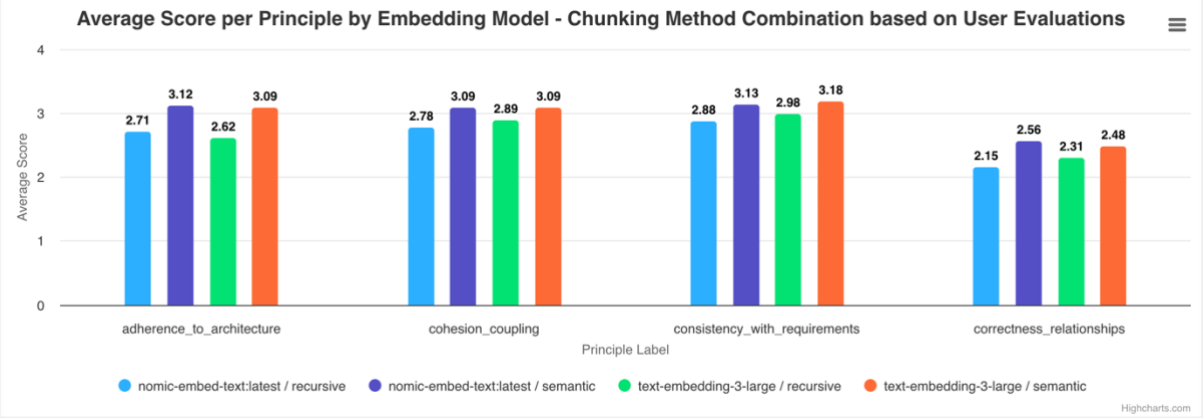
Συγκεκριμένα, το nomic-embed-text:latest επιτυγχάνει ελαφρώς υψηλότερες βαθμολογίες στην κατηγορία «Συμμόρφωση με την Αρχιτεκτονική», ενώ το text-embedding-3-large παρουσιάζει ελαφρώς καλύτερη απόδοση στα υπόλοιπα τρία κριτήρια αξιολόγησης. Παρά αυτές τις μικρές διαφορές, τα αποτελέσματα υπογραμμίζουν ένα σημαντικό σημείο: το nomic-embed-text:latest, ως ένα δωρεάν, ανοιχτού κώδικα μοντέλο embeddings που μπορεί να αναπτυχθεί τοπικά, προσφέρει μια αξιόπιστη εναλλακτική λύση στο εμπορικό, επί πληρωμή text-embedding-3-large.

Αυτή η σύγκριση υπογραμμίζει την αυξανόμενη δυναμική των τοπικών, ανοιχτού κώδικα μοντέλων να ανταγωνιστούν τα εμπορικά αντίστοιχα στις εργασίες embeddings, παρέχοντας μια οικονομικά αποδοτική και ασφαλή λύση για όσους δίνουν προτεραιότητα στην ιδιωτικότητα και την ανεξαρτησία από υπηρεσίες βασισμένες στο cloud.



Εικόνα 4.29 Average Score by embedding model based on human evaluations

Το γράφημα στην Εικόνα 4.30 απεικονίζει τις βαθμολογίες που έλαβαν τα διαγράμματα, αυτή τη φορά ομαδοποιημένα σύμφωνα με τον συνδυασμό του μοντέλου embeddings και της μεθόδου τμηματοποίησης. Τα αποτελέσματα δείχνουν μια σχετικά ισορροπημένη απόδοση σε όλους τους συνδυασμούς, χωρίς να παρατηρούνται σημαντικές αποκλίσεις. Οι βαθμολογίες ευθυγραμμίζονται με τις τάσεις που σημειώθηκαν στα δύο προηγούμενα γραφήματα, αντικατοπτρίζοντας συνεπείς τάσεις ως προς το πώς αυτές οι παράμετροι επηρεάζουν την ποιότητα των παραγόμενων διαγραμμάτων.



Εικόνα 4.30 Average Score by embedding model - chunking method based on human evaluations

Η επίδραση της παραμέτρου των αρχείων **RAG** στην απόδοση των παραγόμενων διαγραμμάτων κλάσεων συνιστά μία ακόμα καθοριστική παράμετρο για το πείραμα μας. Όπως συζητήθηκε στην ενότητα 3.1.5 , πραγματοποιήσαμε πειράματα με τρεις διακριτούς τύπους αρχείων **RAG** :

1. **RAG1:** Αυτό το αρχείο περιέχει εκτενή γνώση για τις επιλεγμένες αρχιτεκτονικές, αντλημένη από το θεμελιώδες κείμενο *Software Engineering, 10th Edition* του Ian Sommerville. Λειτουργεί ως μια υψηλής ποιότητας και αυθεντική πηγή πληροφοριών.

2. **RAG2, RAG3, RAG4:** Σε αυτή την προσέγγιση, το περιεχόμενο του **RAG1** χωρίστηκε σε ξεχωριστά αρχεία, ειδικά για κάθε αρχιτεκτονική. Για παράδειγμα:
   - **RAG2** περιέχει μόνο το περιεχόμενο που αφορά την αρχιτεκτονική **Client-Server**.
   - **RAG3** επικεντρώνεται αποκλειστικά στην αρχιτεκτονική **Three-Tier**.
   - **RAG4** είναι ειδικό για την αρχιτεκτονική **Model-View-Controller**.

   Αυτή η στοχευμένη προσέγγιση διασφαλίζει ότι τα LLMs λαμβάνουν πληροφορίες

που αφορούν άμεσα την ζητούμενη αρχιτεκτονική.

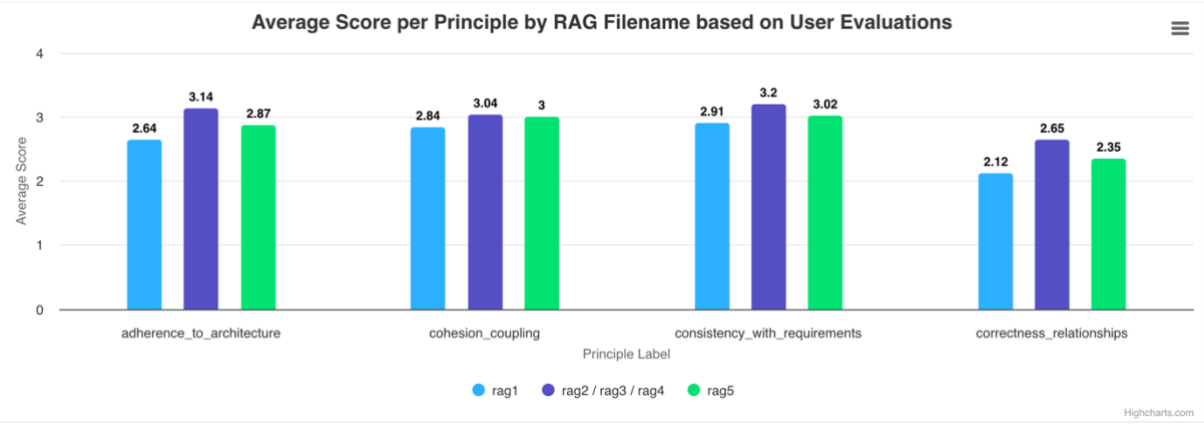
3. **RAG5:** Αυτό το αρχείο συγκεντρώνει πληροφορίες από δημόσια διαθέσιμες πηγές, όπως η Wikipedia και το GeeksforGeeks, αντιπροσωπεύοντας ένα πιο γενικευμένο και λιγότερο επιμελημένο σύνολο δεδομένων.

Το γράφημα στην Εικόνα 4.31 αποκαλύπτει αρκετές αξιοσημείωτες τάσεις σχετικά με την επίδραση των τύπων αρχείων **RAG** στην ποιότητα των παραγόμενων διαγραμμάτων κλάσεων. Η προσέγγιση που απέδωσε τα καλύτερα αποτελέσματα ήταν ο διαχωρισμός του περιεχομένου σε αρχιτεκτονικά εξειδικευμένα αρχεία (**RAG2, RAG3, RAG4**). Παρέχοντας στα LLMs στοχευμένες και σχετικές πληροφορίες, αυτή η μέθοδος ελαχιστοποίησε τη σύγχυση και διασφάλισε ότι το πλαίσιο ευθυγραμμίστηκε άμεσα με την αιτούμενη αρχιτεκτονική, ανεξάρτητα από την ακρίβεια του **RAG**. Αυτή η εξειδίκευση φαίνεται να ενισχύει την ικανότητα των μοντέλων να παράγουν ακριβή και συνεκτικά διαγράμματα κλάσεων.

Ίσως προς έκπληξη μας, η δεύτερη καλύτερη απόδοση προήλθε από το **RAG5**, το οποίο βασίστηκε σε δημόσιες πηγές, όπως η Wikipedia και το GeeksforGeeks. Παρόλο που ήταν λιγότερο αυστηρό και πιθανώς λιγότερο αξιόπιστο, αυτή η προσέγγιση ξεπέρασε το **RAG1**, το οποίο αντλήθηκε από το αυθεντικό εγχειρίδιο του Ian Sommerville. Αυτό το αποτέλεσμα υποδηλώνει ότι η ποικιλία και η προσβασιμότητα των πληροφοριών από διαδικτυακές πηγές μπορεί να προσφέρουν στα LLMs ένα ευρύτερο και πιο πρακτικό πλαίσιο κατανόησης των αρχιτεκτονικών απαιτήσεων, ακόμα κι αν το υλικό είναι λιγότερο επίσημο.

Αντίθετα, το **RAG1**, το οποίο προέρχεται από ένα θεμελιώδες ακαδημαϊκό κείμενο, παρουσίασε σχετικά ασθενέστερη απόδοση. Παρόλο που οι πληροφορίες ήταν υψηλής ποιότητας, το ευρύτερο πεδίο και η λιγότερο στοχευμένη δομή τους ενδέχεται να περιόρισαν την αποτελεσματικότητά τους. Τα αποτελέσματα υποδεικνύουν ότι ακόμη και τα δεδομένα υψηλής ποιότητας πρέπει να παρέχονται με ακριβή και στοχευμένο τρόπο, ώστε να μεγιστοποιηθεί η χρησιμότητά τους για τα LLMs.

Αυτά τα ευρήματα υπογραμμίζουν τον κρίσιμο ρόλο των στοχευμένων και συναφών πληροφοριών στην καθοδήγηση της απόδοσης των LLMs. Επιπλέον, η ισχυρή απόδοση των διαδικτυακών πηγών τονίζει τις δυνατότητες αξιοποίησης προσβάσιμων και ποικιλόμορφων συνόλων δεδομένων για τη βελτίωση των εργασιών με τη βοήθεια AI, υποδεικνύοντας μια υποσχόμενη κατεύθυνση για μελλοντική έρευνα.



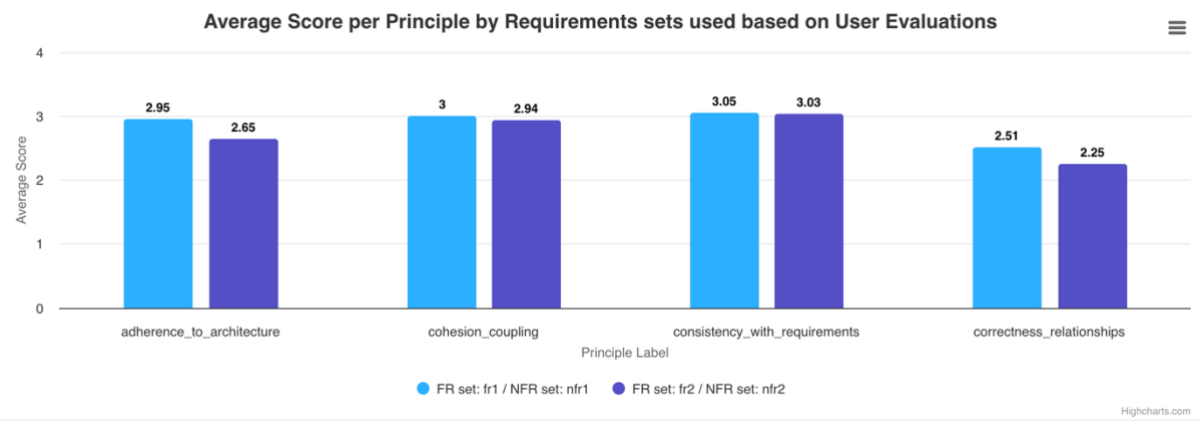
Εικόνα 4.31 Average Score by RAG file based on human evaluations

Ένας ακόμη κομβικός παράγοντας που επηρεάζει σημαντικά τις αποκρίσεις των LLMs είναι τα σύνολα **FR/NFR**. Η δομή τους, το επίπεδο λεπτομέρειας και η γλώσσα που χρησιμοποιείται στο prompt φαίνεται να παίζουν καθοριστικό ρόλο στην ποιότητα των παραγόμενων διαγραμμάτων κλάσεων. Στο επόμενο γράφημα, επιστρέφουμε με δεδομένα βαθμολογιών στην παρατήρηση που παρουσιάσαμε με διαγράμματα κλάσεων στην ενότητα 4.1 .

Το γράφημα στην Εικόνα 4.32 παρουσιάζει το μέσο όρο βαθμολογιών που πέτυχαν τα διαγράμματα κλάσεων, τα οποία δημιουργήθηκαν με τα δύο σύνολα **FR/NFR** που παρουσιάστηκαν στην ενότητα 3.1.2. Όπως φαίνεται, τα διαγράμματα που δημιουργήθηκαν χρησιμοποιώντας το πρώτο σύνολο υπερέχουν σταθερά σε σχέση με εκείνα που δημιουργήθηκαν με το δεύτερο σύνολο, σε όλα τα κριτήρια αξιολόγησης. Αξιοσημείωτο είναι ότι οι διαφορές στις βαθμολογίες είναι πιο έντονες σε ορισμένες κατηγορίες, ενώ σε άλλες είναι σχετικά μικρές.

Συγκεκριμένα, στις κατηγορίες «**Συνοχή και Σύζευξη**» και «**Συνοχή με τις Απαιτήσεις**», οι διαφορές στις βαθμολογίες είναι μικρές. Αντίθετα, το γράφημα αναδεικνύει σημαντικές διαφορές στις κατηγορίες «**Συμμόρφωση με την Αρχιτεκτονική**» και «**Ορθότητα των Σχέσεων Κλάσεων**», όπου τα διαγράμματα που δημιουργήθηκαν με το δεύτερο σύνολο εμφανίζουν αισθητά χαμηλότερες επιδόσεις.

Αν και αυτή η αρχική παρατήρηση υποδηλώνει μια σημαντική διαφορά απόδοσης μεταξύ των δύο συνόλων, οι υποκείμενοι λόγοι αυτής της συμπεριφοράς απαιτούν περαιτέρω διερεύνηση. Για κατανόηση σε μεγαλύτερο βάθος αυτής της συμπεριφορά, έχουμε δημιουργήσει ένα ακόμη διάγραμμα για αυτή την ανάλυση.



Εικόνα 4.32 Average Score by requirements set based on human evaluations

Το γράφημα στην Εικόνα 4.33 παρέχει μια βαθύτερη κατανόηση, αναλύοντας το μέσο όρο βαθμολογιών των διαγραμμάτων ανά τύπο αρχιτεκτονικής (**Client-Server**, **Model-View-Controller** και **Three-Tier**) και σύνολο **FR/NFR** που χρησιμοποιήθηκε. Αυτή η λεπτομερής απεικόνιση ρίχνει φως στα μοτίβα που παρατηρήθηκαν στην Εικόνα 4.32 .**Error! Reference source not found.**

- Αρχιτεκτονικές Client-Server και Model-View-Controller:
    - Στις αρχιτεκτονικές αυτές, τα διαγράμματα που δημιουργήθηκαν με το δεύτερο σύνολο παρουσιάζουν σημαντική πτώση στις βαθμολογίες σε όλα τα κριτήρια αξιολόγησης, ιδιαίτερα στις κατηγορίες **"Συμμόρφωση με την Αρχιτεκτονική"** και **"Ορθότητα των Σχέσεων Κλάσεων"**.
- Αρχιτεκτονική Three-Tier:

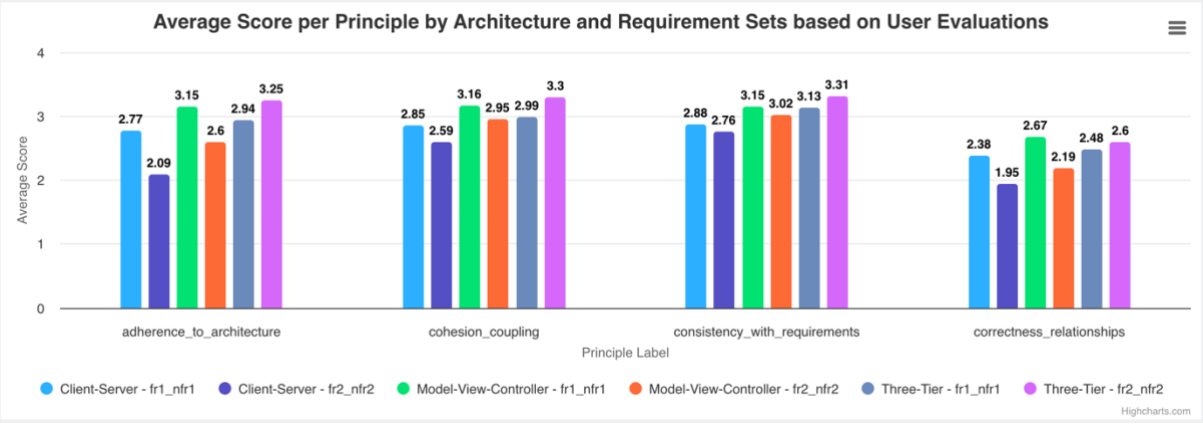
- Ενδιαφέρον παρουσιάζει το γεγονός ότι το δεύτερο σύνολο αποδίδει καλύτερα σε αυτή την αρχιτεκτονική, με υψηλότερες μέσες βαθμολογίες σε όλα τα κριτήρια.

Αυτή η αρχικά απρόσμενη συμπεριφορά μπορεί να αποδοθεί στη δομή του δεύτερου συνόλου **FR/NFR** (**FR2**). Όπως αναφέρθηκε νωρίτερα στο 3.1.2, το **FR2** διαχωρίζει τις απαιτήσεις σε τρία διακριτά τμήματα: **Αλληλεπιδράσεις Χρήστη, Λογική, και Λειτουργίες Δεδομένων**. Αυτή η τμηματοποίηση ευθυγραμμίζεται στενά με τις αρχές της αρχιτεκτονικής **Three-Tier**, όπου οι ευθύνες συνήθως χωρίζονται σε τρία επίπεδα: **Επίπεδο Παρουσίασης, Επίπεδο Λογικής, και Επίπεδο Πρόσβασης στα Δεδομένα**. Ως αποτέλεσμα, τα LLMs τείνουν ακολουθούν αυτόν τον διαχωρισμό στα παραγόμενα διαγράμματα, οδηγώντας σε υψηλότερες βαθμολογίες για την αρχιτεκτονική **Three-Tier**.

Αντίθετα, οι αρχιτεκτονικές Client-Server και MVC δεν ευθυγραμμίζονται εγγενώς με αυτή την τμηματοποιημένη προσέγγιση. Οι σχεδιαστικές τους αρχές βασίζονται σε διαφορετικούς διαχωρισμούς ευθυνών, και η δομή του FR2 φαίνεται να μειώνει την ικανότητα των LLMs να δημιουργούν διαγράμματα που να συμμορφώνονται με αυτά τα αρχιτεκτονικά πρότυπα. Αυτή η δυσαρμονία είναι ιδιαίτερα εμφανής σε κατηγορίες που σχετίζονται στενά με την αρχιτεκτονική, όπως η «**Συμμόρφωση με την Αρχιτεκτονική**» και η «**Ορθότητα των Σχέσεων Κλάσεων**», όπου οι βαθμολογίες καταγράφουν σημαντική επιδείνωση.

Αυτά τα ευρήματα υπογραμμίζουν τη σημασία της προσαρμογής των συνόλων FR/NFR στις αρχές της αιτούμενης αρχιτεκτονικής ή της διασφάλισης ότι είναι ουδέτερα και δεν προσανατολίζονται προς κάποιο συγκεκριμένο αρχιτεκτονικό πρότυπο. Η δομή και η διατύπωση των απαιτήσεων μπορούν να επηρεάσουν σημαντικά την ικανότητα των LLMs να δημιουργούν ακριβή και σχετικά διαγράμματα κλάσεων.

Για μια πιο ξεκάθαρη κατανόηση, ανατρέξτε στη δομή των δύο συνόλων FR/NFR που παρουσιάστηκαν στο 3.1.2, καθώς και στα συγκεκριμένα διαγράμματα που συζητήθηκαν στο 4.1, τα οποία απεικονίζουν αυτή τη συμπεριφορά.



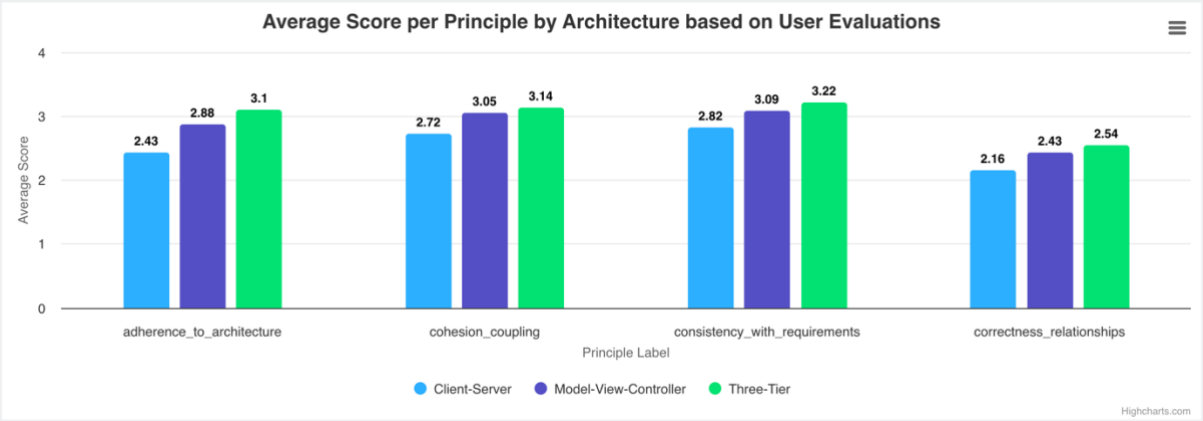
Εικόνα 4.33 Average Score by architecture and requirements set based on human evaluations

Τέλος, αναλύουμε κατά πόσο η ζητούμενη αρχιτεκτονική επηρεάζει τις μέσες βαθμολογίες των παραγόμενων διαγραμμάτων κλάσεων. Τα επόμενα γραφήματα παρουσιάζουν το μέσο όρο βαθμολογιών ομαδοποιημένες ανά αιτούμενη αρχιτεκτονική.

Το γράφημα στην Εικόνα 4.34 αποκαλύπτει ότι τα διαγράμματα με ζητούμενη αρχιτεκτονική Client-Server σημειώνουν τις χαμηλότερες μέσες βαθμολογίες. Οι άλλες δύο αρχιτεκτονικές, Three-Tier και Model-View-Controller, παρουσιάζουν μικρότερες διαφορές στις

βαθμολογίες τους, με τα διαγράμματα Three-Tier να υπερέχουν ελαφρώς σε σχέση με αυτά του MVC.

Ωστόσο, είναι σημαντικό να σημειωθεί ότι το συγκεκριμένο γράφημα ενσωματώνει τον επίδραση της επιλογής του συνόλου FR, και ιδιαίτερα την κακή ευθυγράμμιση του δεύτερου συνόλου FR2 με τα αρχιτεκτονικά πρότυπα Client-Server και MVC.

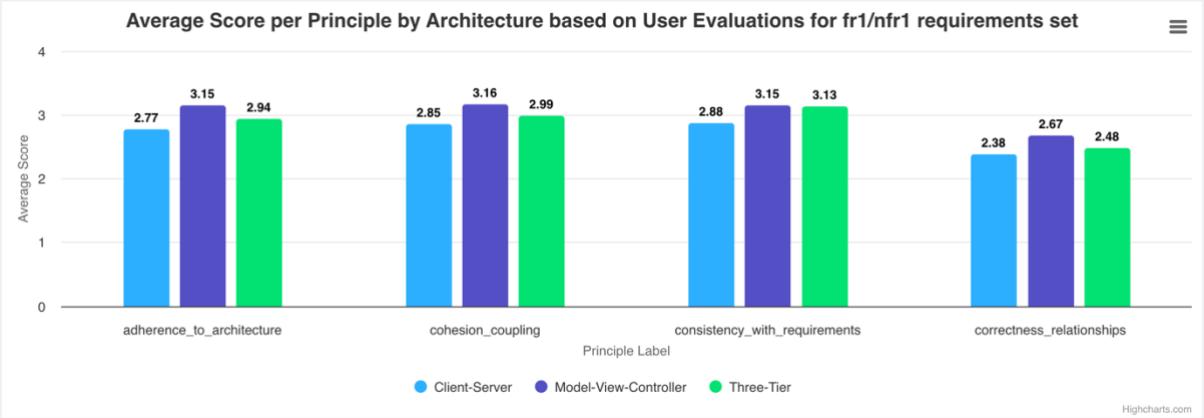


Εικόνα 4.34 Average Score by architecture based on human evaluations

Προκειμένου να απομονώσουμε την επίδραση της ζητούμενης αρχιτεκτονικής, το γράφημα στην Εικόνα 4.35 παρουσιάζει την ίδια ανάλυση, λαμβάνοντας υπόψη μόνο τα διαγράμματα που δημιουργήθηκαν χρησιμοποιώντας το σύνολο FR1, το οποίο, όπως παρατηρήθηκε κατά την εξερεύνηση των διαγραμμάτων κλάσεων, εμφανίζει πιο ισορροπημένη και συνεπή συμπεριφορά μεταξύ των αρχιτεκτονικών.

Όπως αναμενόταν, τα αποτελέσματα δείχνουν μειωμένες διαφορές στις βαθμολογίες. Ενώ η αρχιτεκτονική Client-Server παραμένει αυτή με τις χαμηλότερες βαθμολογίες, η σχετική απόδοση του MVC και του Three-Tier αλλάζει: τα διαγράμματα MVC τώρα ξεπερνούν σε απόδοση αυτά του Three-Tier.

Τα ευρήματα υποδεικνύουν ότι η αρχιτεκτονική Client-Server πιθανώς παρουσιάζει εγγενείς προκλήσεις για τα LLMs, καθιστώντας πιο δύσκολο για αυτά να ακολουθήσουν τις αρχές της σε σύγκριση με τις άλλες δύο αρχιτεκτονικές. Αντίθετα, η σχετική απόδοση του MVC και του Three-Tier φαίνεται να εξαρτάται σε μεγάλο βαθμό από το χρησιμοποιούμενο σύνολο FR. Αυτό υπογραμμίζει τη σημασία τόσο της ζητούμενης αρχιτεκτονικής όσο και του σχεδιασμού των συνοδευτικών συνόλων FR/NFR για τον καθορισμό της ποιότητας των παραγόμενων διαγραμμάτων κλάσεων.

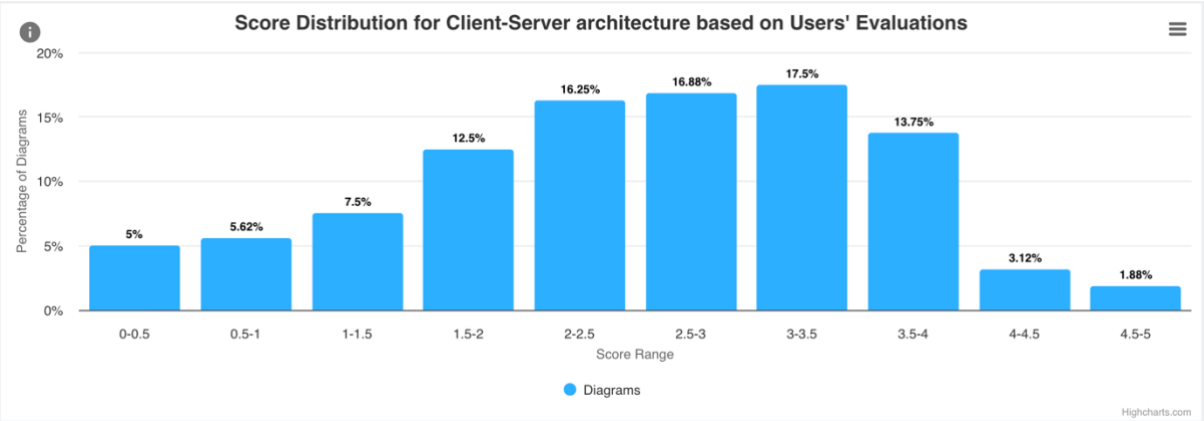


Εικόνα 4.35 Average Score by architecture for FR1 set based on human evaluations

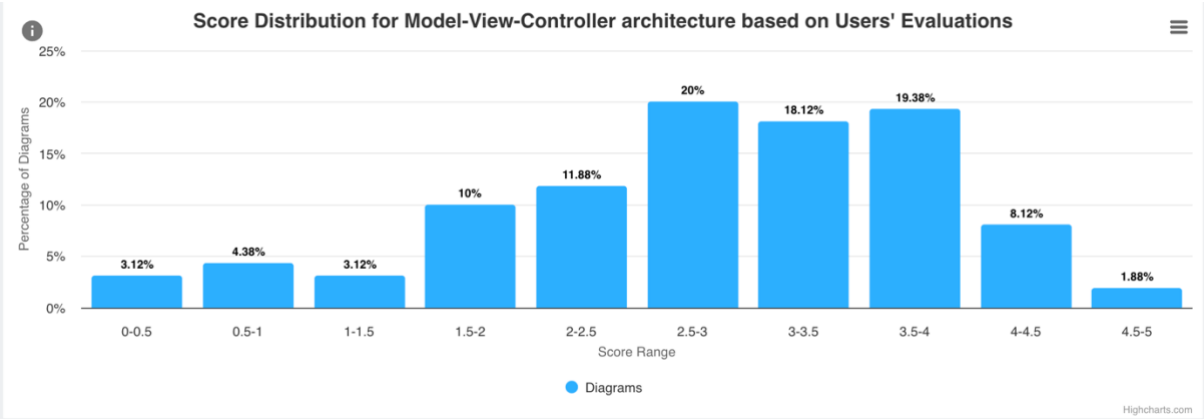
Παρόμοια συμπεράσματα προκύπτουν και από την ανάλυση της κατανομής των βαθμολογιών, η οποία εξετάζει τις αξιολογήσεις για κάθε ζητούμενη αρχιτεκτονική ξεχωριστά. Τα γραφήματα στην Εικόνα 4.36, στην Εικόνα 4.37 και στην Εικόνα 4.38 αποκαλύπτουν ότι τα διαγράμματα που δημιουργήθηκαν με την αρχιτεκτονική Client-Server παρουσιάζουν μεγαλύτερα ποσοστά χαμηλών μέσων βαθμολογιών σε όλες τις κατηγορίες κριτηρίων σε σύγκριση με τις άλλες δύο αρχιτεκτονικές. Παρατηρείται ότι καθώς μεταβαίνουμε από το Client-Server στο Model-View-Controller (MVC) και στη συνέχεια στο Three-Tier, η κορυφή των γραφημάτων κατανομής μετατοπίζεται προοδευτικά προς τα δεξιά. Επιπλέον, παρατηρείται ότι τα διαγράμματα με μέσες βαθμολογίες μεταξύ 3.5 και 5 αντιπροσωπεύουν:

- Λιγότερο από 20% για το **Client-Server**,
- Περίπου 30% για το **MVC**, και
- Σχεδόν 40% για το **Three-Tier**.

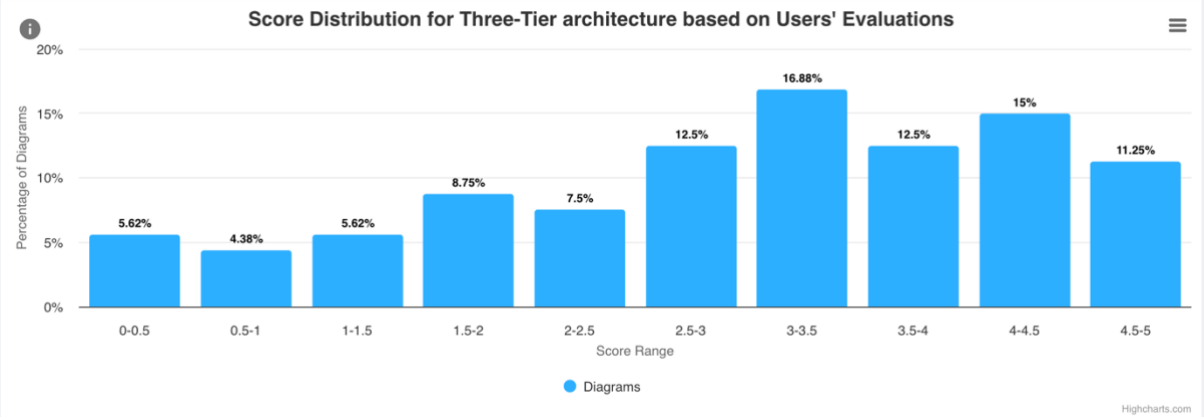
Αυτές οι κατανομές υπογραμμίζουν τη σχετική δυσκολία που αντιμετωπίζουν τα **LLMs** στο να ευθυγραμμιστούν με την αρχιτεκτονική **Client-Server**, σε αντίθεση με τη πιο σταθερή απόδοση που παρατηρείται στα πρότυπα **Three-Tier** και **MVC**.



Εικόνα 4.36 Score distribution for Client-Server architecture based on human evaluations



Εικόνα 4.37 Score distribution for MVC architecture based on human evaluations



Εικόνα 4.38 Score distribution for Three Tier architecture based on human evaluations

4.3.3 Σύνοψη αποτελεσμάτων

Η ανάλυση των αποτελεσμάτων αναδεικνύει σημαντικά ευρήματα σχετικά με τους παράγοντες που επηρεάζουν την ποιότητα των διαγραμμάτων κλάσεων που δημιουργούνται από LLMs. Αρχικά, είναι σημαντικό να επισημανθεί ότι οι αξιολογήσεις των χρηστών είναι υποκειμενικές. Για τον λόγο αυτό, η διεύρυνση του δείγματος των αξιολογητών θα μπορούσε να ενισχύσει την αξιοπιστία των ευρημάτων. Παρά την υποκειμενικότητα, τα αποτελέσματα του πειράματος παραμένουν ιδιαίτερα χρήσιμα, καθώς οι συγκριτικές αξιολογήσεις μεταξύ των παραγόντων ευθυγραμμίζονται με τις αντιλήψεις των χρηστών και όπως παρουσιάσαμε ερμηνεύονται λογικά. Τα αποτελέσματα καταγράφουν λοιπόν μια πρώτη τάση ως προς την επιρροή των διαφόρων παραγόντων. Ειδικότερα, οι τεχνικές RAG αποδείχθηκαν εξαιρετικά χρήσιμες, με τη σημασιολογική κατάτμηση (semantic chunking) να υπερέχει της αναδρομικής μεθόδου (recursive method) και τα embedding models να παρουσιάζουν ισορροπημένα και ανταγωνιστικά αποτελέσματα μεταξύ ανοιχτού κώδικα και εμπορικών επιλογών.

Η δομή των συνόλων FR/NFR είχε σημαντικό αντίκτυπο στην ποιότητα των διαγραμμάτων, με τα ουδέτερα ,ως προς την αρχιτεκτονική, σύνολα να αποδίδουν τα καλύτερα αποτελέσματα, ενώ σύνολα που πιθανώς εισάγουν biases όπως το FR2 δημιούργησαν προκλήσεις για αρχιτεκτονικές όπως το Client-Server και το MVC. Μεταξύ των ζητούμενων αρχιτεκτονικών, το Client-Server σημείωσε σταθερά χαμηλότερες βαθμολογίες, υποδεικνύοντας εγγενείς δυσκολίες των LLMs να ευθυγραμμιστούν με τις αρχές του. Αντίθετα, οι αρχιτεκτονικές Three-Tier και MVC παρουσίασαν καλύτερη απόδοση, για το MVC ιδιαίτερα όταν χρησιμοποιήθηκε το σύνολο FR1.

Επιπλέον, το περιεχόμενο των αρχείων RAG έδειξε ότι τα αρχιτεκτονικά στοχευμένα αρχεία απέδωσαν την υψηλότερη απόδοση, ενώ τα αρχεία RAG που αντλήθηκαν από διαδικτυακές πηγές ξεπέρασαν, σε ορισμένες περιπτώσεις, αυτά που βασίζονταν σε ακαδημαϊκά κείμενα.

Αυτά τα ευρήματα υπογραμμίζουν τη σημασία εισόδων με ακρίβεια, στοχευμένου περιεχομένου και βελτιστοποιημένων διαδικασιών για τη βελτίωση της απόδοσης των LLMs σε εργασίες αρχιτεκτονικής λογισμικού. Η μελλοντική έρευνα θα πρέπει να εστιάσει στη βελτιστοποίηση αυτών των παραμέτρων για περαιτέρω βελτίωση της ποιότητας των διαγραμμάτων και να εξετάσει την προσαρμοστικότητα των LLMs σε διαφορετικά αρχιτεκτονικά πρότυπα.

5 Σύνοψη

Η παρούσα εργασία επικεντρώθηκε στη διερεύνηση των δυνατοτήτων και περιορισμών των μεγάλων γλωσσικών μοντέλων στη δημιουργία αρχιτεκτονικών ως διαγράμματα κλάσεων UML που ακολουθούν συγκεκριμένα αρχιτεκτονικά πρότυπα, όπως Client-Server, Three-Tier και Model-View-Controller. Μέσα από συστηματική πειραματική διαδικασία που περιλάμβανε 480 σενάρια, αναλύθηκε η επίδραση διάφορων παραμέτρων, όπως η διατύπωση λειτουργικών και μη λειτουργικών απαιτήσεων, οι τεχνικές RAG, τα μοντέλα ενσωμάτωσης και οι μέθοδοι κατάτμησης. Επιπλέον, αναπτύχθηκε ένα διαδικτυακό εργαλείο παρουσίασης για την απεικόνιση και αξιολόγηση των αρχιτεκτονικών, επιτρέποντας την αξιολόγηση τόσο από ανθρώπινους εμπειρογνώμονες όσο και από τα ίδια τα γλωσσικά μοντέλα.

5.1 Ανά Αρχιτεκτονική

Τα αποτελέσματα της μελέτης αναδεικνύουν την πολλά υποσχόμενη δυναμική των LLMs στη δημιουργία αρχιτεκτονικών ως διαγράμματα κλάσεων UML. Ένα σημαντικό ποσοστό από τα παραγόμενα διαγράμματα παρουσίασε ισχυρή κατανόηση των αρχιτεκτονικών αρχών, αποτυπώνοντας με ακρίβεια τα ζητούμενα πρότυπα και ανταποκρινόμενο στα βασικά κριτήρια αξιολόγησης. Ωστόσο, η αρχιτεκτονική Client-Server προκάλεσε ιδιαίτερες δυσκολίες, καταλήγοντας συχνά σε χαμηλότερες βαθμολογίες συγκριτικά με τις αρχιτεκτονικές Three-Tier και MVC. Οι δύο τελευταίες παρουσίασαν μεγαλύτερη συμμόρφωση όταν συνοδεύονταν από καλά δομημένες εισόδους, υπογραμμίζοντας τη σημασία της ποιότητας των εισόδων και της ευθυγράμμισης με την αρχιτεκτονική. Το φαινόμενο αυτό μπορεί να αποδωθεί στον πλουραλισμό του υφιστάμενου υλικού, στο οποία έχουν εκπαιδευτεί τα LLMs. Οι αρχικές πρακτικές που αφορούσαν το μοντέλο του client-server είχαν να κάνουν με thick clients, οι οποίοι υλοποιούσαν το σύνολο της επιχειρησιακής λογικής και οι server είχαν μονάχα την ευθύνη της βάσης δεδομένων. Η εξέλιξη των πρακτικών αυτό έχει οδηγήσει σε διαφορετική κατανομή των ευθυνών μεταξύ client και server, με τους εξυπηρετητές πλέον να αναλαμβάνουν κομμάτι -αν όχι εξ' ολοκλήρου- της επιχειρησιακής λογικής.

Παρά την ύπαρξη υψηλής ποιότητας διαγραμμάτων, κάποια εξ' αυτών έλαβαν μέτριες βαθμολογίες, αντανακλώντας μία επαρκή κατανόηση των απαιτήσεων, αλλά με αδυναμίες σε τομείς όπως «η ορθότητα των σχέσεων κλάσεων» και «η συμμόρφωση με την αρχιτεκτονική». Από την άλλη πλευρά, διαγράμματα με χαμηλή απόδοση, που ήταν πραγματικά αποτυχημένα διαγράμματα, απέτυχαν να σεβαστούν τα αρχιτεκτονικά πρότυπα ή παρουσίασαν έλλειψη συνοχής, αναδεικνύοντας τις προκλήσεις που δημιουργούν οι ασυνεπείς είσοδοι, οι λιγότερο αποτελεσματικές τεχνικές και το συχνά διφορούμενο υλικό στο οποίο έχουν εκπαιδευτεί τα μοντέλα.

5.2 Ανά Μέθοδο RAG

Οι τεχνικές RAG και τα στοχευμένα αρχεία συμφραζομένων βελτίωσαν σημαντικά την απόδοση των LLMs. Η σημασιολογική κατάτμηση και τα αρχιτεκτονικά στοχευμένα

αρχεία RAG απέδωσαν τα καλύτερα αποτελέσματα, αποδεικνύοντας τη σημασία της προσαρμογής των μεθόδων ανάκτησης στις συγκεκριμένες απαιτήσεις.

Η μελέτη αποκάλυψε επίσης ότι αρχεία RAG από διαδικτυακές πηγές ξεπέρασαν σε ορισμένες περιπτώσεις τις ακαδημαϊκές πηγές, υποδεικνύοντας ότι τα ποικίλα, προσβάσιμα σύνολα δεδομένων μπορούν να προσφέρουν καλύτερα αποτελέσματα από παραδοσιακά περιεχόμενα. Αυτό το εύρημα τονίζει τις δυνατότητες αξιοποίησης μη συμβατικών αλλά αποτελεσματικών πηγών δεδομένων για τη βελτίωση των παραγόμενων αποτελεσμάτων των LLMs.

5.3 Ανά Μεγάλο Γλωσσικό Μοντέλο

Τα τοπικά μοντέλα ανοιχτού κώδικα, όπως το **Gemma2:27b**, παρουσίασαν ανταγωνιστική απόδοση συγκριτικά με τα κορυφαία διαδικτυακά μοντέλα, όπως το GPT-4o. Αυτό το εύρημα υπογραμμίζει τη δυναμική για ασφαλείς, οικονομικά αποδοτικές και ιδιωτικές λύσεις στον αυτοματισμό της τεχνολογίας λογισμικού.

Η ανάλυση αποκάλυψε επίσης σαφείς διαφορές στις επιδόσεις των μοντέλων, τονίζοντας τη σημασία της επιλογής κατάλληλων LLMs και της αποφυγής εκείνων που παρουσιάζουν σταθερά χαμηλή απόδοση. Αυτά τα ευρήματα αποτελούν τη βάση για μελλοντικές βελτιώσεις, εστιάζοντας στη βελτιστοποίηση των αποκρίσεων των LLMs και στην εφαρμογή δοκιμασμένων τεχνικών για την ενίσχυση της συνολικής ποιότητας των αποτελεσμάτων.

Συνολικά, τα αποτελέσματα παρέχουν πολύτιμες γνώσεις για το τι λειτουργεί και τι όχι, προσφέροντας σαφή κατεύθυνση για μελλοντικές βελτιώσεις. Τονίζουν τη σημασία της επιλογής κατάλληλων μοντέλων, της μείωσης των προκαταλήψεων στις προτροπές που ενδέχεται να επηρεάσουν τη συμπεριφορά των LLMs, της αποφυγής μοντέλων με σταθερά χαμηλή απόδοση και της αξιοποίησης δοκιμασμένων τεχνικών για τη βελτιστοποίηση της απόδοσης. Αυτή η ολοκληρωμένη ανάλυση όχι μόνο αναδεικνύει τις τρέχουσες δυνατότητες των LLMs, αλλά και δείχνει την κατεύθυνση για στοχευμένες εξελίξεις στην τεχνολογία λογισμικού με τη βοήθεια της τεχνητής νοημοσύνης.

5.4 Future Work

Τα αποτελέσματα αυτής της μελέτης όχι μόνο αναδεικνύουν τις τρέχουσες δυνατότητες των μεγάλων γλωσσικών μοντέλων στη δημιουργία αρχιτεκτονικών μέσω διαγραμμάτων κλάσεων UML, αλλά και ανοίγουν τον δρόμο για αρκετές υποσχόμενες κατευθύνσεις με στόχο τη βελτίωση και την ενίσχυση της αποτελεσματικότητάς τους.

Μία πιθανή περιοχή εξερεύνησης είναι η επέκταση της πολυπλοκότητας των εισόδων. Αντί να χρησιμοποιούνται προτροπές με λειτουργικές και μη λειτουργικές απαιτήσεις (FR/NFR) και την αιτούμενη αρχιτεκτονική, μελλοντικές εργασίες θα μπορούσαν να διερευνήσουν τη χρήση ενός ολοκληρωμένου εγγράφου **Software Requirements Specification (SRS)** ως είσοδο. Αυτή η ολοκληρωμένη προσέγγιση θα επέτρεπε στα LLMs να επεξεργάζονται πιο λεπτομερείς και ενοποιημένες αναπαραστάσεις των απαιτήσεων του συστήματος, οδηγώντας πιθανώς σε πιο ακριβή και συμφραζόμενα αποτελέσματα. Σε μια τέτοια κατεύθυνση τα διαγράμματα του SRS θα μπορούσαν να μεταβούν από στατικές εικόνες τύπου PNG σε δομημένες περιγραφές όπως κώδικα PlantUML, προσφέροντας μεγαλύτερη ευελιξία για τροποποιήσεις και αξιολογήσεις.

Ένας ακόμη παράγοντας που θα μπορούσε να εξεταστεί σε μεγαλύτερο βάθος σε μελλοντική εργασία είναι το σύνολο των αρχιτεκτονικών προτύπων. Βασιζόμενοι στην εμπειρία και τις γνώσεις που αποκτήθηκαν από θεμελιώδεις αρχιτεκτονικές, όπως Client-Server, Three-Tier και Model-View-Controller (MVC), θα μπορούσαμε να εξετάσουμε τη μετάβαση σε πιο σύνθετες αρχιτεκτονικές, όπως οι Microservices και η Service-Oriented Architecture.

Η εστίαση σε τοπικά μοντέλα, όπως το **Gemma2:27b**, που παρουσίασε ισχυρές επιδόσεις σε αυτή τη μελέτη, αποτελεί μια ακόμη κρίσιμη κατεύθυνση. Οι προσπάθειες θα μπορούσαν να επικεντρωθούν στη βελτίωση των τεχνικών **RAG**, προσαρμοσμένων ειδικά για αυτά τα μοντέλα, περιλαμβάνοντας τη βελτιστοποίηση στρατηγικών κατάτμησης και τη ρύθμιση των embedding models. Επιπλέον, η εξειδίκευση αυτών των τοπικών μοντέλων μέσω της μεθόδου **few-shot learning** θα μπορούσε να διερευνηθεί. Χρησιμοποιώντας τα κορυφαία διαγράμματα από το παρόν πείραμα ως μέρος του συνόλου δεδομένων για εξειδίκευση, μπορεί να καταστεί δυνατή η περαιτέρω ενίσχυση της κατανόησης των αρχιτεκτονικών προτύπων και της δημιουργίας διαγραμμάτων από το μοντέλο.

Αυτές οι προσπάθειες, σε συνδυασμό με μια βαθύτερη διερεύνηση της αλληλεπίδρασης μεταξύ αρχιτεκτονικών προτύπων, συνόλων FR/NFR και διαδικασιών RAG, θα συμβάλουν στη δημιουργία ενός πιο ανθεκτικού και αξιόπιστου πλαισίου για το σχεδιασμό αρχιτεκτονικών λογισμικού με τη βοήθεια τεχνητής νοημοσύνης. Αξιοποιώντας τις γνώσεις από αυτή τη μελέτη, οι μελλοντικές εργασίες μπορούν να συνεχίσουν να διευρύνουν τα όρια των δυνατοτήτων των LLMs σε αυτόν τον κρίσιμο τομέα.

6 Βιβλιογραφία

[1] Shaveta, “A review on machine learning,” *International Journal of Science and Research Archive*, vol. 9, no. 1, pp. 281–285, May 2023, doi: 10.30574/ijsra.2023.9.1.0410.

[2] R. Mu and X. Zeng, “A Review of Deep Learning Research,” *KSII Transactions on Internet and Information Systems*, vol. 13, no. 4, pp. 1738–1764, Apr. 2019, doi: 10.3837/tiis.2019.04.001.

[3] X. Wang, “The application of NLP in information retrieval,” *Applied and Computational Engineering*, vol. 42, no. 1, pp. 290–297, Feb. 2024, doi: 10.54254/2755-2721/42/20230795.

[4] H. Li, G. K. Rajbahadur, and C.-P. Bezemer, “Studying the Impact of TensorFlow and PyTorch Bindings on Machine Learning Software Quality,” *ACM Transactions on Software Engineering and Methodology*, Jul. 2024, doi: 10.1145/3678168.

[5] “ETHICAL CONSIDERATION IN AI,” *International Research Journal of Modernization in Engineering Technology and Science*, May 2024, doi: 10.56726/IRJMETS55881.

[6] H. W. Marar, “Advancements in software engineering using AI,” *Computer Software and Media Applications*, vol. 6, no. 1, p. 3906, Feb. 2024, doi: 10.24294/csma.v6i1.3906.

[7] C. M. Hicks, C. S. Lee, and K. L. Foster-Marks, “The New Developer Executive Summary The New Developer AI Skill Threat, Identity Change & Developer Thriving in the Transition to AI-Assisted Software Development.” [Online]. Available: https://www.pluralsight.com/product/flow/developer-success-lab/dsl-navigate-toolkit

[8] I. Ozkaya, “Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications,” May 01, 2023, *IEEE Computer Society*. doi: 10.1109/MS.2023.3248401.

[9] P. Tembhekar, M. Devan, and J. Jeyaraman, “Role of GenAI in Automated Code Generation within DevOps Practices: Explore how Generative AI,” *Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online)*, vol. 2, no. 2, pp. 500–512, Oct. 2023, doi: 10.60087/jklst.vol2.n2.p512.

[10] Z. Gao, “A review on statistical language and neural network based code completion,” *Applied and Computational Engineering*, vol. 22, no. 1, pp. 233–239, Oct. 2023, doi: 10.54254/2755-2721/22/20231222.

[11] M. Atemkeng, S. Hamlomo, B. Welman, N. Oyetunji, P. Ataei, and J. L. K. E. Fendji, “Ethics of Software Programming with Generative AI: Is Programming without Generative AI always radical?,” Aug. 2024, doi: https://doi.org/10.48550/arXiv.2408.10554.

[12] N. M. Dr. Naveenkumar Jayakumar, “Role of Machine Learning & Artificial Intelligence Techniques in Software Testing,” *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 6, pp. 2913–2921, Apr. 2021, doi: 10.17762/turcomat.v12i6.5800.

[13] M. A. Job, “Automating and Optimizing Software Testing using Artificial Intelligence Techniques,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 5, 2021, doi: 10.14569/IJACSA.2021.0120571.

[14] S. Gautam, A. Khunteta, and P. Sharma, “A Review on Software Testing Using Machine Learning Techniques,” *ECS Trans*, vol. 107, no. 1, pp. 3393–3406, Apr. 2022, doi: 10.1149/10701.3393ecst.

[15] Dusica Marijan; Arnaud Gotlieb, “Software Testing for Machine Learning,” 2022, doi: 10.48550/arxiv.2205.00210.

[16] L. Kharb, “Automated Testing in Machine Learning Systems,” *International Journal of Progressive Research in Engineering Management and Science*, Nov. 2023, doi: 10.58257/IJPREMS32282.

[17] J. Calle and C. Zapata, “QUARE: Towards a Question-Answering Model for Requirements Elicitation,” Jul. 29, 2022. doi: 10.21203/rs.3.rs-1872151/v1.

[18] C. Cheligeer, J. Huang, G. Wu, N. Bhuiyan, Y. Xu, and Y. Zeng, “Machine learning in requirements elicitation: a literature review,” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 36, p. e32, Oct. 2022, doi: 10.1017/S0890060422000166.

[19] Zarina Che Embi, Khaleduzzaman, and Ng Kok Why, “A Systematic Review on Natural Language Processing and Machine Learning Approaches to Improve Requirements Specification in Software Requirements Engineering,” *International Journal of Membrane Science and Technology*, vol. 10, no. 2, pp. 1563–1577, Sep. 2023, doi: 10.15379/ijmst.v10i2.1828.

[20] W. Alhoshan, R. Batista-Navarro, and L. Zhao, “Towards a Corpus of Requirements Documents Enriched with Semantic Frame Annotations,” in *2018 IEEE 26th International Requirements Engineering Conference (RE)*, IEEE, Aug. 2018, pp. 428–431. doi: 10.1109/RE.2018.00055.

[21] C. Liu, Z. Zhao, L. Zhang, and Z. Li, “Automated Conditional Statements Checking for Complete Natural Language Requirements Specification,” *Applied Sciences*, vol. 11, no. 17, p. 7892, Aug. 2021, doi: 10.3390/app11177892.

[22] Q. Lu, L. Zhu, J. Whittle, and J. B. Michael, “Software Engineering for Responsible AI,” *Computer (Long Beach Calif)*, vol. 56, no. 4, pp. 13–16, Apr. 2023, doi: 10.1109/MC.2023.3242055.

[23] R. Khankhoje, “Quality Challenges and Imperatives in Smart AI Software,” in *Soft Computing, Artificial Intelligence and Applications*, Academy & Industry Research Collaboration Center, Dec. 2023, pp. 143–154. doi: 10.5121/csit.2023.132412.

[24] A. Barredo Arrieta *et al.*, “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI,” *Information Fusion*, vol. 58, pp. 82–115, Jun. 2020, doi: 10.1016/j.inffus.2019.12.012.

[25] E. A. Abdelnabi, A. M. Maatuk, and M. Hagal, “Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques,” in *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA*, IEEE, May 2021, pp. 288–293. doi: 10.1109/MI-STA52233.2021.9464433.

[26] O. MacMillan-Scott and M. Musolesi, “(Ir)rationality and cognitive biases in large language models,” *R Soc Open Sci*, vol. 11, no. 6, Jun. 2024, doi: 10.1098/rsos.240255.

[27] R. Dhar, K. Vaidhyanathan, and V. Varma, “Can LLMs Generate Architectural Design Decisions? -An Exploratory Empirical study,” Mar. 2024, [Online]. Available: http://arxiv.org/abs/2403.01709

[28] T. Eisenreich, S. Speth, and S. Wagner, “From Requirements to Architecture: An AI-Based Journey to Semi-Automatically Generate Software Architectures,” Jan. 2024, doi: 10.1145/3643660.3643942.

[29] S. Yang and H. Sahraoui, “Towards automatically extracting UML class diagrams from natural language specifications,” in *Proceedings - ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems, MODELS 2022: Companion Proceedings*, Association for Computing Machinery, Inc, Oct. 2022, pp. 396–403. doi: 10.1145/3550356.3561592.

[30] I. Altawaiha and A. Al-Hgaish, “ClassDiagGen Tool: Fine-Tuning the GPT-3 Model for Automated Class Diagram Generation from Textual Descriptions,” May 02, 2024. doi: 10.21203/rs.3.rs-4350615/v1.

[31] Ian. Sommerville, *Software engineering*. Pearson, 2011.

[32] P. Kumar and Y. Singh, “A Software Reliability Growth Model for Three-Tier Client Server System.”

[33] M. Yener and A. Theedom, “Model View Controller Pattern,” 2015. [Online]. Available: www.wrox.com/go/

[34] Eric Karlsson, “The evolution and erosion of a service-oriented architecture in enterprise software”.

[35] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, “From monolithic systems to Microservices: An assessment framework,” *Inf Softw Technol*, vol. 137, p. 106600, Sep. 2021, doi: 10.1016/j.infsof.2021.106600.

7 Ορολογία

Απόδοση	Ξενόγλωσσος όρος
Τεχνητή νοημοσύνη	Artificial Intelligence (AI)
Τεχνολογία λογισμικού	Software engineering
Παραγωγικό AI	Generative AI
Βαθιά Μάθηση	Deep learning
Μέθοδος Επαυξημένης Ανάκτησης	Retrieval Augmented Generation (RAG)
Ευρετικοί κανόνες	Heuristic rules
Μοντέλο πελάτη-εξυπηρετητή	Client-Server model
Αρχιτεκτονική στρωμάτων	Layered architecture
Μεγάλα γλωσσικά μοντέλα	Large Language models (LLMs)
Αρχή μοναδικής ευθύνης	Single responsibility principle
Αντίστροφη μηχανική	Reverse engineering
Μηχανισμός προσοχής	Attention mechanism
Σημασιολογική κατάτμηση	Semantic chunking
Αναδρομική κατάτμηση	Recursive chunking
Μοντέλα ανοιχτού κώδικα	Open-source models

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF COMPUTER SCIENCE

Exploring the utilization of LLM tools in Software Architecture

DIPLOMA THESIS

OF

MICHAIL TSILIMIGKOUNAKIS

Supervisor : Vassilios Vescoukis, Professor NTUA

Athens, November 2024

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF COMPUTER SCIENCE

Exploring the utilization of LLM tools in Software Architecture

DIPLOMA THESIS

OF

MICHAIL TSILIMIGKOUNAKIS

Supervisor : Vassilios Vescoukis, Professor

Approved by the examination committee in November 2024 .

V.Vescoukis	N.Papaspyrou	G.Stamou
Professor NTUA	Professor NTUA	Professor NTUA

Athens, November 2024

(Signature)

…….
Tsilimigkounakis Michail

Acknowledgements

As I complete my diploma thesis, my five-year journey at the School of Electrical and Computer Engineering at the National Technical University of Athens (NTUA) also comes to an end. I am deeply grateful to my friends and family for their constant support and encouragement throughout my undergraduate studies.

I would like to sincerely thank my professor, Mr. Vassilis Vescoukis, for the opportunity to work together, as well as for his guidance, time, feedback, and excellent cooperation during my diploma thesis.

I am also especially grateful to PhD researcher Mr. Christos Hadjichristofi for his assistance and continuous availability to help me throughout this process. Their time and support are truly appreciated.

Finally, I extend my thanks to everyone who was part of this journey at NTUA and contributed to this experience.

Abstract

The support across all phases of the software life cycle by large language models (LLMs) has gained momentum with the introduction of powerful LLMs during the past couple of years. Apart from code generation, it is worth exploring if using LLMs in early phases of software development, such as requirements engineering and architecture, can accelerate development and improve the quality of software systems. In this regard, AI-support in software architecture could generate high-level or even detailed architectural descriptions, such as UML class or component diagrams, from textual requirements. This thesis explores the capabilities and limitations of selected commercial and open source LLMs in generating class diagrams that comply with specific architectural patterns, namely Client-Server, Three-Tier, and Model-View-Controller (MVC). Through a systematic planning and evaluation of 480 scenarios, we assess how factors such as the description of functional and non-functional requirements, RAG (Retrieval-Augmented Generation) materials and processes, embedding models, and the choice of specific LLMs, ranging from smaller quantized models to larger, more advanced ones, impact the quality of LLM-generated UML class diagrams that describe software architectures for a small application.

Our study also introduces a custom tool for managing, evaluating, and analyzing the generated class diagrams, facilitating both human expert and AI-based assessments. The results of this work reveal a broad range of outcomes, from well-structured diagrams aligned with software requirements and the targeted architectural principles to incomplete or erroneous outputs influenced by model limitations, RAG challenges and other factors relating to the inherent wide spectrum of interpretations of the materials that have been used in the training of LLMs. This research is a first step in the investigation of the potential of LLMs for creative work in the software life cycle, and specifically in architectural design. We also investigate how locally-run LLMs can be guided to produce architectures that apply the principles required by each development context, without exposing any information to publicly available commercial services, to support secure and efficient automation in software development.

Table of Contents

Table of figures

1 Introduction

Software development is a complex, multistage process that ranged from the elicitation of user requirements to the development of fully functional systems; the process undergoes various phases, including discovery and analysis of requirements, architecture, design, programming, testing, deployment and maintenance. One of the most critical steps is the architecture and design phase, where the definition of high-level structures that satisfy the functional and non-functional requirements of a given system, is done, usually by creating Unified Modeling Language (UML) diagrams, such as class or component diagrams, that guide the next steps of the software life cycle. These diagrams provide a static structural representation of a software system by defining its "building blocks" such as classes or components, containing state (attributes) and behavior (methods), as well as associations among them. They serve as a blueprint that guides the rest of the development process, helping developers and stakeholders better understand the architecture and interactions that need to be implemented within the system. However, creating such diagrams remains a labor-intensive and error-prone task, often subject to human misinterpretation of architectural styles and principles. This introduces inefficiencies and other fallacies that can ripple through the development lifecycle.

With the advent of **artificial intelligence (AI)**, particularly through the development of large language models (LLMs) based on deep learning, new opportunities for automating software engineering tasks have emerged. These AI models, trained on massive datasets, have demonstrated remarkable capabilities in natural language processing and generation. This has paved the way for their application in tasks like code generation and debugging, and, possibly in the automation of design tasks such as architecture definition and generation of the corresponding UML diagrams. The potential to leverage AI to generate accurate and well-structured architectures as UML diagrams from textual software requirements is a challenging approach to one of the most important phases of software development.

The primary objective of this study is to explore the feasibility and effectiveness of using LLMs to generate software architectures, particularly in the context of specific architectural patterns such as Client-Server or Three-Tier. Our investigation focuses on understanding how different parameters—such as structure of functional and non-functional requirement sets, model selection, and the application of Retrieval-Augmented Generation (RAG)—impact the quality, completeness and accuracy of the generated diagrams, meaning the satisfaction of all functional and non-functional requirements, and the application of the requested architectural principles. Additionally, we aim to evaluate the **suitability of local LLMs**, ranging from smaller, resource-efficient models to larger, more advanced ones, as viable alternatives to commercial LLM tools like ChatGPT and Gemini, for specific activities within the scope of this work. This exploration of local LLM tools is motivated by the need of businesses, including software developers, to safeguard sensitive data, which often necessitates avoiding the use of online LLMs.

By assessing the performance of local models in terms of architectural adherence, class relationships, cohesion, coupling, and alignment with software requirements, we aim to provide valuable insights regarding their potential as secure and effective tools for software architecture and design automation. Through a combination of human expert evaluation and AI-based assessments,

this study seeks to explore the capabilities, limitations, and scalability of both local and online LLMs in AI-assisted software architectural design.

In the following sections, we will detail our methodology and experimental setup, introduce the custom tool we developed for managing and evaluating class diagrams, present our results and analysis, and conclude with a discussion of implications and future directions for research in AI-assisted software architecture design.

1.1 The evolution of modern AI

Artificial Intelligence (AI) is revolutionizing numerous fields by offering innovative ways to tackle challenges and discover solutions. At its core, AI involves creating systems capable of tasks traditionally requiring human intelligence, such as reasoning, learning, decision-making, and language understanding. By leveraging advanced techniques, AI enables machines to process vast amounts of data, adapt to new conditions, and improve over time. This allows AI to approach existing well-defined problems and unlock new opportunities across diverse domains.

Modern artificial intelligence (AI) is grounded in advanced computational models, including machine learning and deep learning. Machine learning enables systems to identify patterns and make predictions by analyzing vast amounts of data [1] , while deep learning, a subset of machine learning, uses neural networks to tackle more complex problems such as image recognition and speech processing [2]. These methods have revolutionized how machines understand and process high-dimensional information, forming the backbone of AI applications like natural language processing (NLP). NLP focuses on enabling machines to understand, generate, and interact with human language, powering technologies such as virtual assistants, language translation tools, and sentiment analysis systems [3].

AI's applications span diverse fields, transforming industries and daily life. In healthcare, AI-driven diagnostic tools improve accuracy and speed, while in transportation, self-driving vehicles leverage real-time data and predictive algorithms. Modern AI tools, including TensorFlow, PyTorch, and OpenAI's models like ChatGPT, empower researchers and developers to design complex systems with ease [4]. These platforms facilitate rapid prototyping, scalability, and integration into existing technologies. Despite its successes, modern AI also raises important ethical questions about bias, privacy, and decision-making transparency, emphasizing the need for responsible development and regulation [5].

1.2 AI in Software Engineering

AI tools' ability to imitate creative work makes AI an attractive tool in software engineering [6]. In all cases, by automating tasks like code generation, refactoring, and debugging, AI could improve productivity and support the development of software of higher quality by executing code generation consistently; also, it can identify, and address coding errors and the general feeling is that many coding tasks can be partially or even entirely executed by LLMs. Beyond supporting development, AI offers the potential to streamline workflows by analyzing team operations and optimizing processes. However, integrating such capabilities in the software lifecycle is not straightforward; it requires careful consideration of system requirements, development contexts, and the limitations of current AI models [6].

While AI's potential extends across the entire software development lifecycle (SDLC), realizing this impact involves navigating significant challenges. For example, using LLMs for tasks like generating architectural models, automating testing, or predicting bottlenecks needs to be done with attention to accuracy, completeness, and correct application of the desired design principles. These offerings are far from "ready out of the box" and require thoughtful planning, evaluation, and customization to suit specific project needs. Factors like the quality of training data, domain-specific constraints, and ethical considerations must also be considered to ensure meaningful and dependable outcomes.

Ultimately, AI's role in software engineering amplifies human expertise but it is yet impossible to substitute it. By fostering systems that learn and adapt, AI drives innovation and enables teams to address the growing complexity of modern applications. However, the integration of AI into the SDLC is not just about adopting new tools—it's about understanding the trade-offs, limitations, and responsibilities associated with leveraging AI effectively [7], [8].

The following literature review synthesizes recent studies that explore the applications, benefits and limitations of AI assistance in software engineering grouped by field.

1.2.1 Code Generation and Assistance

Generative AI, particularly using machine learning (ML) and deep learning (DL) algorithms, has been pivotal in automating various aspects of the code generation process. Tembhekar emphasizes that these technologies enable the automation of tasks traditionally performed by developers, enhancing efficiency within DevOps practices [9]. The integration of natural language processing (NLP) techniques has further facilitated the development of AI systems capable of "understanding" and generating code, thereby streamlining workflows.

Code completion has become an essential area of research, with neural language models playing a significant role in enhancing developer productivity. Gao reviews the evolution of statistical language and neural network-based code completion systems, highlighting their effectiveness in improving coding efficiency [10]. However, the ethical implications of AI-driven code generation remain a critical area of concern. Atemkeng et al. [11] underscore the importance of responsible usage of generative AI tools, pointing to the need for safeguards to ensure that AI produces not only functional but also secure code. These discussions underline the necessity for continued regulation and oversight of AI technologies in software development.

In conclusion, the integration of AI in code generation represents a significant advancement in software engineering, driven by deep learning and natural language processing techniques. The ongoing research in this domain continues to explore the balance between automation and the ethical considerations of AI use, ensuring that the benefits of these technologies are harnessed responsibly.

1.2.2 Automated Testing

Automated testing powered by AI techniques has shown significant promise in improving software quality and reducing testing time. Mulla emphasizes that AI-driven automated testing allows for comprehensive test suites to be executed promptly with each software change, thereby facilitating continuous integration and delivery [12]. This is particularly crucial in modern software development environments where rapid iterations are common. Moreover, Job highlights that the

complexity of software systems necessitates the adoption of automation in testing to meet quality standards within shorter evaluation periods [13].

The application of machine learning (ML) techniques in automated testing has been extensively documented. For instance, Gautam et al. provide a comprehensive review of how ML algorithms can automate error detection processes, thereby enhancing the reliability of software systems [14].

Despite the advantages, the implementation of AI in automated testing is not without challenges. Marijan identifies key challenges in testing machine learning systems, including the need for specialized testing methodologies that address the unique characteristics of AI models [15]. The paper outlines a research agenda aimed at advancing the state of testing for machine learning applications, emphasizing the importance of developing robust testing frameworks. Similarly, the work by Latika Kharb discusses the implications of automated testing in machine learning systems, stressing the necessity for trust and transparency in AI applications [16].

In conclusion, the literature indicates that AI has the potential to revolutionize automated testing in software engineering by enhancing efficiency, accuracy, and reliability. However, the challenges associated with testing AI systems necessitate further research and development to establish robust methodologies that can address the unique demands of this field.

1.2.3 Natural Language Processing for Requirements and Design

Natural Language Processing (NLP) has emerged as a pivotal technology in the area of software requirements engineering and design, significantly enhancing the processes of requirements specification, elicitation, and analysis.

A systematic literature review conducted by Calle and Zapata introduces the QUARE (Question Answering for Requirements Elicitation), a question-answering model for improving the RE process expanding the concept of NLP4RE (Natural Language Processing for Requirements Engineering) [17]. This model is intended to support software analysts in the RE process by answering RE-related questions, extracting the answers from requirements documents regardless the requirements writing style. Moreover, the integration of ML techniques into requirements elicitation processes has been shown to automate and streamline the traditionally cumbersome tasks associated with requirements handling, thereby enhancing efficiency and effectiveness [18], [19].

Furthermore, the role of NLP in transforming natural language requirements into formal representations is critical. Semantic parsing techniques have been employed to add semantic structure to requirements, facilitating a clearer understanding of the intended functionalities [20]. This transformation is essential for ensuring that requirements are not only understandable but also actionable, as they can be directly utilized in the software development lifecycle.

The challenges associated with NLP in requirements engineering are also noteworthy. Despite the advancements, a significant portion of software requirements specifications remains in natural language, which can lead to misunderstandings and misinterpretations. The literature indicates that approximately 95% of software requirements are described in natural language, which highlights the ongoing need for effective NLP tools to automate the analysis and validation of these specifications [21]. Thus, ongoing research is necessary to refine NLP techniques to improve the quality and reliability of requirements documentation.

In conclusion, the integration of NLP and ML into software requirements engineering presents a promising avenue for enhancing the clarity, accuracy, and efficiency of requirements specification and design processes. While significant progress has been made, challenges remain in the effective application of these technologies, necessitating further research and development to fully realize their potential in the field.

In summary, the literature indicates that AI and machine learning are poised to revolutionize software engineering by enhancing efficiency, improving quality, and automating processes. However, successful integration necessitates a careful consideration of the interplay between human expertise and algorithmic capabilities, as well as an adaptation of existing practices to fully leverage the potential of these technologies.

1.3 Challenges of AI in Software Engineering

The integration of Artificial Intelligence (AI) into software engineering presents a variety of technical challenges that must be addressed to harness its full potential. One significant challenge is the complexity of verifying and validating AI-based systems. Traditional software verification techniques often fall short in the context of AI due to the intricate nature of AI algorithms and their reliance on large datasets, which can introduce biases and uncertainties [22]. Another critical challenge is the quality assurance of AI-driven software. The integration of AI technologies raises concerns regarding data quality, algorithm transparency, and ethical implications. Quality assurance professionals face difficulties in testing AI systems, as traditional testing methodologies may not adequately address the unique characteristics of AI, such as learning from data and evolving over time [23]. Furthermore, the potential for biased outcomes due to skewed training data necessitates rigorous scrutiny of the data used in AI applications [23]. The need for explainability in AI systems is also paramount, as stakeholders require insights into how decisions are made by AI algorithms to ensure accountability and trust [24].

In software architecture —a critical phase of the software development lifecycle— the potential for AI to automate various aspects remains a subject of debate. From generating architectural models to identifying potential design flaws early in development, AI could play a transformative role.

However, several factors make this process challenging for LLMs:

1. **Ambiguity and Complexity in Natural Language.**
   Requirements are usually written in natural language, which can be ambiguous, inconsistent, incomplete and lack the precision needed to support architectural decisions like class generation and design. So, most of the tools require consistent user intervention and interaction in the process of UML class diagrams generation. Even with the substantial enhancements that have been made recently, it seems that we cannot say that solutions could generate all the UML elements and data semantics automatically, i.e., class names, operations, and relationships, i.e., associations, and other advanced relationship types such as generalization, aggregation, and dependency [25].
2. **Difficulty in Inferring Relationships and Hierarchical Structures**.
   One of the core challenges in class diagram generation is the accurate representation of relationships such as inheritance, association, aggregation, and composition. Requirements documents rarely provide explicit details about these relationships; instead, they imply

relationships that need to be inferred, requiring significant abstraction and hierarchical reasoning in software development, which LLMs are not inherently designed for [26].

3. **Lack of Specialized Training on Formal Software Models**.
   Most LLMs are trained on large corpora of general text (e.g., Wikipedia, code repositories), which contain relatively little data specifically about software design patterns or UML structures. This leads to limitations when trying to generate diagrams that require adherence to the conventions of formal software modeling. While LLMs are trained on code and text, the distinct syntactic and semantic rules of class diagrams often remain challenging for these models to learn due to limited representation in their training datasets.
4. **The Need for Contextual Domain Knowledge and Design Patterns**.
   Translating textual descriptions into architectures often requires an understanding of specific domain knowledge, design patterns and generally accepted best practices. LLMs are typically not equipped to apply such contextual knowledge, especially in domain-specific scenarios where established patterns (e.g., MVC, Singleton) may play a crucial role in the architectural layout. Without this context, models tend to generate overly simplistic or incorrect representations. Additionally, the knowledge that exists about training LLMs often includes ambiguities, interpretive pluralism, and conflicts regarding what constitutes a pattern or how to apply it effectively. Vaidhyanathan et al. explore whether LLMs can effectively generate architectural design decisions in a zero-shot setting. Their findings suggest that while LLMs can assist in generating design decisions, they are not capable of doing so autonomously. Instead, these models are better suited to supporting architects in documenting and refining design decisions [27].

Motivated by these challenges, we aim to explore the evolving domain of AI-assisted software architecture. Our approach involves conducting a well-designed and comprehensive experiment. In the following sections we will analyze our methodology and the specifics of our experiment. We will also present our evaluation results, showcasing the performance of large language models (LLMs) in providing assistance in software architecture, and discuss the potential future challenges that emerge from our findings.

2 AI-assisted software architecture

This chapter explores the related work in the rapidly evolving field of AI-assisted software architecture, presents documentation for key architectural patterns, and outlines our approach's unique distinction and motivation in relation to the existing literature.

2.1 Related Work – AI in software architecture

The automatic generation of Unified Modeling Language (UML) diagrams from textual requirements has recently gained momentum, driven by the advent of LLMs. This emerging topic seeks to explore the potential of advanced AI systems in addressing long-standing challenges in software architecture, such as bridging the gap between natural language requirements and formal system design artifacts.

Traditional approaches to UML diagram generation often relied on natural language processing (NLP) techniques combined with heuristic rules or ontologies. While these methods demonstrated the feasibility of automation, they were limited by their reliance on structured input formats and the need for manual intervention. For example, earlier tools struggled with ambiguities inherent in natural language requirements and often produced incomplete diagrams.

Eisenreich, Speth, and Wagner propose a structured six-step process for bridging textual requirements and software architecture generation[28] . The process begins with the automatic creation of a domain model and use-case scenarios based on textual requirements, followed by manual refinement to enhance precision and relevance. Using the domain model, scenarios, and non-functional requirements, multiple architecture candidates and their corresponding architectural decisions are automatically derived. These candidates are then subjected to automatic evaluation and comparison. Afterward, manual refinements are made to selected candidates, culminating in the manual selection of the best-fitting architecture for implementation. In their exploratory analysis, the authors employed advanced large language models (LLMs) such as LLaMA2 70-B and GPT-3.5 to generate domain models from textual requirements. By providing the models with requirement sets and prompting them to generate PlantUML domain models, the authors observed that while both LLMs successfully identified key concepts, they struggled with task-specific alignment, often misunderstanding the intended prompts. Instead of modeling the domain as expected, the LLMs focused on modeling the system itself, highlighting a misalignment between the prompt and the generated output.

Another attempt to address the ambiguity challenges of natural language was introduced by Yang and Sahraoui through a AI-driven approach to UML class diagram generation [29]. Their method employs a binary classifier, utilizing machine learning to label sentences as describing either a class or a relationship. The process involves fragmenting English input into individual sentences, generating UML class diagram fragments from these, and then combining the fragments into a cohesive final diagram. To support this approach, the researchers created a dataset of UML diagrams paired with corresponding English specifications, mapping textual descriptions to UML fragments. This dataset, generated through a crowdsourcing initiative, while relatively small, was sufficient to

train the classifier and evaluate their methodology. Despite the innovative nature of their approach, the correctness of the generated diagrams was limited, which the authors attributed to the imprecision of the NLP tools employed. They noted that leveraging more advanced NLP tools could significantly enhance the accuracy and robustness of the results, underscoring the potential for future refinement and development in this area.

Another approach that tries to leverage GPT-3.5's natural language processing (NLP) capabilities for automating the extraction of UML class diagram elements was introduced by Iyad and Areen through the ClassDiagGentool [30]. The ClassDiagGentool represents a significant step forward, enhancing GPT-3.5's capabilities, through few-shot learning, to automate the generation of UML class diagrams. By fine-tuning GPT-3 on a dataset of 50 diverse case studies (pairs prompt – ideal generated text), ClassDiagGen demonstrated exceptional precision (98.6%) and recall (93.3%), significantly outperforming earlier methods. Its pipeline integrates textual analysis, PlantUML code generation, and diagram rendering into a cohesive workflow, showcasing the practical potential of LLMs in streamlining software design processes.

2.2 Architectural Patterns

This section provides a formal description of key architectural patterns, outlining their structure, components, and application in software design. It aims to present these patterns clearly and accurately, serving as a reference for understanding their role and implementation in architectural design.

2.2.1 Client-Server Architecture

The client-server architecture is a widely-used conceptual model in computer networking, where multiple clients (remote processors) interact with a central server to request and receive services. This structure is designed to separate the roles of service providers and service consumers in a networked environment.

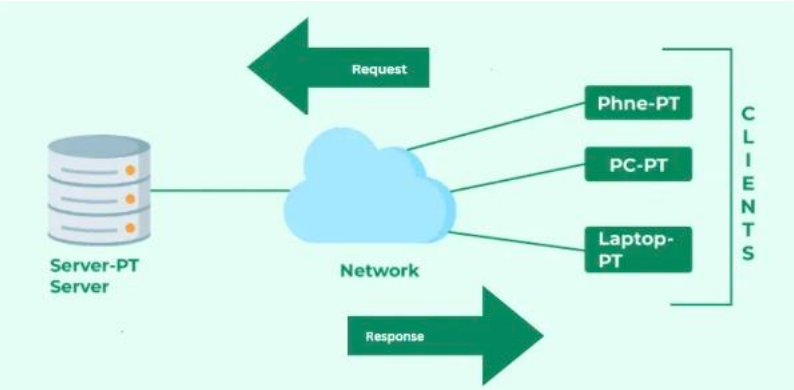


Figure 2.1 Client-Server model (source)

In a client-server model, the system is structured around various services hosted on dedicated servers, with clients accessing and utilizing these services [31]. The main components of this architecture are:

- **Servers** that provide services to other parts of the system. For instance, print servers handle printing requests, file servers manage file storage and retrieval, and compile servers perform programming language compilation tasks.
- **Clients** that request services from the servers. Typically, multiple instances of a client application are running simultaneously across different devices.
- **Network** infrastructure enabling clients to connect to these services. Most client-server architectures are set up as distributed systems, using Internet protocols to facilitate this connectivity.

2.2.2 Layered Architecture

In a layered architecture, the system's functionality is divided into distinct layers, with each layer depending only on the services provided by the layer directly below it. This structure supports step-by-step system development, allowing services within each layer to become accessible to users as they are completed.

This design is adaptable and portable; as long as the interface remains the same, any layer can be swapped out for another with similar functionality. Additionally, changes within a layer or new features affect only the directly neighboring layer, preserving stability throughout the rest of the system. By confining machine-specific dependencies to the innermost layers, this architecture facilitates multi-platform compatibility, as only the innermost layers need adjustments to work with a different operating system or database [31].

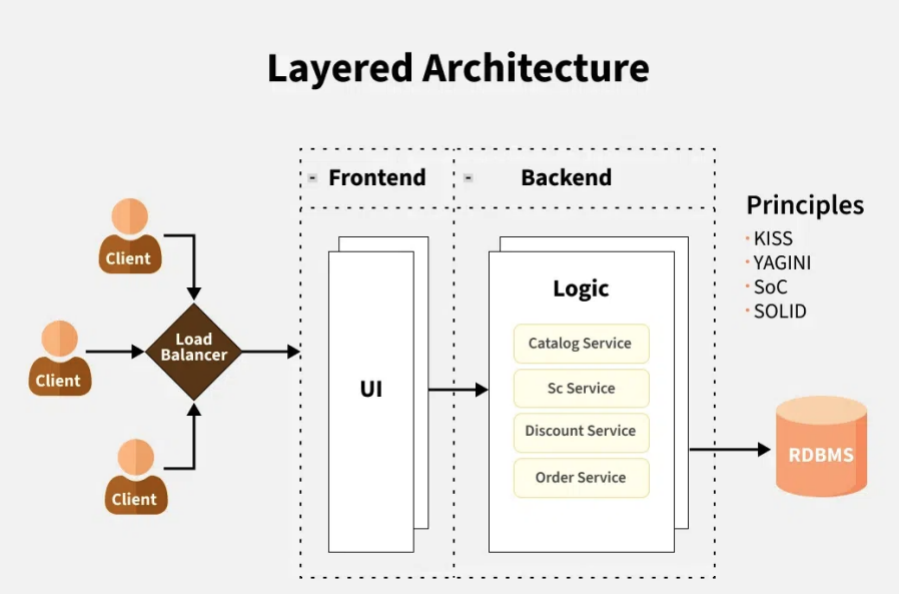


Figure 2.2 Layered Architecture model (source)

In summary, the layered architectural pattern organizes the system into a hierarchy of layers, each with related functionality. Each layer provides services to the one above it, with the lowest layers offering essential core services that support the entire system.

2.2.3 Three-Layered (Three–Tier) Architecture

This model consists of three layers: the presentation layer, business logic layer, and data layer, with the data stored at the backend.

- The **Presentation Layer** includes forms or server pages that present the application's user interface. This layer displays data gathers user input, and forwards requests to the next layer.
- The **Business Layer** handles requests from the presentation layer, applying business rules to the data. It processes these requests, passes them to the data layer, and incorporates the application's business logic.
- The **Data Layer** contains data access logic, database drivers, and query engines, enabling direct interaction with the database to retrieve or store data as needed. [32]

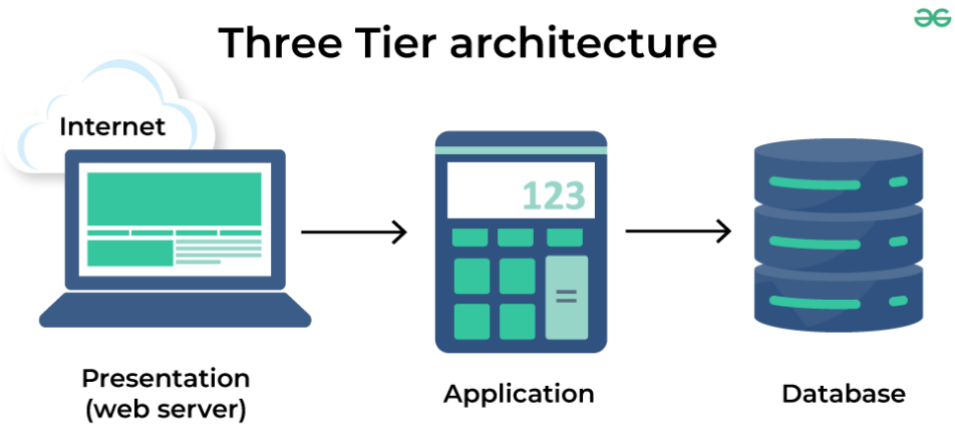


Figure 2.3 Three Tier architecture (source)

2.2.4 Model-View-Controller (MVC) Architecture

In the MVC (Model-View-Controller) pattern:

- The **Model** represents the application's data and associated business logic. It can be a single object or a complex network of related objects. In Java EE applications, data is stored within domain objects, typically deployed in an EJB module. Data is transferred between the database layer and the model through data transfer objects (DTOs) and accessed with data access objects (DAOs).
- The **View** provides a visual display of the data in the model. Each view represents a specific subset of the model, effectively filtering the data. Users interact with model data through the view, which enables them to trigger business logic that updates the model as needed.
- The **Controller** connects the view to the model and manages the application's flow. Based on user input, it determines which view to display and which business logic to execute. The controller receives input from the view, forwards it to the model for processing, and then selects and renders the appropriate view to present to the user. [33]

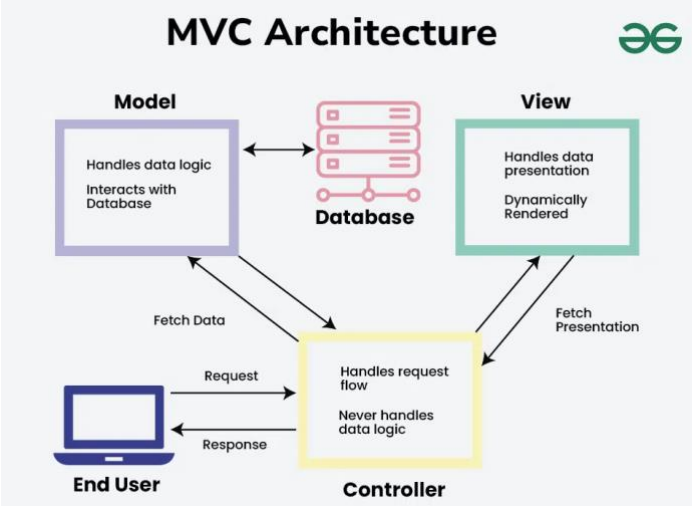


Figure 2.4 MVC architecture (source)

2.2.5 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural pattern where software components, known as services, are designed to provide specific, reusable functions. Each service within an SOA system encapsulates a discrete business function and interacts with other services via standardized interfaces and protocols, typically through message exchanges over a network. SOA emphasizes loose coupling between services, allowing each one to operate independently and be modified, replaced, or upgraded without affecting other services in the system [34].

SOA enables diverse applications to integrate smoothly, as services are technology-agnostic and communicate using standard protocols, such as SOAP or REST. This flexibility allows services to be developed and deployed across different platforms and languages, making it ideal for large, complex systems that require interoperability among varied applications and data sources. Additionally, SOA facilitates scalability, as services can be deployed across multiple servers and scaled based on demand.

The modular design of SOA also enhances system maintainability, as changes in one service do not necessitate changes in others. By aligning services with business processes, SOA enables organizations to streamline operations, respond swiftly to changing requirements, and reuse core services across various applications. Consequently, SOA is widely adopted in enterprise environments, where system integration, reusability, and adaptability are crucial.

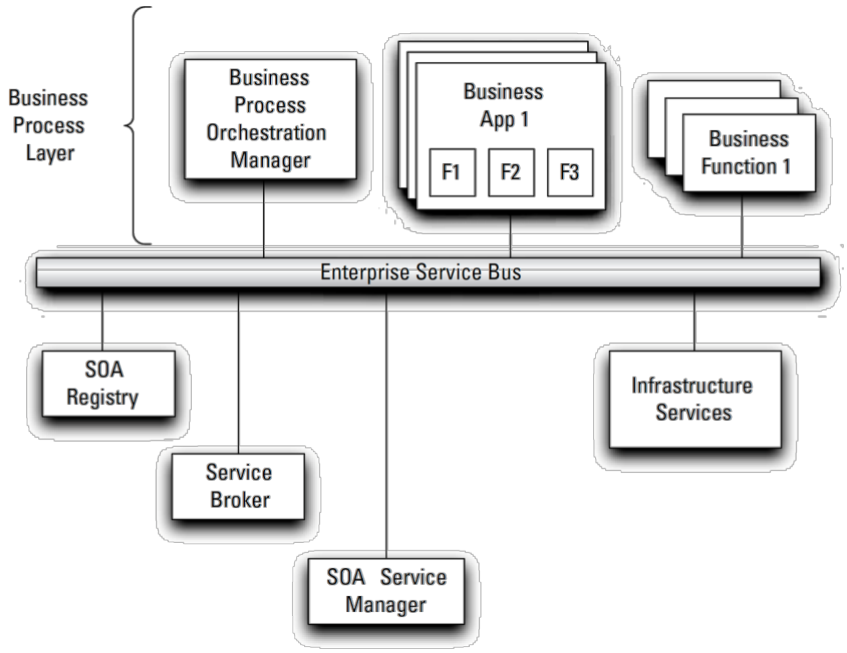


Figure 2.5 SOA architecture (source)

2.2.6 Microservices Architectural Pattern

The microservices architectural pattern is gaining widespread adoption among organizations, ranging from industry giants like Amazon, Netflix, and Spotify to small and medium-sized enterprises. This approach is characterized by the design of independent, autonomous services, each focused on a specific, clearly defined function.

Unlike traditional monolithic architectures, microservices promote a vertical decomposition of applications into discrete, business-oriented services. This structure enables each service to be developed, deployed, and tested independently, often by different teams utilizing diverse technology stacks. Microservices offer numerous advantages: each service can be built using the programming language and technology best suited to its function, enabling flexibility and optimization. Services can also scale individually, allowing for targeted resource allocation, and they can be deployed on infrastructure that best fits their requirements.

This decentralized, modular approach contributes to greater system maintainability and resilience. With smaller, independent services, the system is more fault-tolerant; the failure of one service does not impact the entire application, a risk often associated with monolithic systems. As a result, microservices architectures enable organizations to enhance scalability, streamline development processes, and improve system robustness [35]. Microservices architecture is a progression of the SOA architectural style. In SOA, each service represents a complete business capability, whereas microservices focus on smaller, highly specialized software components dedicated to a single task. Microservices overcome the limitations of SOA, making them better suited for modern, cloud-based enterprise environments [34].

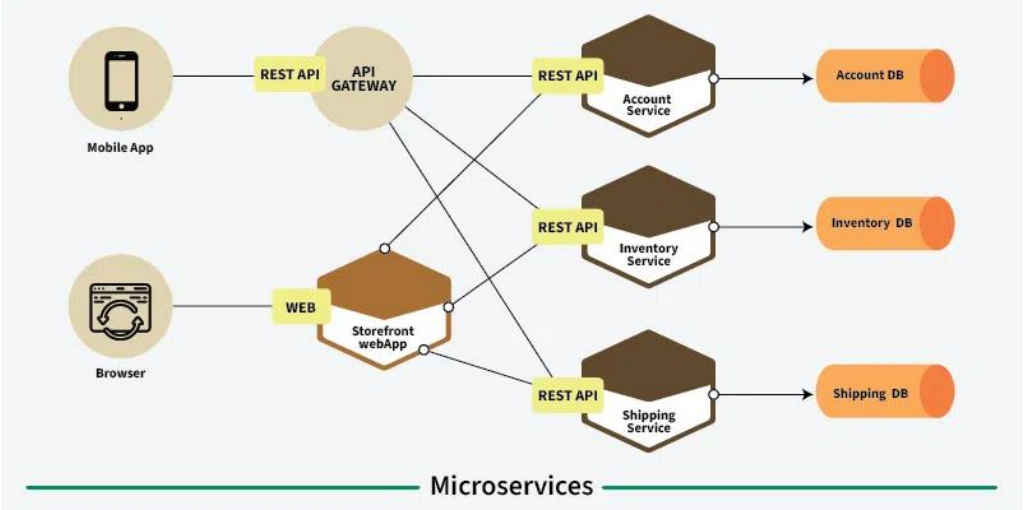


Figure 2.6 Microservices architecture (source)

2.3 Selected Architectural Patterns

In this work, we will focus on a select group of well-established architectural patterns: Client-Server, Three-Tier, and Model-View-Controller (MVC). Our decision to concentrate on these specific patterns stems from our perspective that AI-assisted software architecture is still an emerging field, with outcomes that are yet to be thoroughly validated. Given this, we aim to explore and experiment with a simple application to be implemented with well-established architectures, which provide a clear foundation and controlled environment for evaluating the impact of AI assistance. Engaging with more intricate patterns, such as Microservices or Service-Oriented Architecture (SOA), might introduce complexities that could obscure or complicate the analysis of AI's influence on architectural effectiveness.

Furthermore, by selecting foundational patterns, we can establish a baseline for future research. This will allow us to better compare results and gradually scale to more complex architectures in subsequent studies, once initial results have been validated. Another reason for this selection is the practicality of studying widely used, straightforward patterns, as they lend themselves well to experimentation and are commonly understood across various development environments. Thus, our focus on Client-Server, Three-Tier, and MVC enables a balanced exploration of AI's potential in software architecture while laying the groundwork for future advancements in this area.

2.4 Motivation of Our Approach

The insights gained from the discussed studies in section 2.1 point to key areas that remain underexplored or insufficiently addressed. Our approach is shaped by the following considerations:

1. **Scalability and Accessibility of Models**
   While existing work often focuses on large-scale, cloud-based LLMs like GPT-3.5 or heavily fine-tuned systems like ClassDiagGen, these approaches may not be viable for businesses or researchers concerned with privacy, cost, or scalability. Our work emphasizes exploring **local models**, such as LLaMA and Mistral, which can be run on-premises. This direction responds to a growing demand for AI solutions that prioritize data security and independence from cloud infrastructure.

2. **Evaluating the suitability and performance of Smaller Models**
   Many of the related studies focus on large, resource-intensive models, but smaller or quantized models (e.g., Q4 quantization) are often overlooked. Our approach aims to systematically test models across a spectrum of sizes, from small-scale models to high-capacity ones. This not only provides a more practical lens for resource-constrained environments but also explores whether smaller models can achieve comparable results in architecture generation.
3. **Combining Retrieval-Augmented Generation (RAG) with LLMs**
   Unlike the approaches discussed earlier in section 2.1, our research incorporates **RAG techniques** to enhance the LLM's ability to handle ambiguities in natural language requirements. By grounding the models with external knowledge, such as domain-specific RAG files, we aim to mitigate common alignment issues and explore if the overall consistency and correctness of generated diagrams can be improved by this method.
4. **Comprehensive Experimentation with Architectural Patterns**
   Previous studies have not systematically examined the impact of requesting the adoption of specific architectural patterns (e.g., Client-Server, Three-Tier, MVC) on the quality of the generated architectures. Our work addresses this gap by evaluating how well different LLMs respect these architectural principles and how the description of functional and non-functional requirements affects the outputs.

2.5 Research Questions

This work aims to explore the potential and boundaries of AI in assisting software development, with a focus on generating architectures, improving model training, evaluating AI's performance, and determining the suitability of various AI models and training materials. The following research questions guide our study:

1. **Can AI generate architectures that apply explicitly specified architectural patterns from a given textual problem description?**
   This question explores AI's ability to interpret problem statements or requirements and translate them into well-structured architectures that adhere to requested architectural patterns. By addressing this question, we aim to assess AI's capacity to create relevant architectures without extensive human guidance, thereby evaluating its potential to streamline the architectural design processes.
2. **What should a problem description contain to guide LLMs to produce meaningful architectures?**
   Exploring a suitable structure, content, and specificity of input descriptions is essential to refining AI's ability to generate accurate and effective architectures. This question seeks to identify which elements of a description are most useful in guiding AI toward producing architecturally sound outputs.
3. **Can responses be enhanced by using Retrieval-Augmented Generation and similar methods?**
   RAG and related techniques provide a means of enriching AI-generated responses with relevant context or knowledge. This question will investigate whether RAG can improve the accuracy, relevance, and quality of AI-assisted architectural designs and the potential limitations of this method.

4. **How can training be improved with RAG materials, and what effect do different materials have on the performance of LLMs?**
   This question addresses the impact of diverse training materials on the quality of AI responses in architecture. Understanding how various RAG sources influence AI's design abilities can help improve training methods and produce more reliable, context-aware responses.
5. **Are open-source models reliable for AI-assisted architecture, and what is the minimum acceptable model type/size?**
   The performance of open-source models in architecture assistance is examined here, with a focus on identifying the smallest, most efficient model capable of providing acceptable results. This question will inform recommendations for cost-effective, accessible AI solutions.
6. **How can we evaluate the effectiveness of AI assistance in software architecture design?**
   Evaluating AI's contributions to architectural design requires well-defined criteria and metrics. This question explores potential evaluation methods to measure the quality, accuracy, and overall utility of AI-generated architectural suggestions.

Each question is critical to understanding how AI can be meaningfully integrated into software architecture processes, from creation to evaluation. By investigating these aspects, this research aims to provide a foundational understanding of AI's role in architecture and to identify methods for optimizing its performance and reliability in real-world applications.

3 Approach

3.1 Parameters

To thoroughly examine our research questions, we decided to establish specific parameters to guide our process and ensure clarity in our experimental setup.

3.1.1 Architectures Considered

As previously discussed, we have chosen a subset of commonly used architectural patterns: Client-Server, Three-Tier, and Model-View-Controller (MVC). This selection is motivated by their simplicity, broad adoption, and their ability to provide a controlled environment for our experiments. By focusing on these foundational patterns, we can direct our efforts toward analyzing AI's effectiveness in adhering to widely recognized architectures, which simplifies isolating the AI's performance from the complexities introduced by more intricate patterns, such as Microservices or SOA.

3.1.2 Case study: The DCC app

The **Dummy Coordination Conversion (DCC)** Application is a software app that manages coordinate groups in cartesian and polar formats. The system allow users to convert, store, retrieve, modify, and delete coordinate groups.

This choice of a really simple and straightforward application allows us to center our attention on the AI's ability to respect the requested architectural pattern that satisfies very simple, specified requirements. Such a simpler app reduces complexity, making it easier to evaluate AI performance with a sharper focus on architectural and diagrammatic accuracy. Our textual input for the app description will include a high-level overview of the application, as well as specific Functional and Non-functional Requirements.

To gain a comprehensive understanding of how each architectural pattern influences our application, we developed three distinct versions of the app in Java, with each version adhering to one of the selected architectures: Client-Server, Three-Tier, and Model-View-Controller. We chose Java as the programming language for its robust type system and its direct support for object-oriented principles, which we believe make the architecture easier to generate, understand, and implement.

Functional Requirements Input

Initially, we focus on the Functional Requirements. To observe and evaluate differences in AI-generated outputs, we have prepared two versions of the functional requirements set. Each set has been crafted to highlight distinct aspects of the software's functionality and will be presented in

detail. Afterward, we will provide an in-depth comparison and analysis of the two sets to assess how their variations might influence the AI's class diagram generation and architectural fidelity.

This approach—providing different versions of requirements—aims to shed light on how requirement granularity and specificity impact the quality and accuracy of AI-assisted outputs, allowing us to gather meaningful insights into the optimal level of detail needed for AI-supported architecture generation. Figure 3.1 shows the FR set 1 and Figure 3.2 shows the FR set 2.

```
1. Coordinate Group Creation:
    - Users can create a coordinate group consisting of both Cartesian and polar coordinates.
    - Input one type of coordinate (either Cartesian or polar), and the software should handle the conversion to the other type.
    - A coordinate group consists of:
      - Cartesian: (x, y)
      - Polar: (r, θ)

2. Coordinate Group Storing:
    - Store coordinate groups in a MySQL database with the following attributes:
      - Unique auto-generated identifier (integer).
      - User-defined string label for identification.
      - Automatically assigned timestamp.

3. Coordinate Group Retrieval:
    - Retrieve coordinate groups either by:
      - Label: Search using a user-defined string label.
      - View all: Retrieve and display all saved coordinate groups.

4. Coordinate Group Modification:
    - Select and modify details of a coordinate group (Cartesian or polar values, label) and save the updated version.

5. Coordinate Group Deletion:
    - Delete a coordinate group from the database.
```

Figure 3.1 The first set of functional requirements (fr1)

```
1. Logic
1.1 Define cartesian coordinates as a pair (x, y) of high-precision numbers
1.2 Define polar coordinates as a pair (r, θ) of high-precision numbers
1.3 Convert cartesian (x, y) to polar (r, θ) coordinates
1.4 Convert polar (r, θ) to cartesian (x, y) coordinates
1.5 A coordinate group contains cartesian (x, y) and polar (r, θ) coordinates

2. Data operations
2.1 CreateOp: create a unique auto-generated integer identifier (id) for the coordinate group edited, add a timestamp, and
save the label and the values of the coordinates group as a new record into the database
2.2 ReadOp: Create a list of all saved coordinate group records. For each record, show: id, label, cartesian coordinate
values, polar coordinate values, datetime added or modified), sorted by id
2.3 UpdateOp: Update a coordinate group record with new values entered by the user and save the updated record of the
coordinates group into the database
2.4 DeleteOp: Delete a coordinate group record from the database

3. User interactions
3.1 Input one new coordinate group and one string to label the group
3.2 When the user inputs cartesian coordinates (x, y), calculate the corresponding polar (r, θ) coordinates by converting
cartesian to polar
3.3 When the user inputs polar coordinates (r, θ), calculate the corresponding cartesian coordinates by converting polar
to cartesian
3.4 When "add" is selected, execute CreateOp for the current values of the coordinate group and the label
3.5 When "update" is selected, execute the UpdateOp for the selected record in the list created by the ReadOp
3.6 When "delete" is selected, execute the DeleteOp for the selected record in the list created by the ReadOp
3.7 When initializing and when "add", "update" or "delete" is executed, refresh and display the list created by the ReadOp
3.8 Filter the list of coordinate groups displayed, by a label value entered by the user
3.9 Clear the active filter and display all saved coordinate group records, as in 3.7
```

Figure 3.2 The second set of functional requirements (fr2)

The two sets of functional requirements describe the same core functionalities but differ in their structure, granularity, and focus. Here are the essential differences:

Structure and Organization

- **First Set:** Organized by high-level functional areas (Creation, Storing, Retrieval, Modification, Deletion). Each function is presented as a user-facing feature.

- **Second Set:** Divided into three major categories—Logic, Data Operations, and User Interactions—which focus on lower-level implementation details, breaking down each functional area into smaller, specific operations.

Granularity and Detail Level

- **First Set:** Presents requirements in a straightforward, user-centric format, specifying key actions without diving into precise operational details. For example, it explains "Coordinate Group Retrieval" at a general level, describing how users can search by label or view all records.
- **Second Set:** Goes into finer detail on each operation, specifying exact actions for different stages (e.g., "CreateOp," "ReadOp") and what each entail. The second set even details internal system operations, such as sorting records by ID or refreshing the display after each action.

Technical vs. User-Centric Language

- **First Set:** Describes functions using language that aligns closely with user tasks and high-level goals, making it easier for end-users or stakeholders to understand.
- **Second Set:** Uses more technical language and terms, such as "CreateOp," "ReadOp," and "high-precision numbers," making it more suited to developers or technical audiences. It also emphasizes data handling and internal logic (e.g., converting coordinates, refreshing lists) rather than just user actions.

Specification of User Interactions

- **First Set:** Assumes users perform actions like retrieval and modification without detailing interaction steps.
- **Second Set:** Explicitly details user interactions, including specific actions like "add," "update," and "delete," with explanations for each step (e.g., converting coordinates when inputting data, displaying the list after each action). It specifies user actions in a systematic manner, clarifying exactly how users engage with each functionality.

Emphasis on Data Processing and Refreshing

- **First Set:** Mentions storing, retrieving, and deleting coordinate groups but doesn't elaborate on data refresh mechanisms or display updates.
- **Second Set:** Emphasizes refreshing the displayed list upon every create, update, or delete operation, highlighting the user's view of the data as an important part of the functionality. It also provides options for filtering or clearing filters on the display, enhancing data management clarity.

Focus on Data Transformation and Storage

- **First Set:** Implies the need for coordinate conversion but does not detail the logic behind transformations.

- **Second Set:** Explicitly outlines coordinate transformations (Cartesian to polar and vice versa) in the "Logic" section, specifying the internal operations required to support user inputs and conversions.

In summary, the **first set** is user-oriented, outlining high-level functionalities in accessible language, ideal for non-technical stakeholders. In contrast, the **second set** is technical and implementation-focused, detailing operations, user interactions, and internal processes, making it more suitable for developers needing precise guidance on each operation. Together, these distinctions show that the first set is tailored to functional understanding, while the second set emphasizes operational clarity and technical precision.

We anticipate that most would conclude that the second set of requirements is more likely to lead to better generated architectures by LLMs. This expectation arises from the second set's emphasis on explicitly defined business logic, more granular details for each function, and a clearer delineation of architectural components. The structured nature of the second set may guide LLMs to produce more precise and well-aligned architectures, potentially resulting in classes that are both more relevant to the application's business context and closely aligned with architectural standards.

However, while this reasoning appears logical, it remains untested. In our experiments, we aim to validate whether these assumptions hold true by evaluating the quality and accuracy of class diagrams generated from each set of requirements. Through this, we hope to derive meaningful insights into the influence of requirement specificity and detail on AI-assisted architectural generation, providing evidence-based conclusions on the optimal requirements format for LLM-driven design.

Non-functional requirements set

Figure 3.3 shows the first NFR set.

```
- Frontend: Developed using Java with the Swing framework for user interaction.
- Database: MySQL for persistent storage of coordinate groups.
- TCP/IP Connection: To handle the communication between the application and the database.
```

Figure 3.3 The first set of non-functional requirements (nfr1)

Figure 3.4 shows the second NFR set.

```
- Frontend: To be developed using Java with the Swing framework for user interaction
- Logic: To be developed using Java classes, according to the object-oriented design principles
- Database server: MySQL for persistent storage of coordinate groups
- Database connection: TCP/IP connection for the communication between the application and the database
- Design: Apply the "Single Responsibility Principle"
```

Figure 3.4 The second set of non-functional requirements (nfr2)

Upon comparing the two sets of non-functional requirements, it is evident that the second set offers a more detailed and structured approach, hopefully leading LLMs to improved clarity and adherence to design principles.

The **first set** provides a high-level overview of the non-functional requirements, focusing on essential components like the frontend framework (Java with Swing), database choice (MySQL), and communication protocol (TCP/IP). However, it lacks specificity regarding design principles or object-oriented programming practices, which may leave certain implementation details open to interpretation.

In contrast, the **second set** enhances clarity by breaking down requirements into more specific categories, such as "Logic" and "Design," and introduces guidelines for structuring code. For example, it specifies that the logic layer should be developed with Java classes following object-oriented principles, ensuring a modular and maintainable structure. Furthermore, the inclusion of the "Single Responsibility Principle" in the design section provides an explicit directive for the organization of classes, reinforcing clean and focused code.

The second set's emphasis on object-oriented design and single-responsibility guidelines could result in a more maintainable and scalable system displayed by the generated class diagram, as it directs developers toward a robust architectural foundation.

In summary, while the **first set** gives a concise overview of technical requirements, the **second set** provides a more comprehensive roadmap by incorporating object-oriented and design principles. This added detail may ultimately enhance the system's structure and quality, making it better suited for complex projects where clarity and maintainability are paramount. Our forthcoming experiments will examine whether these differences in non-functional requirements lead to observable improvements in class diagram in AI-assisted implementations.

It is important to note that the first set of functional requirements will be paired with the first set of non-functional requirements, and likewise, the second sets will be combined. This alignment reinforces our expectation that using the second sets together could lead to better results.

3.1.3 Prompt input

For our experiment, we have chosen a single, comprehensive prompt:

```
Task Description:
You are tasked with processing a software description and its requirements to generate a class diagram using PlantUML.
The class diagram should respect the requested software architecture, define all necessary classes, and accurately
represent associations between them.

Overview: {Software_app_overview}

Software Architecture: {Architecture}

Functional Requirements: {FR}

Non-Functional Requirements: {NFR}

Generate a class diagram using PlantUML that defines all necessary classes and associations based on the described
architecture, requirements, and functionality. Ensure that:
- The diagram reflects the separation of concerns based on the requested architecture.
- Each class is properly defined with attributes and methods.
- All associations (e.g., composition, aggregation, or inheritance) between the classes are included in plantUML and are
clearly represented.
- Create the appropriate packages to include the classes.
```

Figure 3.5 Prompt

Prompt as shown in Figure 3.5 was designed to encapsulate all relevant information, from the application's overview and architecture to its functional and non-functional requirements, guiding the LLM in generating a complete class diagram in PlantUML. By including both high-level

guidance and specific directives on attributes, methods, associations, and package organization, the prompt aims to provide the LLM with the necessary context to produce accurate and organized class diagrams.

The decision to use only one version of the prompt, rather than creating multiple versions is driven by a few key considerations:

- **Avoiding Redundancy**: Unlike the functional requirements, where different levels of granularity and specificity could yield significantly varied responses, the prompt here is designed to be comprehensive, covering all necessary aspects (overview, architecture, functional and non-functional requirements). Creating additional versions of the prompt could lead to redundancy, as each would need to repeat similar instructions to capture the same complete context for the LLM.
- **Maintaining Consistency in Evaluation**: With multiple versions of the functional and non-functional requirements, as well as variations in the RAG files (as we will discuss later), our focus is on evaluating how changes in input detail influence the AI's interpretation and output.
- **Focus on Core Research Questions**: Our primary goal is to explore how AI responds to differences in the FR/NFR sets, RAG files, and model performance, not how it responds to variations in the prompt structure itself. By keeping the prompt consistent, we can better isolate the impact of the remaining changes on output quality without confounding results with prompt-based differences.

3.1.4 Selection of LLMs

To gain insights into how different models perform, we selected a variety of models, chosen based on specialization and parameter size, which affects memory requirements, as shown in the Table 3.1

Model Name	Source	Parameter size	Quantization	Model Task
llama3.1:latest	local	8B	Q4_0	Multilingual Text and code
phi3:medium-128k	local	14B	Q4_0	• commercial and research use in English. • applications which require: 1) memory/compute constrained environments 2) latency bound scenarios 3) strong reasoning (especially math and logic) 4) long context
gemma2:27b	local	27.2B	Q4_0	• Multiple data modalities, • Variety of text generation tasks, including question answering, summarization, and reasoning.

Model Name	Source	Parameter size	Quantization	Model Task
command-r	local	32.3B	Q4_0	• Optimized for long context tasks such as retrieval-augmented generation (RAG) and using external APIs and tools
mixtral:8x7b	local	46.7B	Q4_0	• It gracefully handles a context of 32k tokens. • It handles English, French, Italian, German and Spanish. • It shows strong performance in code generation.
gpt-4o (chatGPT4o)	online	200B	N/A	-
gpt-4o (Software Architecture Visualiser)	online	N/A	N/A	• A tool that automatically generates interactive, real-time diagrams like PlantUML from codebases, aiding in the understanding and design of software systems
gemini	online	N/A	N/A	-
copilot	online	N/A	N/A	-
claude3.5 Sonnet	online	175B	N/A	-

Table 3.1 LLM selection

This variety allows us to evaluate models with different parameter sizes, ranging from smaller, resource-efficient models like **llama3.1:latest** to more extensive, high-capacity models like **command-r**, providing insights into how model size and capacity affect the output. Additionally, we included models with specific optimizations, such as **GPT-4o (Software Architecture Visualizer)**, tailored for architectural tasks, as well as more general-purpose models like **Copilot**. By testing this diverse lineup, we also address the growing interest in using smaller, locally hosted models as alternatives to larger, cloud-based ones, an approach that aligns with the need for data security and privacy, especially in business environments handling sensitive information.

All the models selected for this study utilize the **Q4 quantization parameter**, which reduces the precision of the model's weights to 4 bits. Standardizing this parameter across all models was a deliberate choice, particularly for these initial stages of our research into AI-assisted software architecture. This approach ensures a fair comparison of the models' capabilities while also allowing us to explore the practical advantages of using quantized LLMs in resource-constrained environments.

This comprehensive evaluation allows us to observe how model specialization (e.g., general-purpose models versus those optimized for specific applications) and deployment preferences (local

versus online) influence the class diagram generation process. It enables us to compare performance across a spectrum of scenarios, identifying which types of models excel at producing high-quality class diagrams aligned with specified architectural patterns while balancing scalability, security, and computational efficiency.

3.1.5 Retrieval Augmented Generation (RAG)

What is RAG?

Large language models (LLMs) can sometimes be unpredictable. While they can provide accurate answers on some occasions, they may also generate irrelevant or inaccurate information, largely because they rely on statistical patterns in language rather than actual understanding. LLMs are trained to identify word statistical relationships, but they lack true comprehension of meaning.

Retrieval-Augmented Generation (RAG) is an AI framework designed to improve the accuracy and relevance of LLM-generated responses. By grounding the model in external, up-to-date knowledge sources, RAG supplements the LLM's internal knowledge with verified information, helping to ensure that responses are accurate and reliable.

Implementing RAG in LLM-based question-answering systems offers two primary advantages: it provides the model with access to current, verified facts, and it allows users to view the sources behind the model's responses, which supports transparency and verifiability. This traceability builds trust, as users can check the validity of the information provided.

RAG also brings additional benefits. By anchoring an LLM's output to external, factual data, it minimizes the chances of the model relying solely on its internal parameters. This reduces risks such as inadvertent exposure of sensitive data or the generation of "hallucinated" (incorrect or misleading) information. Furthermore, RAG reduces the need for frequent retraining or parameter updates to keep the model current, which in turn lowers the computational and financial costs associated with maintaining LLM-powered chatbots, especially in enterprise environments.

In essence, RAG is the process of retrieving relevant contextual data from an external source and feeding it to a large language model along with the user's prompt. This external information enriches the model's responses, enhancing its base knowledge to produce more accurate and contextually appropriate outputs, whether text or images.

What are Embeddings and how they used in RAG?

Embeddings are mathematical representations that capture the semantic meaning of words, sentences, or entire documents by encoding them as dense vectors in a high-dimensional space. The purpose of embeddings is to position text with similar meanings closer together in vector space, enabling efficient comparison of concepts based on their meanings rather than exact wording.

In the context of **Retrieval-Augmented Generation (RAG)**, embeddings play a crucial role in retrieving relevant information from a large corpus based on a user's query. This retrieved information is then used to help the model generate contextually rich and accurate responses. Here's how embeddings function in RAG:

1. **Creating Embeddings for the Corpus**
   Before any queries are processed, embeddings are generated for each document or text segment in the corpus. This step effectively maps the entire corpus into a high-dimensional vector space, allowing similar documents to be clustered by meaning.
2. **Query Embedding**
   When a user submits a query, the query text is also transformed into an embedding. This query embedding exists within the same vector space as the corpus embeddings, representing the semantic intent of the user's input.
3. **Similarity-Based Retrieval**
   The system then compares the query embedding with all the document embeddings in the vector space. Using similarity measures (such as cosine similarity), it identifies and retrieves documents most closely aligned with the query's meaning.
4. **Augmentation with Retrieved Information**
   The retrieved documents serve as supplementary knowledge or context for the model, which then generates a response grounded in real-world data. This augmentation enables the model to provide more accurate, nuanced, and contextually relevant answers.

Through embeddings, RAG can effectively retrieve and leverage semantically relevant information, enhancing the quality and reliability of the model's responses by rooting them in external, meaningful context. Figure 3.6 shows the flow of RAG technique.

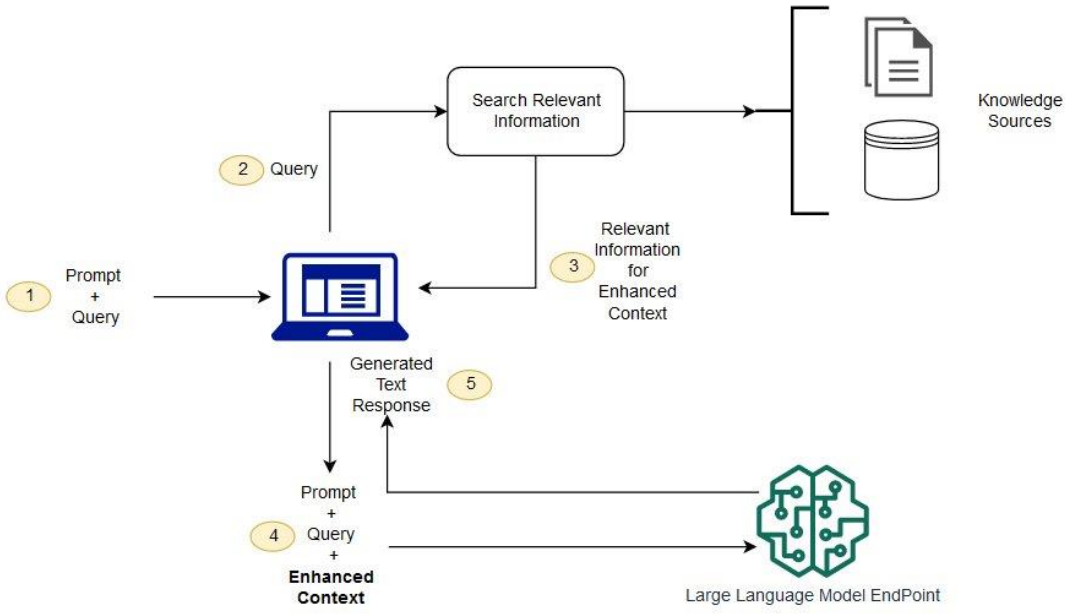


Figure 3.6 Conceptual flow of RAG method (source)

Which techniques could apply to enhance Retrieval Augmented Generation?

As Generative AI rapidly advances, researchers and practitioners are exploring a variety of techniques to improve large language models (LLMs) in Retrieval-Augmented Generation (RAG) tasks. By combining the strengths of language models with information retrieval (IR)

systems, RAG enables LLMs to produce more informed and contextually relevant responses. Below, we discuss some key techniques to enhance the effectiveness of LLM-based RAG systems.

1. Selecting the Right Embedding Model:

The effectiveness of retrieval through similarity search is significantly influenced by the quality of embeddings that represent the documents and queries. Therefore, choosing an embedding model carefully is essential, as it directly impacts retrieval accuracy and, consequently, the relevance of LLM-generated responses. Options for embedding models include:

- **Self-hosting** an open-source model,
- Using a **cloud-based provider** offering open-source models,
- Choosing a **proprietary cloud solution** (e.g., OpenAI),
- Utilizing an **integrated end-to-end solution** that includes embedding services, such as Enterprise Bot.

To determine the best embedding model for RAG, several metrics are useful:

- **Average and Retrieval Average Performance**: Measures average model performance on benchmarks, with a specific focus on retrieval effectiveness for RAG tasks.
- **Model Size and Memory Usage**: Larger models often yield better accuracy but require more computational resources.
- **Embedding Dimensions**: Higher-dimensional embeddings can capture more semantic nuances but increase memory requirements.
- **Max Tokens**: Indicates the maximum text length that a single embedding can handle, crucial for extensive text queries.

2. Optimizing Chunking Strategy:

Chunking is the process of breaking down documents or data into smaller units, which can enhance retrieval efficiency and precision. Effective chunking strategies ensure the RAG system can manage longer text inputs, extract relevant information efficiently, and produce coherent, contextually accurate responses. Common chunking strategies include:

- **Sliding Window Chunking**: Divides text into overlapping windows to maintain context.
- **Document-Based Chunking**: Segments based on document boundaries.
- **Semantic Chunking**: Splits based on meaning, grouping related information.
- **Agent-Based Chunking**: Designed for multi-agent interactions.

By experimenting with these chunking techniques, researchers can find the optimal balance between retrieval accuracy, computational efficiency, and response quality.

3. Applying Metadata Filters:

In many RAG applications, documents in the knowledge base may contain valuable metadata, such as source credibility, timestamps, or topical relevance. Leveraging this metadata can enhance response quality and relevance by allowing the RAG system to filter or prioritize specific information based on predefined criteria. For instance, the model can prioritize recent or

authoritative sources, or filter by topic when handling domain-specific tasks, such as in healthcare, finance, or legal applications. Metadata filtering allows RAG systems to select the most trustworthy and contextually relevant information, enhancing the reliability and focus of the generated response.

4. Implementing Query Transformation Techniques:

Sometimes, user input may not align well with the structure of the knowledge base or the LLM's expectations. Query transformation techniques adjust and optimize user input to improve retrieval relevance and accuracy. Examples include:

- Rephrasing or expanding the query to capture additional context,
- Extracting keywords or entities to refine retrieval,
- Translating terms into a specific language or domain-specific jargon,
- Adding task-specific constraints to guide retrieval. Through query transformation, the RAG system better captures the user's intent, retrieves relevant documents, and generates responses that more accurately address the query.

5. ReRanking:

The initial retrieval in a RAG system may not always yield the most relevant documents at the top of the results. ReRanking techniques address this by refining and reordering the retrieved documents, ensuring that the most pertinent information is prioritized for response generation. Effective ReRanking can be achieved through several approaches:

- **Leveraging Additional Signals**: Use factors like semantic similarity, source credibility, or domain-specific relevance to assess and reorder retrieved content based on its alignment with the query.
- **Applying Advanced Ranking Models**: Use machine learning approaches, such as neural ranking models or reinforcement learning, to learn optimal ranking criteria from data, improving the system's ability to identify the most valuable information.
- **Incorporating User Feedback**: Integrate interaction data or user feedback to refine the ranking criteria over time, aligning responses with user expectations and improving the overall relevance.

These techniques each address specific challenges in RAG systems, such as retrieval precision, query optimization, and document ranking. By applying these enhancements, RAG systems can significantly improve their response quality, ensuring outputs that are more trustworthy, relevant, and contextually accurate. As RAG use expands across industries, these improvements contribute to more robust, efficient, and effective AI-powered solutions.

While all of these techniques could potentially enhance the RAG process, our experimentation focuses on the first two: **embedding model choice** and **chunking strategy**. By concentrating on these specific techniques, we aim to better understand their impact on retrieval accuracy and the quality of generated responses.

Embedding Models:

For embedding models, we selected the following two options as shown in Table 3.2:

- **nomic-embed-text-v1.5** is optimized for general-purpose embeddings, often performing well in capturing broad semantic relationships both for short and long context.
- **text-embedding-3-large**, on the other hand, could output larger embedding vectors and may yield finer-grained, detailed embeddings that can enhance retrieval precision in complex queries.

Embedding Model Name	Provider	Parameters size	Context window length used to generate the next token (num_ctx)	Dimension	Modeling tasks
nomic-embed-text-v1.5	nomic-ai	137M	8192 tokens	768	• can only be used to generate embeddings for short and long context
text-embedding-3-large	openAI	N/A	8192 tokens	3072	• high-precision tasks, where capturing the nuances of language is critical. • ideal for complex applications such as deep semantic search, advanced recommendation systems, and sophisticated text analysis.

Table 3.2 Embedding Models selection

These models represent distinct approaches in embedding performance and dimensionality.

By using these two contrasting embedding models, we aim to evaluate how the choice of embedding model affects RAG outcomes, specifically in terms of retrieval relevance and alignment with the given architectural and functional requirements. This comparison offers valuable insights into the role of embedding selection in enhancing LLM-assisted architectural design.

Chunking Methods:

For our experiment, we selected two Chunking Methods aimed at optimizing the retrieval and coherence of text chunks within the RAG process: recursive chunking and semantic chunking. By experimenting with these two distinct approaches, we can explore how different chunking strategies impact the relevance and cohesion of retrieved content.

The recursive chunking method resembles a simpler approach, similar to chunking based on a fixed text length. It splits text recursively to fit within the required size, focusing primarily on structural elements rather than meaning.

In contrast, the **semantic chunking method** is a more advanced approach that leverages the embedding model in the chunking process. By grouping text based on semantic similarity, this method strives to keep conceptually related text together, enhancing the meaningfulness of each chunk.

Recursive Character Split

This method is widely recommended for handling generic text. It utilizes a prioritized list of characters, attempting to split the text by each character in sequence until the chunks reach the desired size. The default character list is ["\n\n", "\n", " ", ""], which first tries to keep paragraphs together, then sentences, and finally words. By preserving these larger units where possible, this method maintains semantically related pieces of text, which can improve retrieval accuracy by keeping related concepts within the same chunk.

Semantic Chunking

Semantic chunking is designed to extract and retain meaning by assessing the semantic relationships between chunks. Rather than splitting text purely by characters, this method keeps together portions that are closely related in meaning. This approach is particularly useful for ensuring that each chunk represents a cohesive idea or theme, which can lead to more accurate and contextually relevant retrieval. Semantic chunking involves taking the embeddings of every sentence in the document, comparing the similarity of all sentences with each other, and then grouping sentences with the most similar embeddings together. For this purpose we have employed the embedding models that we have discussed in the above section.

For implementation, we used functions from the Langchain framework: RecursiveCharacterTextSplitter for the recursive character split and SemanticChunker for semantic chunking. By comparing these two methods, we aim to observe how chunking strategy impacts the coherence and relevance of the retrieved information, ultimately affecting the quality of AI-generated outputs.

RAG Document Sources: Content and Influence on Results

To enhance model performance in generating class diagrams, we supplement the LLM with well-documented presentations of the requested architectures. By providing structured RAG files, we aim not only to improve output quality but also to explore how variations in the source and structure of this information affect results. For this purpose, we created three types of RAG files, each with distinct content or format, to evaluate their impact on the final output.

1. **RAG File from Software Engineering by Ian Sommerville**
   The first RAG file is drawn from a foundational source: *Software Engineering, 10th Edition* by Ian Sommerville. This book is widely respected in the field and covers core software engineering practices in depth. By using content from such an authoritative source, we provide the model with comprehensive and reliable information, which we expect will enhance its ability to generate accurate class diagrams. Including material from Sommerville's text allows us to evaluate the effectiveness of an academically rigorous, structured knowledge base on the model's architectural comprehension.
2. **Separated Architecture Documents**
   In the second approach, we still rely on Sommerville’s book as the information source but modify the format. Here, we separate the documentation for each architecture type into distinct

documents. This approach allows us to pair the RAG file specifically with the architecture requested in the prompt. By isolating each architecture into its own document, we aim to test whether delivering focused, architecture-specific documentation improves the model's ability to produce diagrams that accurately adhere to the requested architecture pattern. This approach helps assess the potential benefits of providing segmented, highly targeted information over a single, combined document.

3. **RAG File Compiled from Various Web Sources**
   For the third RAG file, we gather information from a variety of online resources, including Wikipedia, GeeksforGeeks, and IBM's documentation. This file provides the model with a broad spectrum of perspectives and explanations, potentially adding a layer of diversity to the information. By combining material from different online sources, this RAG file reflects the variety of explanations and nuances commonly found in less formal, more accessible resources. Testing this RAG file allows us to investigate whether varied, multi-sourced information impacts the model's architectural output, especially when compared to the rigorously structured content from academic sources.

In summary, by experimenting with RAG files from these three sources, we aim to understand how differences in content source and document structure influence the quality and adherence of AI-generated class diagrams to specified architectural patterns. This exploration offers insights into the optimal approach for supplementing LLMs with external knowledge for complex architectural tasks.

3.2 Investigation Planning

3.2.1 Reference Architecture and Implementation

To gain a comprehensive understanding of how each architectural pattern impacts our application, we implemented three distinct versions of the app in JAVA, each adhering to one of the selected architectures (Client-Server, Three-Tier, and Model-View-Controller). This approach allows us to observe how each architecture shapes the structure, data flow, and modularization within the application.

For each version, we took care to rigorously adhere to the principles and guidelines associated with each architectural pattern, as discussed in 2.2. This included ensuring clear separation of concerns, modularity, and consistency in data handling and interaction across components. By respecting these core architectural principles, we aim to establish a baseline that AI-assisted architecture outputs can later be compared against.

Following the implementation of each version, we used the **Visual Paradigm** tool to reverse-engineer the Java code into UML class diagrams. This reverse engineering process allowed us to generate visual representations of each architecture, accurately capturing the relationships, dependencies, and hierarchies within the implemented code.

Below, we present the UML class diagrams derived from our implementations, intended as benchmarks for evaluating the architectural adherence of AI-generated diagrams. These diagrams offer a clear and structured view of the components, illustrating how each architectural decision shapes the organization and interaction of classes within the application.

While we do not expect the diagrams from LLMs to be identical to these, they provide a standard for comparison. We remain open to new approaches, provided they align with the established architectural patterns and design principles.

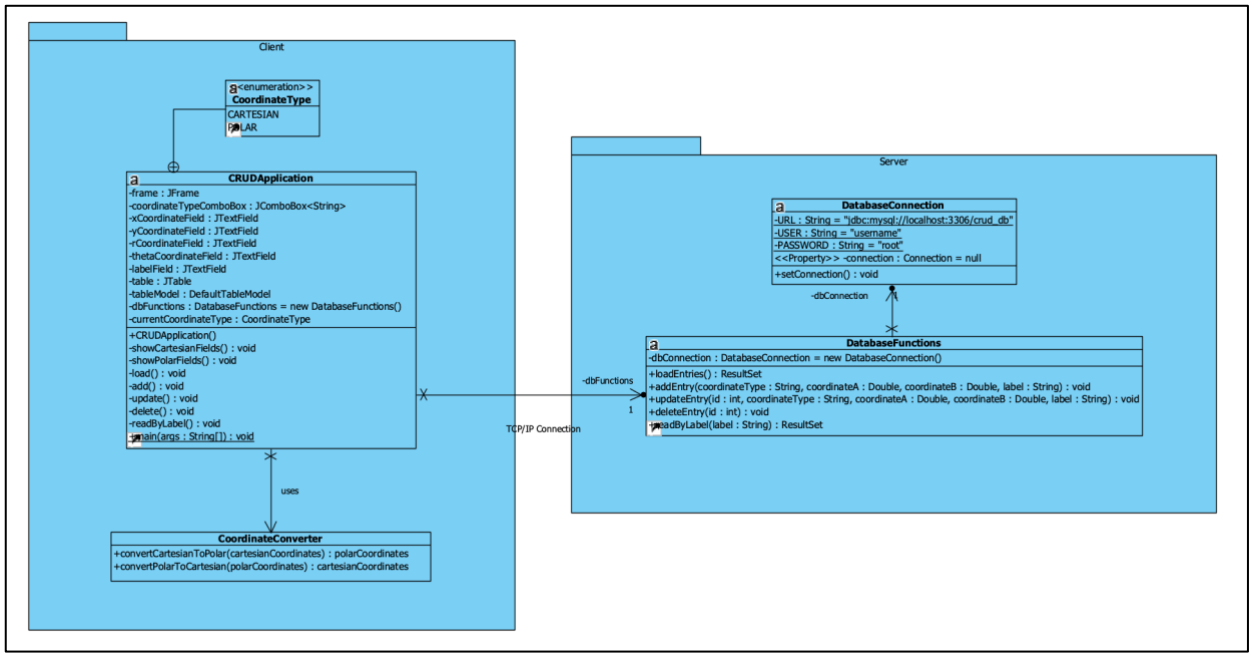


Figure 3.7 Client-Server architecture (DCC app)

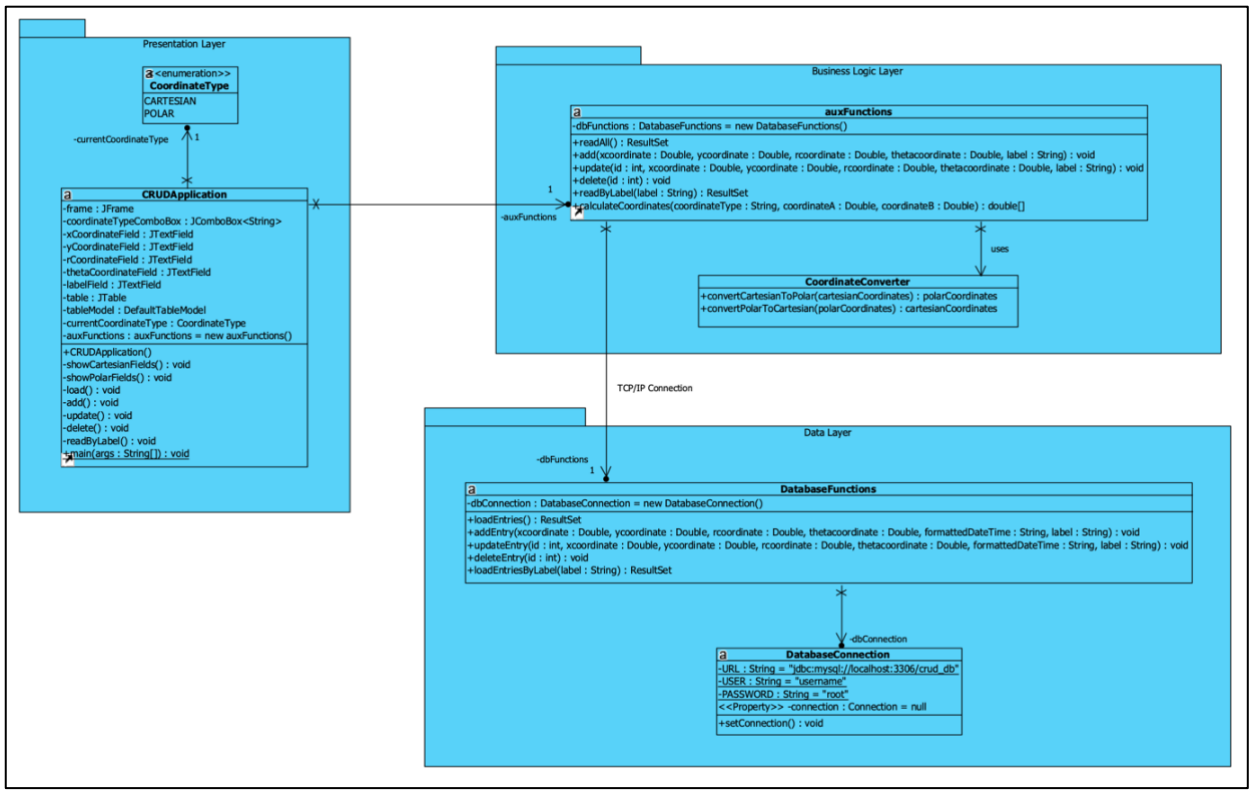


Figure 3.8 Three Tier architecture (DCC app)

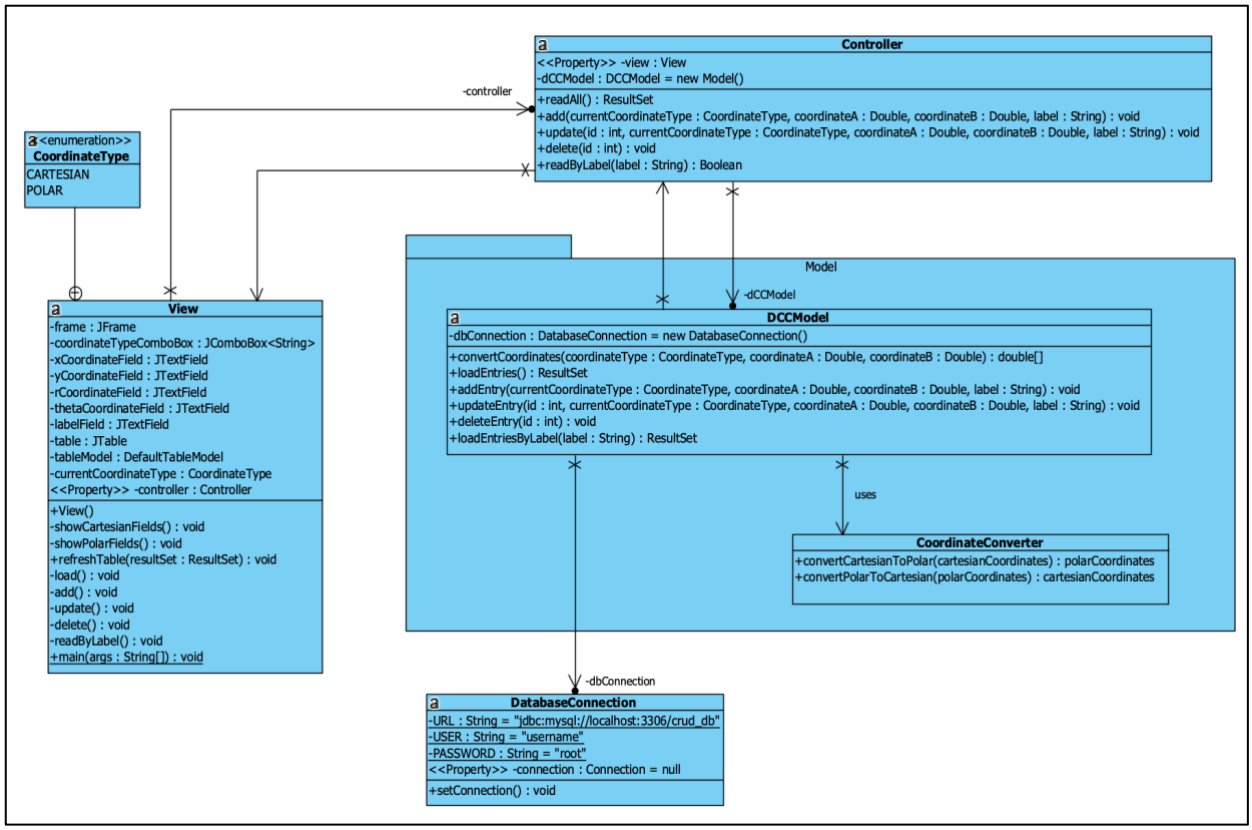


Figure 3.9 MVC architecture (DCC app)

3.2.2 The PlantUML Tool

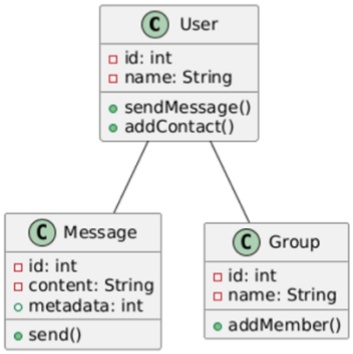
Using a model-oriented tool such as Visual Paradigm for both the generation and evaluation of UML diagrams presented certain technical challenges. Specifically, using Visual Paradigm would require us to approach the workflow in one of two ways, each with limitations.

In the first approach, we could work with class diagrams as images. However, this would restrict us to using only large language models (LLMs) capable of generating and recognizing images, which would divert the project's workflow from its initial objectives. Additionally, relying on image recognition introduces challenges, such as reduced accuracy and increased difficulty in processing and validating diagram components.

The second approach would involve requesting class diagrams from LLMs in XML format, which we could then import into Visual Paradigm. However, this option also posed issues. XML-based class diagrams require complex syntax, which may vary across different platforms, leading to potential compatibility issues and inconsistencies. Moreover, generating XML code could be challenging for LLMs, potentially resulting in errors or incomplete diagrams.

To address these concerns, we chose to use PlantUML. PlantUML combines the ability to create detailed and well-structured UML diagrams with the simplicity of a code-like, text-based language that LLMs can easily generate and interpret. This choice allows us to avoid compatibility issues associated with XML while also sidestepping the challenges of image recognition, thus maintaining a streamlined workflow. PlantUML's straightforward syntax ensures that any LLM with basic text generation capabilities can produce and recognize the diagrams, making it an ideal solution for our needs.

PlantUML also provides a variety of options for visualizing UML diagrams. It supports multiple platforms and tools, including an official web server that functions as an editor and libraries for Java, Python, and React. For our purposes, we leveraged PlantUML's Java .jar file, which allowed us to render and view diagrams directly as images. This functionality gave us an efficient way to both visualize and evaluate the generated class diagrams, streamlining our workflow by enabling quick conversion from PlantUML code to image format for analysis. In Figure 3.10 we can see an example of plantUML code and its corresponding UML diagram.




```
@startuml

'Classes:
'-------------

class User {
  - id: int
  - name: String
  + sendMessage()
  + addContact()
}

class Message {
  - id: int
  - content: String
  + metadata: int
  + send()

}

class Group {
  - id: int
  - name: String
  + addMember()
}


'Associations:
'-------------

"User" -- "Message"
"User" -- "Group"

@enduml
```

Figure 3.10 Example of plantUML code

3.2.3 Deployment, Tools, and Setup

For deploying and running our experiment, we used an Ubuntu 22.04 server equipped with 128 GB RAM. This setup provided the necessary computing resources to handle the large language models (LLMs) and extensive processing required for generating and evaluating UML class diagrams across various scenarios.

Tools for Communication with LLMs

To interact with local LLMs, we utilized the **Ollama framework**, a robust platform designed for managing and running large language models like Llama 3.1 and Mistral on local machines. Ollama allows for flexible integration of advanced AI capabilities, providing developers with powerful tools for tool-calling and model management. Ollama supports various operating systems, including Windows, macOS, and Linux, and offers multiple usage options:

- Direct command-line access through ollama run <modelName>,
- API calls to http://localhost:11434/api/generate,
- Integration through programming frameworks.

For our experiment, we opted to interact with Ollama through Python scripts, leveraging the **LangChain** library to create structured workflows with LLMs. LangChain provides extensive support for LLM integration, making it easier to manage prompt-based responses and RAG processes. In addition to Ollama, we used **OpenAI API** to access the embedding model *text-embedding-large-3* and the *gpt-4o* model for generating embeddings and retrieving enhanced outputs.

RAG and Database for Vector Storage

In the RAG process, we required a database to temporarily store vector embeddings from the chunked text. For this, we utilized **Chroma**, an open-source, AI-native vector database tailored for developer productivity and tightly integrated with LangChain. Chroma enabled efficient storage and retrieval of vectorized data, essential for managing embeddings in the RAG workflow.

Experiment Setup and Execution

To structure the scenarios and automate the class diagram generation process, we created a table that contains detailed information about each generation case, referred to as "scenarios." This includes the following columns:

- **ID**: A unique identifier for each scenario, which links directly to the generated class diagram.
- **familyId**: Identifies the general category (or "family") to which each scenario belongs, facilitating folder structure. The familyId follows the format: *{architecture}{fr_set_number}{nfr_set_number}{prompt_file_number}*

  For example, familyId = cs111 specifies a Client-Server architecture, FR set 1, NFR set 1, and prompt 1.
- **architecture**: Specifies the requested architecture pattern (e.g., Client-Server, Three-Tier).
- **frPathfile**: Path to the functional requirements (FR) set file.
- **nfrPathfile**: Path to the non-functional requirements (NFR) set file.
- **promptPathfile**: Path to the prompt text file.
- **modelMetadata**: Indicates the name and version of the selected LLM (e.g., llama3.1:latest, gemma2:27b).
- **source**: Information about whether the class diagram was generated via code (through Python scripts) or directly through online methods (e.g., querying models like ChatGPT and Gemini on the web).
- **ragDecision**: Specifies whether RAG is applied for the scenario (Yes or No).
- **ragPathfile**: Path to the RAG file containing external data for scenarios where RAG is used.

- **embeddingsMetadata**: JSON object containing details about the RAG process, formatted as follows:

```
{
  "agent": "openai/ollama",
  "model": "nomic-embed-text:latest / text-embedding
    -large-3",
  "chunking_info": {
    "method": "recursive/semantic",
    "chunk_size": "1000/-"
  }
}
```

- **resultPath**: Auto-populated by the script after response generation, specifying the path to the generated response text file.

By using this structured table, our Python scripts can iterate through each scenario, automating the process of generating class diagrams according to specified requirements and configurations. This setup not only facilitates a systematic exploration of each variable (e.g., architecture type, model selection, RAG use) but also ensures that results are organized, consistent, and easily traceable across all scenarios in the experiment.

3.2.4 Experiments Performed

To evaluate the impact of each parameter (architecture type, FR/NFR sets, model selection, RAG usage, and embedding strategy) on the quality of AI-generated class diagrams, we conducted a comprehensive set of experiments designed to measure performance under controlled conditions. We aimed to isolate the effects of each variable by creating and analyzing a variety of scenarios.

Our experimental design involved repeating the same set of scenarios across three distinct architectures: **Client-Server**, **Three-Tier**, and **Model-View-Controller (MVC)**. Each architecture set contained 160 unique scenarios, resulting in a total of 480 experiments. These scenarios were constructed by varying combinations of parameters, including:

- **Large Language Models (LLMs)**: Different models were used to assess how model choice influences diagram generation, including options like llama3.1, gemma2:27b, and command-r.
- **RAG vs. Non-RAG**: We tested scenarios both with and without Retrieval-Augmented Generation (RAG) to observe how supplementary information impacts the accuracy and structure of generated diagrams.
- **Functional and Non-Functional Requirement (FR/NFR) Sets**: We used multiple versions of FR and NFR sets to determine if changes in requirement detail level influence the models' outputs.
- **Embedding Models and Methods**: Different embedding models and chunking methods were incorporated in the RAG process to study how these factors affect retrieval and relevance in AI-generated outputs.

In the following diagram, we provide a detailed breakdown of the scenarios

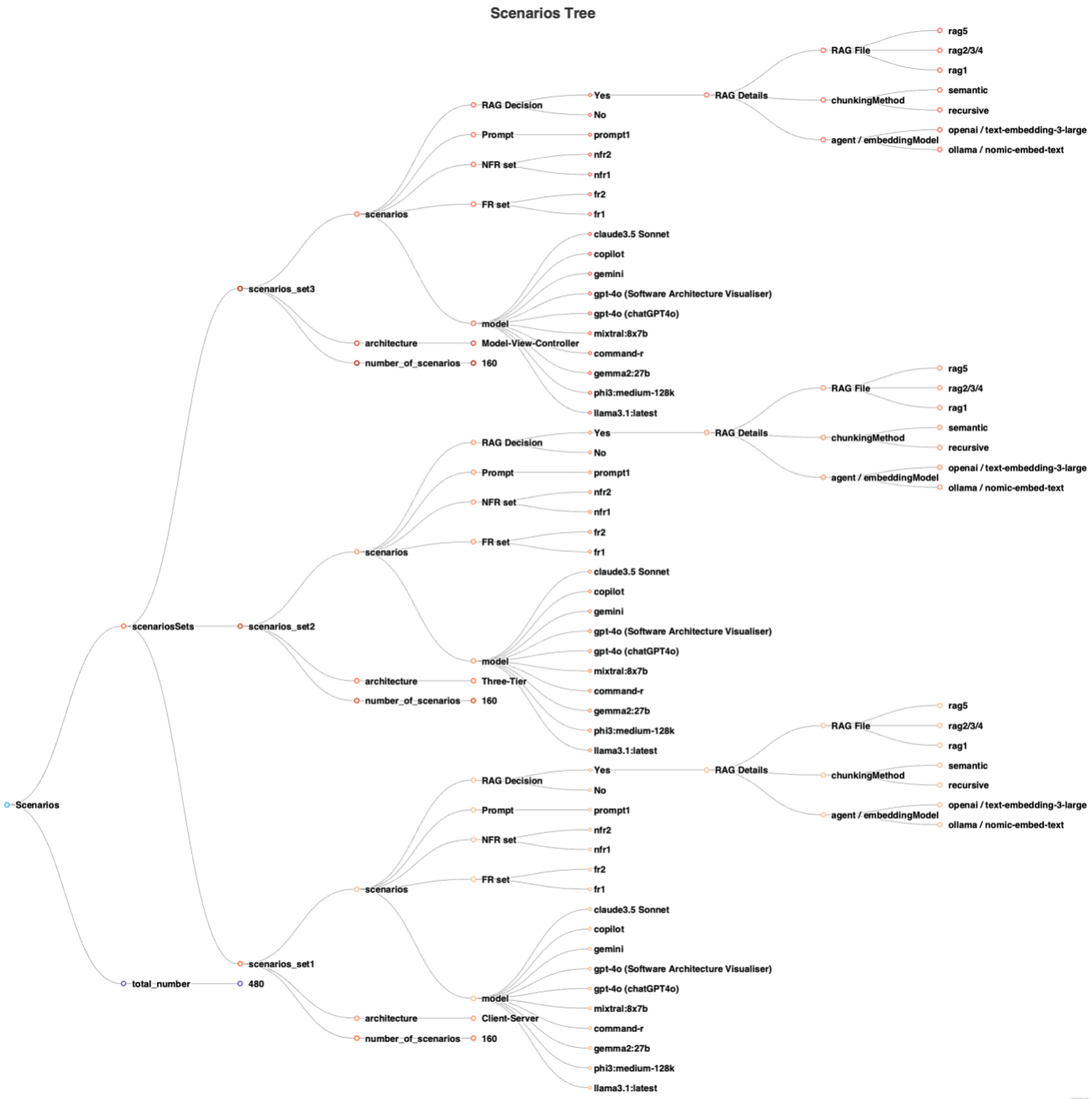


Figure 3.11 Scenarios graph

3.2.5 Assessment by AI and Human Experts

The final phase of our experiment involves a thorough evaluation of the generated class diagrams, conducted through two methods, human expert assessment and AI-based evaluation, across some evaluation keypoints.

Evaluation Keypoints

1. **Adherence to Architecture**
   Experts evaluated how well the requested software architecture was implemented in each class diagram. This criterion focused on whether the architectural principles were respected, including

the appropriate distribution of responsibilities among classes and alignment with the intended architectural pattern.

2. **Correctness of Class Relationships**
   This dimension focused on the accuracy of relationships between classes within the context of the specified architecture. Experts assessed whether associations, dependencies, and communication flows between classes were properly represented and if they adhered to architectural principles.
3. **Cohesion and Coupling**
   High cohesion and low coupling are essential design principles for maintainable and effective architectures. Experts rated each diagram based on the focus and singularity of purpose within classes (high cohesion) and the minimization of inter-class dependencies (low coupling).
4. **Consistency with Software Requirements**
   Experts assessed whether each diagram met both functional and non-functional requirements as specified in the provided requirements sets. This evaluation ensured that the diagrams not only adhered to architectural principles but also accurately represented the required functionality.

Human Expert Evaluation

We assembled a panel of four individuals with varying levels of experience in software architecture to assess the generated diagrams. Each expert independently reviewed and rated the diagrams on the four evaluation criteria, as described earlier:

Each expert provided an integer score from 0 to 5 for each dimension. This provided a rich qualitative and quantitative dataset for analyzing diagram quality.

AI-Based Evaluation

To complement human assessments, we also leveraged LLMs to evaluate the class diagrams. Each LLM was presented with the generated diagram and the specified architecture and was prompted to evaluate the diagrams based on the same four criteria as the human experts. The LLMs were tasked with providing:

- **Integer Scores**: Similar to human evaluators, LLMs assigned a score from 0 to 5 for each criterion, allowing for a direct comparison of AI and human ratings.
- **Textual Reasoning**: For each score, LLMs provided a textual rationale for their rating. This was essential for gaining insights into the LLMs' reasoning processes, helping us understand how well the models recognize architectural principles, identify errors, and appreciate best practices in class diagram design.

The combination of human and AI evaluations allows us to perform a comprehensive analysis of each diagram's quality. By comparing the consistency and alignment of AI-generated scores and justifications with those of human experts, we aim to identify potential strengths and weaknesses in LLM-based evaluation. This dual-assessment approach provides a robust framework for assessing the effectiveness of AI in generating and evaluating architectural diagrams, giving us a comprehensive understanding of its current capabilities and areas for future improvement.

4 Execution and results

4.1 Typical Cases

In this section we will present some UML diagrams generated by the LLMs.

Complete and accurate UML class diagrams

The following class diagrams showcase some interesting and commendable approaches to capturing the architectural patterns. All of these diagrams successfully adhere to the core principles of the three specified architectures. Additionally, they align well with the software requirements, accurately represent class relationships, and demonstrate strong cohesion and low coupling, reflecting a solid performance across key evaluation criteria.

Client-Sever:

Llama3.1:8B | fr1 | nfr1 | rag1 | ollama | nomic-ebed-text:latest | semantic (ID = 3)

The diagram in Figure 4.1 features three distinct packages: **client**, **server**, and **domain**, each with clearly defined roles. The packages are interconnected through detailed class relationships, demonstrating effective communication between components. The diagram successfully incorporates nearly all the software requirements, including specific details about the TCP/IP connection. On the server side, it handles database responsibilities, while the client side includes classes such as DCCApplication with a run() method, a Converter class, and a Manager class equipped with CRUD functions for server interactions.

Overall, this diagram is comprehensive and meets all criteria, making it a robust and accurate representation of the client-server architecture.

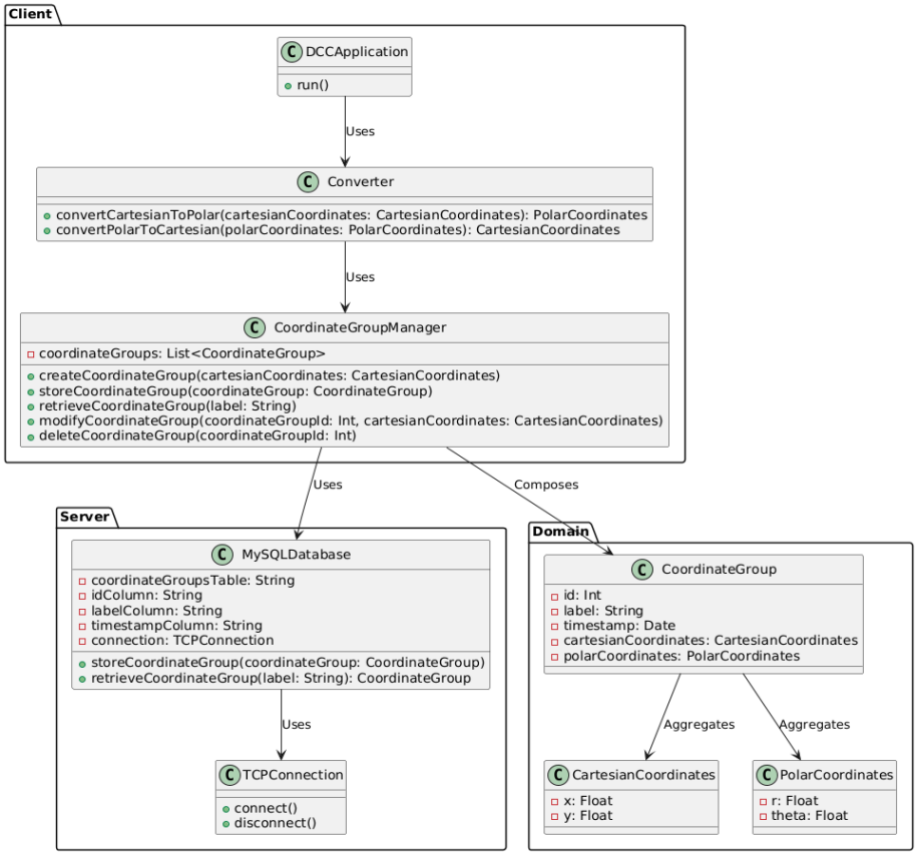


Figure 4.1. Diagram with ID = 3

gemma2:27B | fr1 | nfr1 | rag5 | ollama | nomic-ebed-text:latest | semantic (ID = 168)

The diagram in Figure 4.2 offers a similar and equally correct approach to the client-server architecture as the one described above. It defines distinct **client** and **server** packages, each containing the appropriate classes with sufficient attributes. On the client side, the diagram effectively incorporates both the UI and the application logic, while the server side is responsible for managing the database.

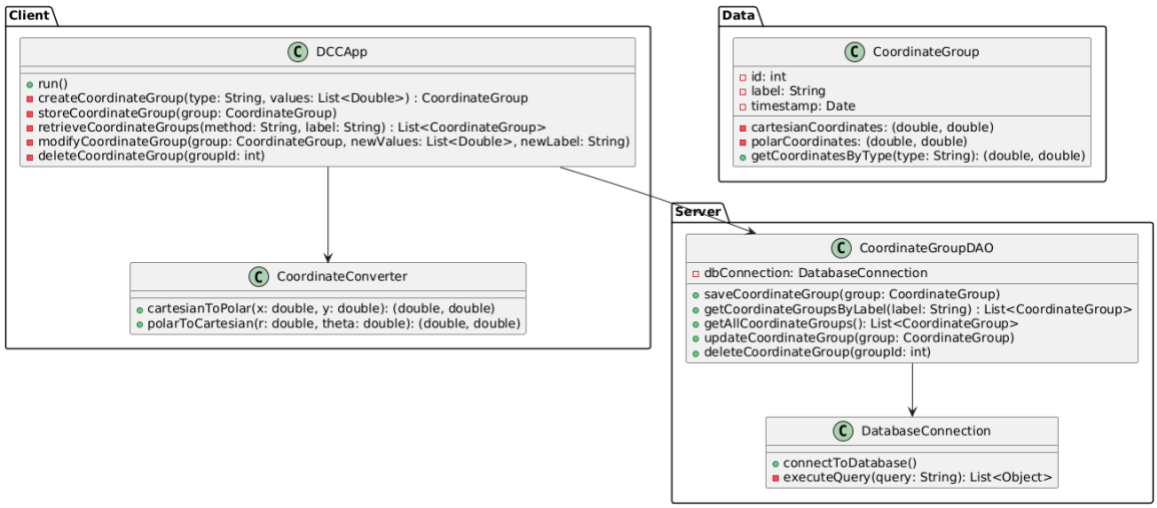


Figure 4.2 Diagram with ID = 168

command-r | fr1 | nfr1 | rag5 | openai | text-embedding-3-large | semantic (ID = 185)

The diagram in Figure 4.3 showcases another commendable approach to the client-server architecture. It consists of two primary packages: **client** and **server**. On the server side, it includes classes dedicated to database management and database connection. The client side features separate classes for **PolarConverter** and **CartesianConverter**, along with a comprehensive Manager class and a DCCApplication class. This diagram adheres to the core principles of the client-server model while demonstrating excellent cohesion within components and appropriate levels of coupling between them.

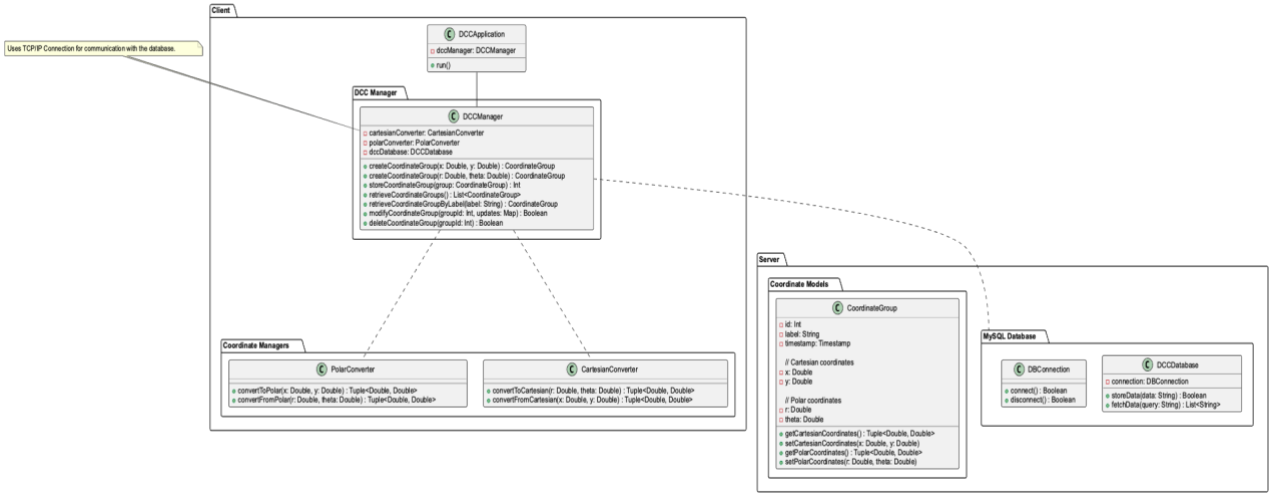


Figure 4.3 Diagram with ID = 185

Three-Tier:

gemma2:27B | fr1 | nfr1 | rag1 | ollama | nomic-ebed-text:latest | semantic (ID = 23)

The diagram in Figure 4.4 demonstrates adherence to the core principles of the Three-Tier architecture. It clearly exhibits a separation of concerns across the layers, not only through the use of distinct packages but also in the allocation of functionality to the appropriate layers. Notably, the coordinate conversion task, which represents the main business logic of the software, is included and correctly placed within the appropriate layer.

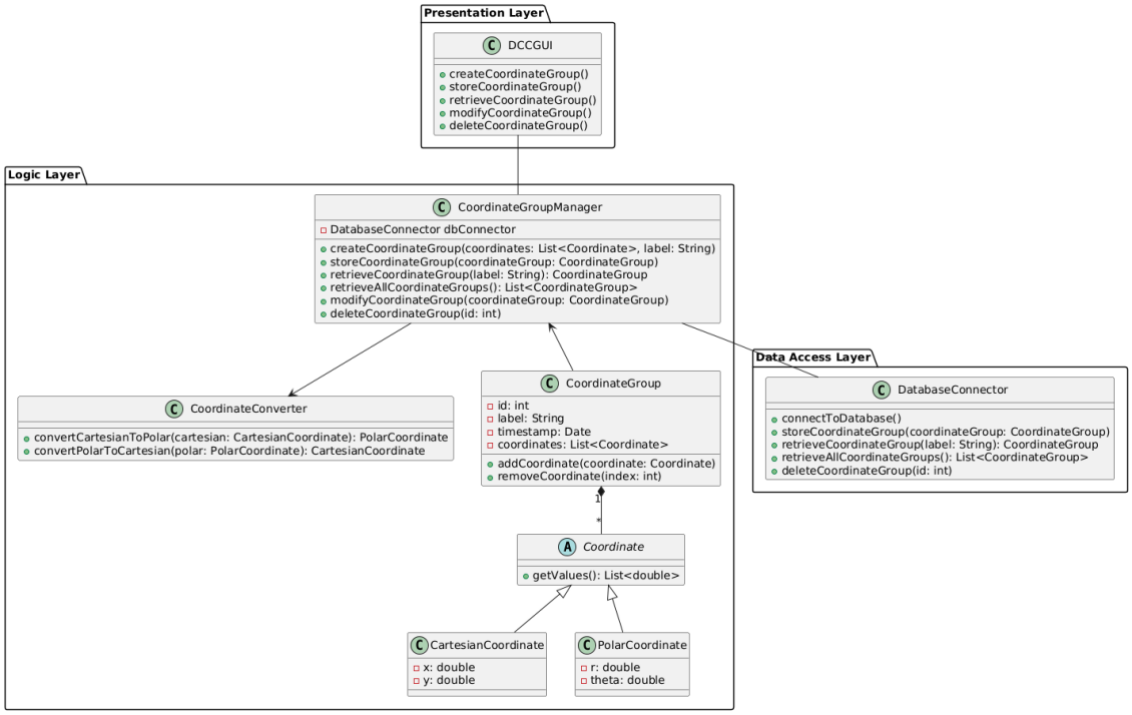


Figure 4.4 Diagram with ID = 23

Llama3.1:8B | fr1 | nfr1 | rag3 | openai | text-embedding-3-large | semantic (ID = 85)

The diagram in Figure 4.5 not only aligns with the requested architecture and its principles but also features well-detailed classes with numerous, accurate attributes. Each layer and its corresponding classes are designed with a comprehensive and thoughtful approach, reflecting a high level of adherence to architectural standards.

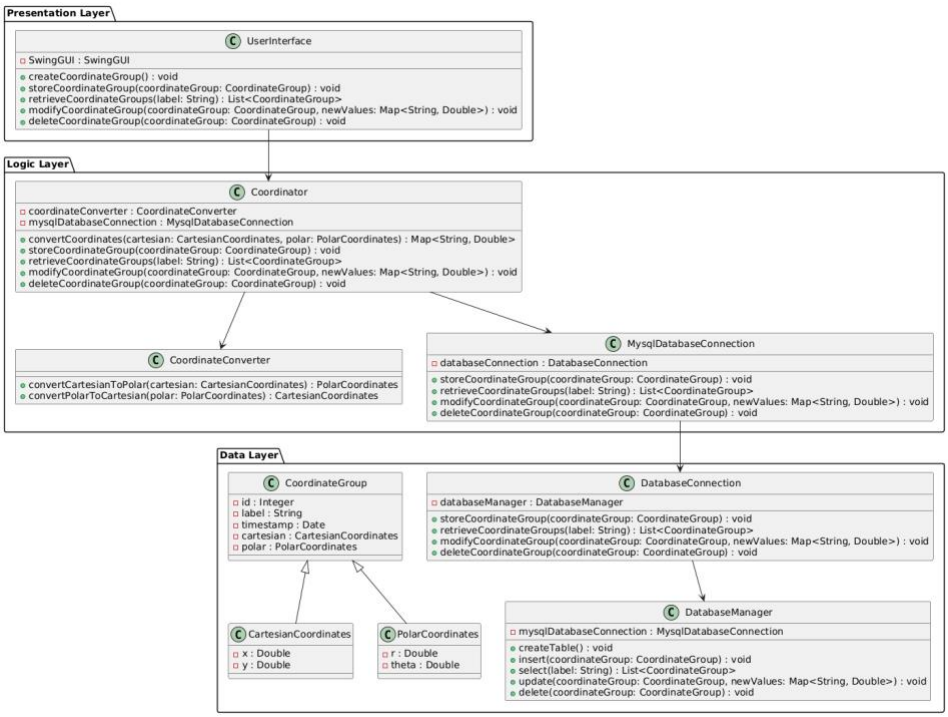


Figure 4.5 Diagram with ID = 85

gpt4o | fr2 | nfr2 | No RAG (ID = 467)

Similar to the previous example, diagram in Figure 4.6 aligns well with the Three-Tier architecture and its principles. It includes clearly defined packages and classes, each with a commendable level of detail.

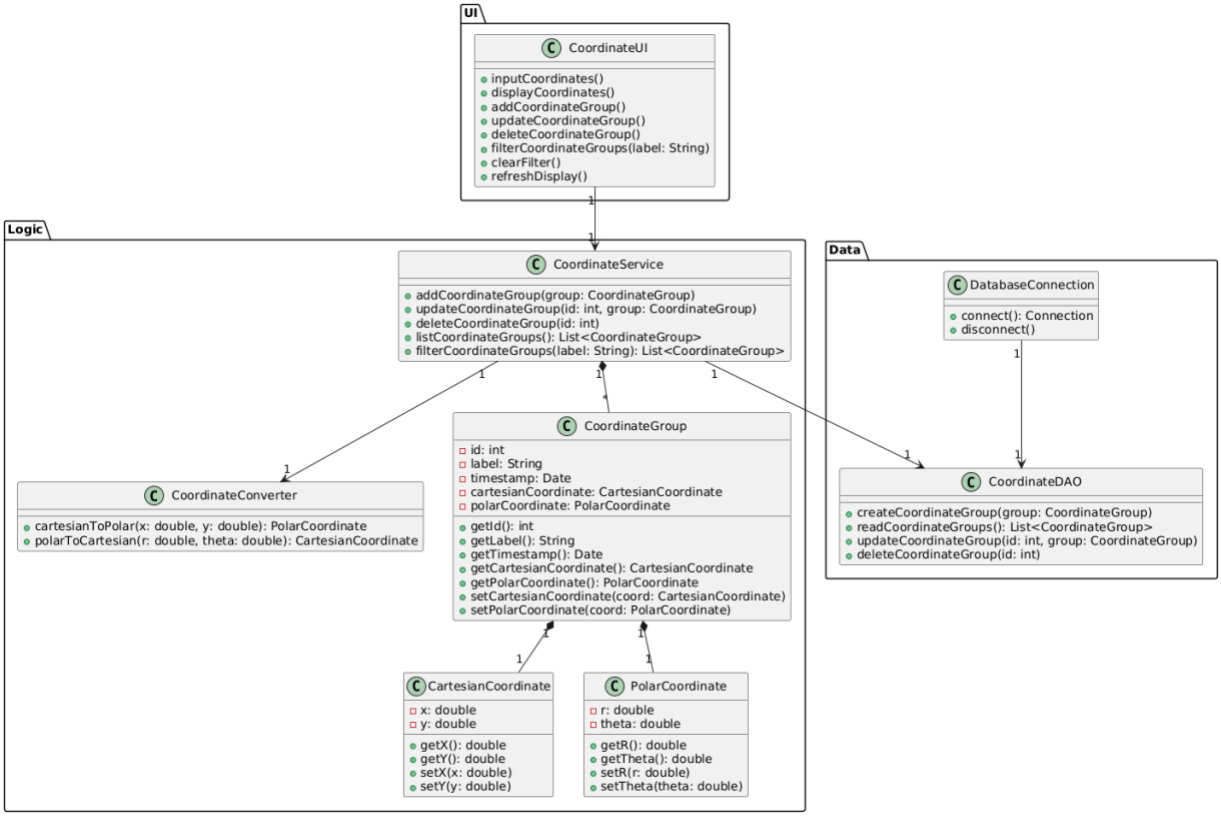


Figure 4.6 Diagram with ID = 467

Model - View – Controller:

The following three diagrams demonstrate significant alignment with the MVC architectural pattern. Each diagram includes the three key packages: **Model**, **View**, and **Controller**, properly organized to reflect the architecture's principles. Moreover, all three diagrams incorporate the business logic for the conversion task, correctly assigning it to the **Model** package as per architectural best practices.

gemma2:27B | fr1 | nfr1 | rag1 | openai | text-embedding-3-large | semantic (ID = 30)

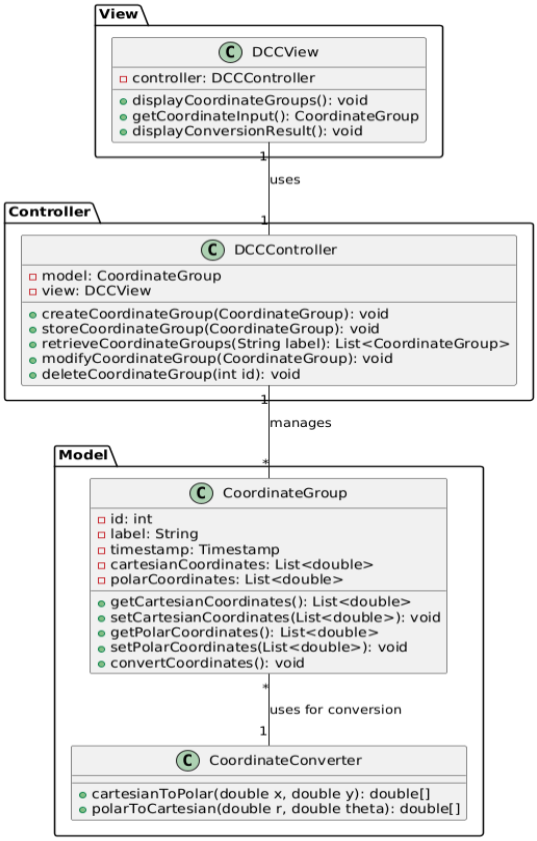


Figure 4.7 Diagram with ID = 30

gpt4oSAV | fr1 | nfr1 | No RAG (ID = 231)

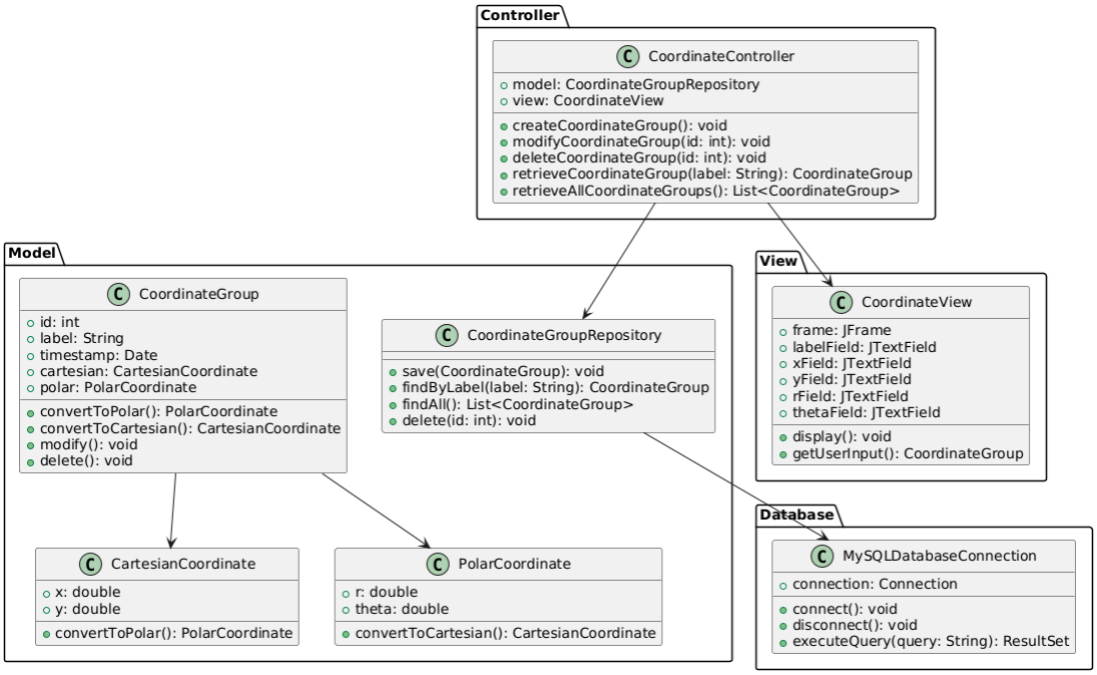


Figure 4.8 Diagram with ID = 231

gemma2:27B | fr2 | nfr2 | rag4 | ollama | nomic-embed-text:latest |semantic (ID = 342)

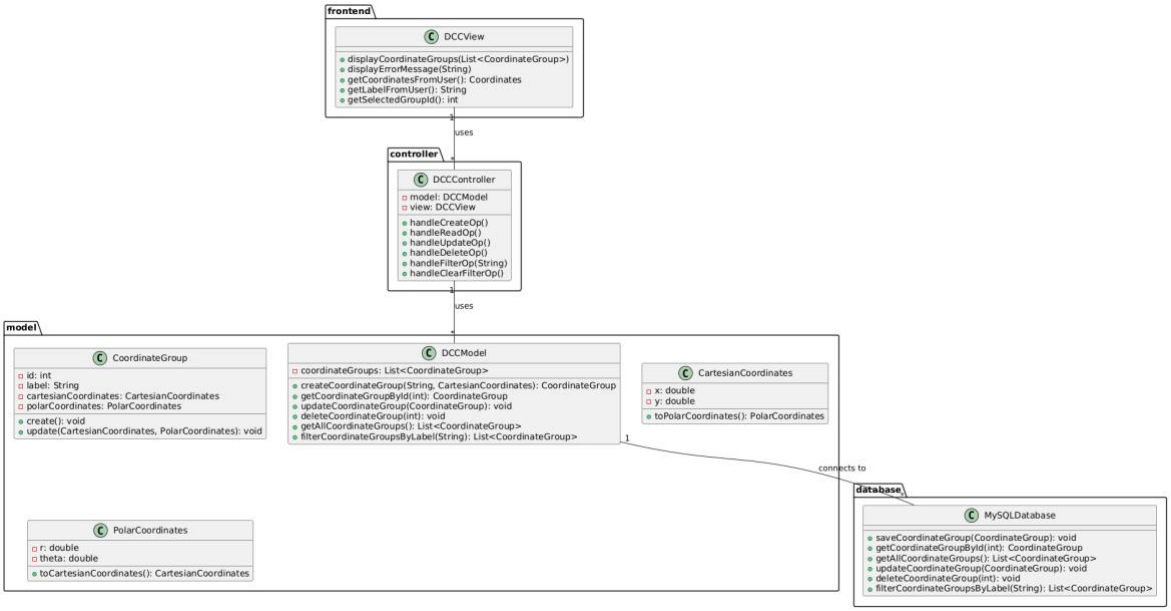


Figure 4.9 Diagram with ID = 342

Incomplete and unstructured UML diagrams

Not all diagrams achieved the high standards of the previously showcased examples. While some diagrams received average evaluation scores, partially succeeding in the keypoints, others fall significantly short. The following diagrams represent particularly poor results, characterized by a lack of class relationships, packages, attributes, and any adherence to the requested architectural pattern. This section presents a selection of such UML diagrams. While these diagrams demonstrate some level of structural coherence, they are generally incomplete and they fail to fully capture the intended architecture and functionality of the system. Below, we provide a more detailed analysis of these diagrams:

Client-Server:

Llama3.1:8B | fr1 | nfr1 | rag1 | openai | text-embedding-3-large | recursive (ID = 4)

The diagram in Figure 4.10 attempts to adhere to the client-server architecture by separating the frontend and backend into distinct packages. However, it falls short in several key aspects. First, the diagram fails to illustrate the client's role within this architecture. Within the frontend package, the Coordinate and DummyCoordinator classes are present, but neither represents the UI component of the client. Additionally, the diagram depicts only a single relationship between the two packages, which is insufficient to capture the complexity of their interactions. Finally, a critical issue is the absence of any representation of the software's conversion functionality, either as a function or as a standalone class.

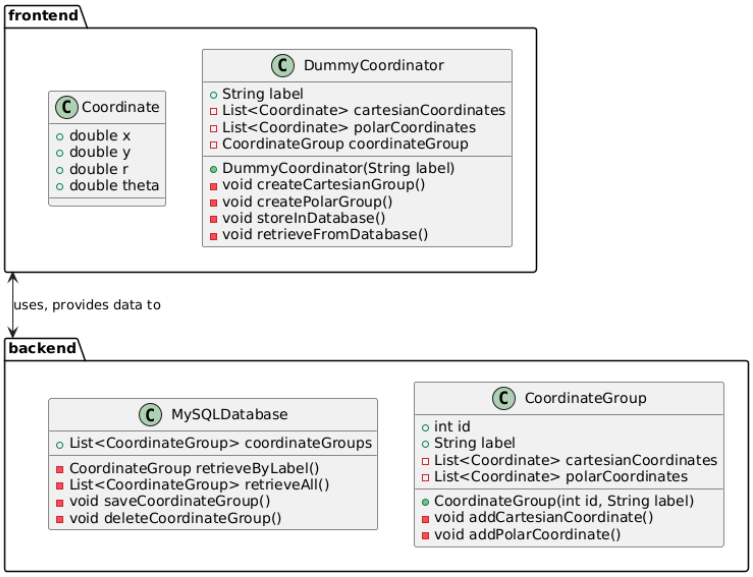


Figure 4.10 Diagram with ID = 4

Llama3.1:8B | fr1 | nfr1 | rag5 | ollama | nomic-embed-text:latest | recursive (ID = 152)

The diagram in Figure 4.11 exhibits confusion regarding the architectural patterns. Although the requested architecture was Client-Server, the generated structure aligns more closely with a Three-Tier architecture. Additionally, the diagram fails to represent a key business logic function of the system—the coordinate conversion task—further detracting from its accuracy and completeness.

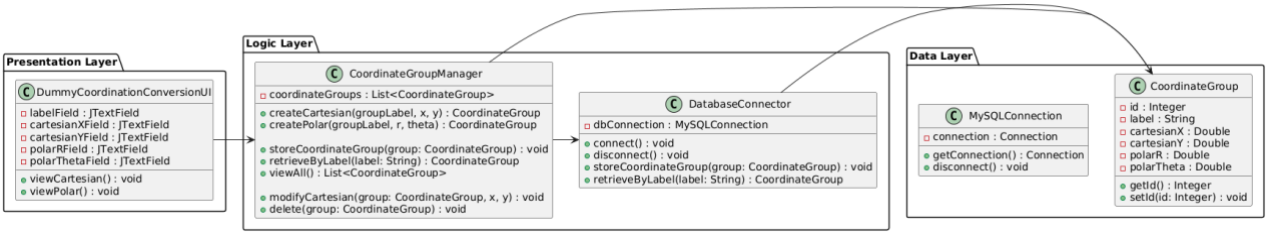


Figure 4.11 Diagram with ID = 152

Three-Tier:

Command-r | fr1 | nfr1 | rag1 | openai | text-embedding-3-large | recursive (ID = 39)

The diagram in Figure 4.12 once again demonstrates confusion regarding architectural patterns. While the requested architecture was Three-Tier, the generated structure appears to follow the principles of a Client-Server architecture instead. Additionally, the diagram lacks any class relationships and completely omits the essential business logic related to the coordinate conversion task.

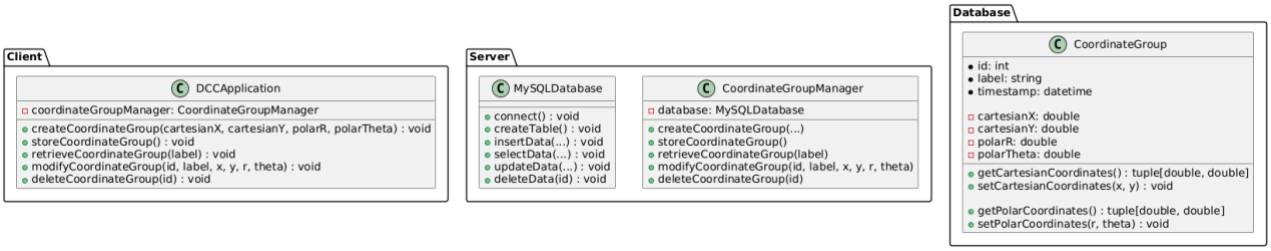


Figure 4.12 Diagram with ID = 39

Gemma2:27B | fr2 | nfr2 | rag3 | openai | text-embedding-3-large | semantic (ID = 340)

The diagram in Figure 4.13 fails to adhere to any fundamental principles of the Three-Tier architecture. Moreover, most of the classes, with the exception of Model, lack attributes apart from a generic description field. Overall, this diagram is poorly constructed and inadequate for representing the application software in any meaningful way.

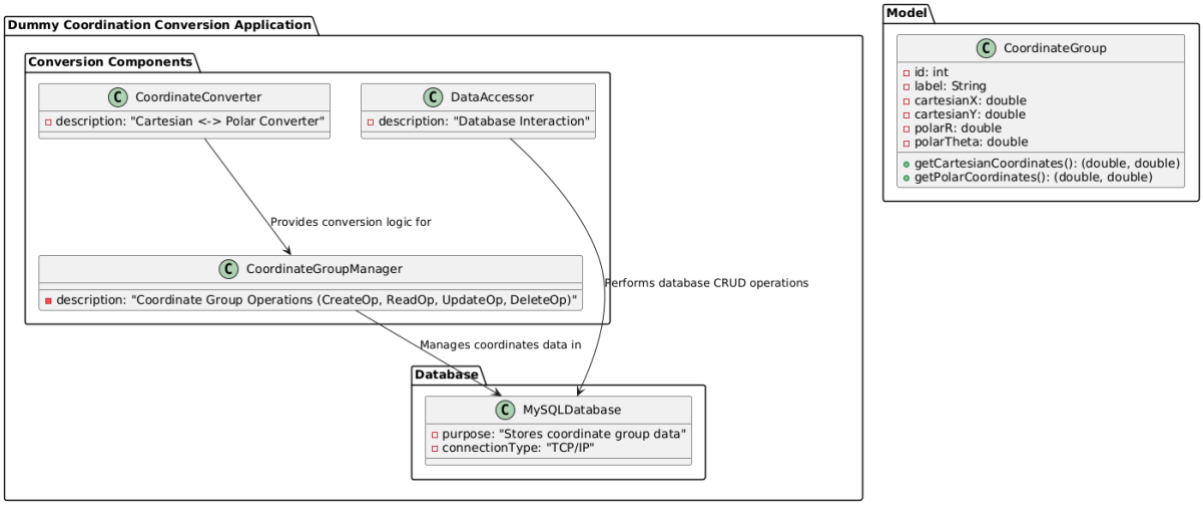


Figure 4.13 Diagram with ID = 340

Model -View – Controller:

Llama3.1:8B | fr2 | nfr2 | rag1 | ollama | nomic-embed-text:latest | recursive (ID = 252)

The diagram in Figure 4.14 attempts to follow some MVC practices, as evidenced by the inclusion of Controllers and Models packages, with relatively appropriate classes within them. However, it lacks a View package or class, and it also fails to represent the critical business logic function for the coordinate conversion task.

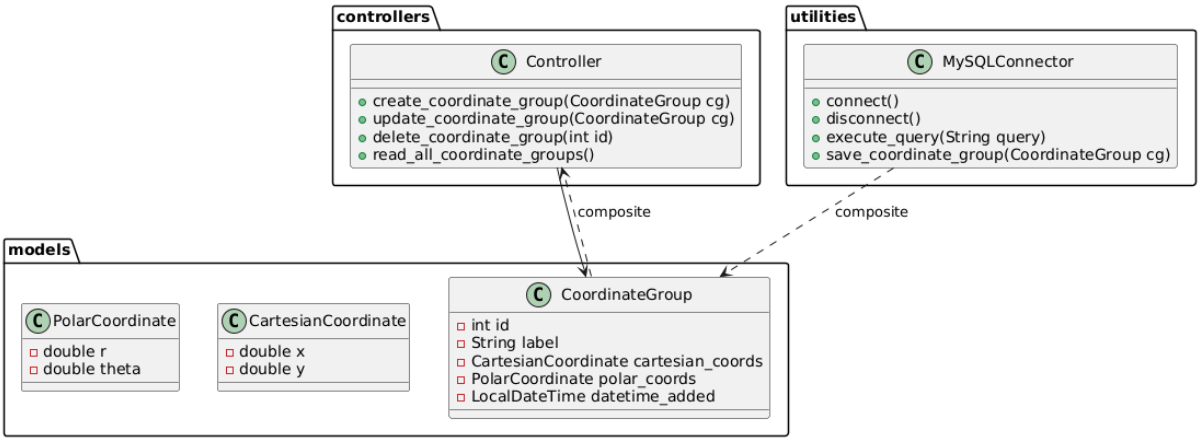


Figure 4.14 Diagram with ID = 252

Command-r | fr2 | nfr2 | rag5 | ollama | nomic-embed-text:latest | recursive (ID = 432)

The following diagram in Figure 4.15 once again demonstrates a misunderstanding of the requested architecture. While the requested pattern was Model-View-Controller (MVC), these diagrams align more closely with a Client-Server approach. Despite this significant issue, the diagrams do exhibit alignment with the functional software requirements.

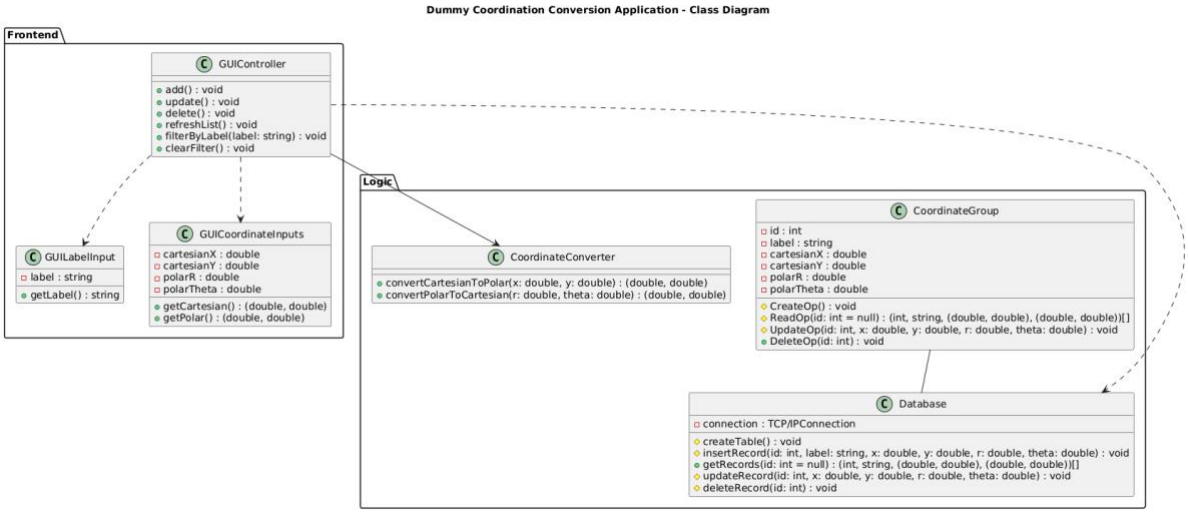


Figure 4.15 Diagram with ID = 432

Weak diagrams

There are also diagrams that can only be described as complete failures across almost all key evaluation criteria. These diagrams display significant confusion regarding architecture, class relationships, and software requirements. The issues may stem from the limitations of the specific model, shortcomings in the RAG process, or a combination of both—such as the model's inability to effectively manage and leverage the RAG process when faced with a complex and extensive prompt. Below, we present an example of such a case:

Phi3:medium-128k | fr1 | nfr1 | rag5 | openai | text-embedding-3-large | recursive (ID = 214)

The requested architecture for diagram in Figure 4.16 was Client-Server; however, it fails to adhere to any architectural pattern. Notably, the classes Storefront and OrderProcessor appear to be taken from an example UML class diagram included in the RAG file. This suggests that the diagram was significantly influenced, and ultimately confused, by the RAG process.

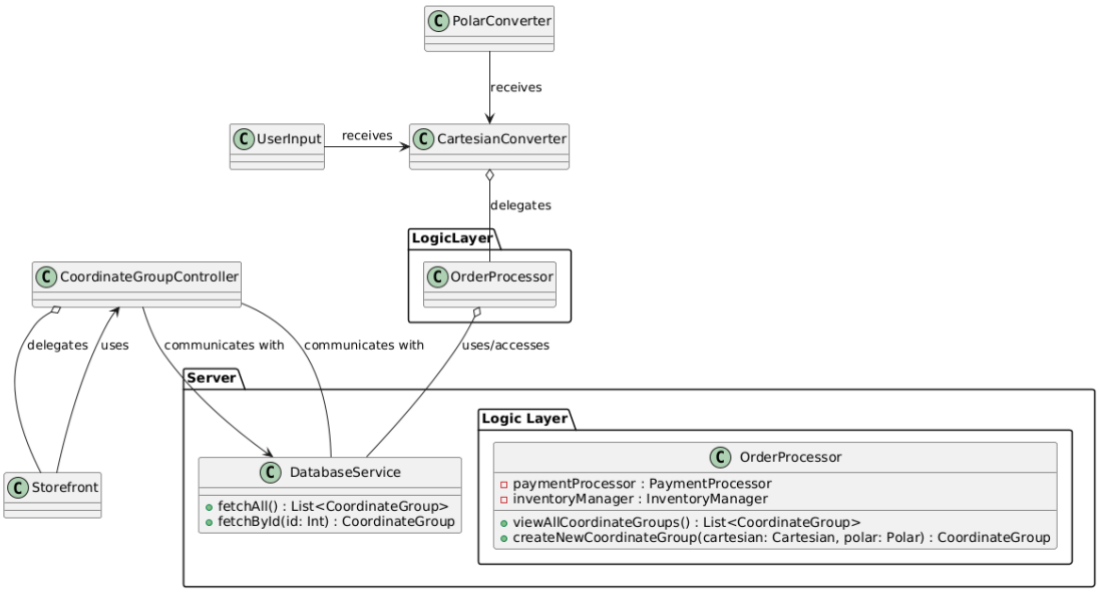


Figure 4.16 Diagram with ID = 214

Mixtral:8x7B | fr1 | nfr1 | rag1 | ollama | nomic-embed-text:latest |semantic (ID = 48)

The diagram in Figure 4.17 includes some classes, such as CoordinateGroup and CoordinateConverter, which represent certain software requirements. However, the diagram totally fails to adhere to any architectural pattern. Additionally, it lacks the use of packages, and overall, it represents an absolutely poorly generated result.

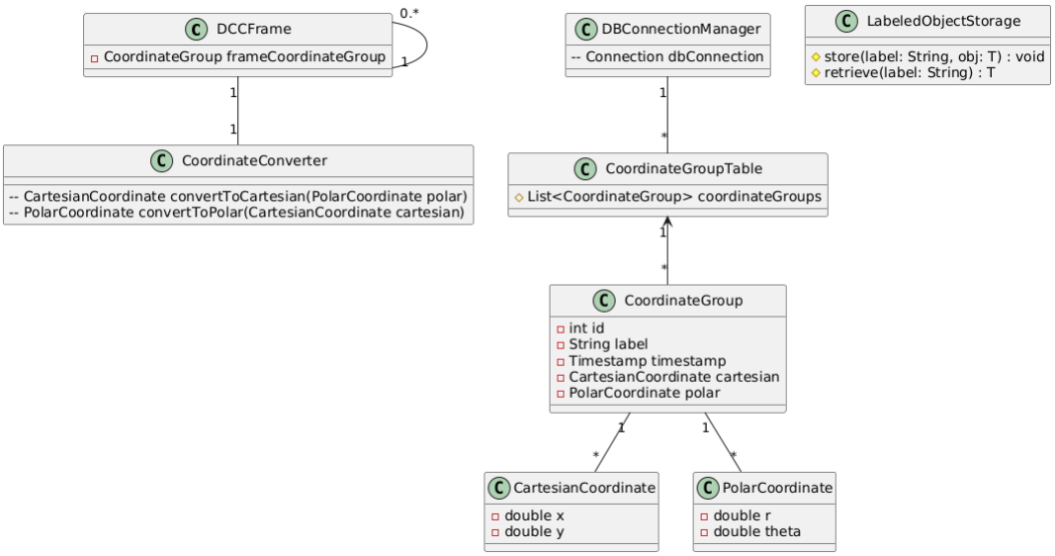


Figure 4.17 Diagram with ID = 48

Impact of fr2 set on diagram structure

At this point, we would like to present an interesting observation made during the exploration and evaluation of the generated diagrams, specifically related to the structure of the **fr2** set. As described in section 3.1.2 , the **fr2** set is organized into segments: **Logic**, **User Interaction**, and **Data Operations**, with different requirements detailed under each segment. This structure has influenced the resulting class diagrams, leading many of them to align with the **Three-Tier Architecture** pattern. Regardless of the requested architecture—whether client-server, Model-View-Controller, or others—the class diagrams often exhibit a structure reminiscent of the three-tier model. Notably, many diagrams generated using the **fr2** set include the packages **Frontend**, **Logic**, and **Data**, with relationships that closely mirror the three-tier architecture. To illustrate this observation, we will present specific diagram examples. Later, in the data results section, we will substantiate this assertion with supporting charts and data analysis.

The following two diagrams illustrate the influence of the **fr2** set, as previously described. They feature three distinct packages: one representing the **UI/Frontend**, another dedicated to the **Logic Component** containing the business logic of the software, and a third focused on the **Database**.

This structure evokes the principles of the **Three-Tier Architecture**, rather than adhering to the requested architectural pattern: Client-Server for the diagram in Figure 4.18 and Model-View-Controller for diagram in Figure 4.19

Gemma2:27B | fr2 | nfr2 | rag1 | openai | text-embedding-3-large | semantic (ID = 260)

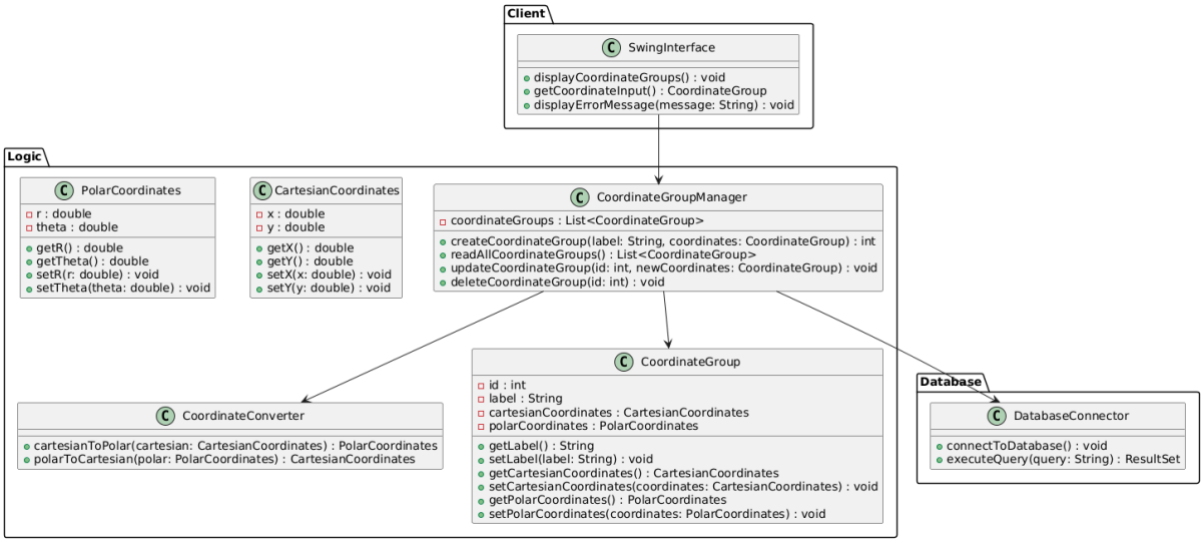


Figure 4.18 Diagram with ID = 260

Mixtral:8x7B | fr2 | nfr2 | No rag (ID = 296)

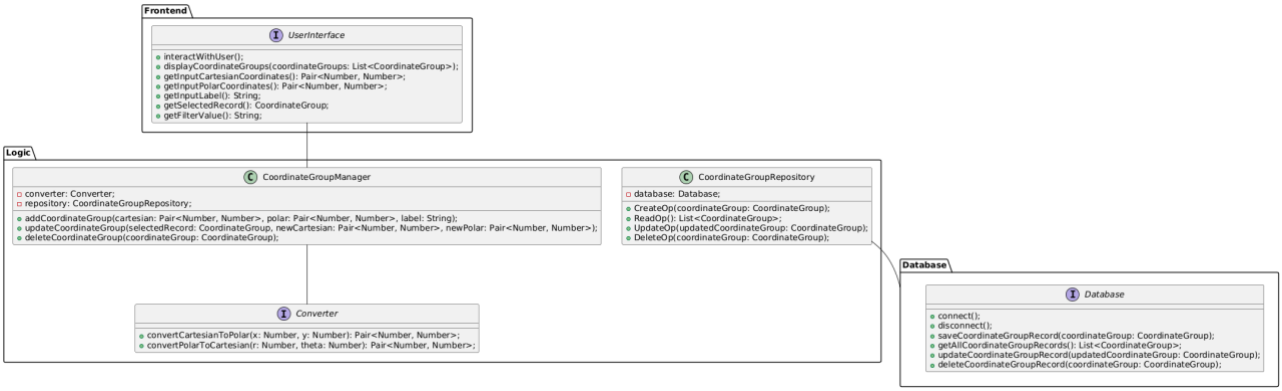


Figure 4.19 Diagram with ID = 296

Command-r | fr2 | nfr2 | rag2 | openai | text-embedding-3-large | recursive (ID = 349)

The following diagram was designed to implement the requested Client-Server Architecture. However, the influence of the fr2 set is evident, as the diagram is organized into three packages: Frontend, Logic, and Database. What sets this diagram apart from others influenced by the fr2 set is its explicit definition of client and server roles. Specifically, the Frontend package is marked with the stereotype <<Client>>, while the Logic and Database packages are labeled with the stereotype <<Server>>. Despite its three-package structure and relationships, which are commonly associated with the **Three-Tier Architecture**, this diagram demonstrates an understanding of and adherence to the **Client-Server** pattern.

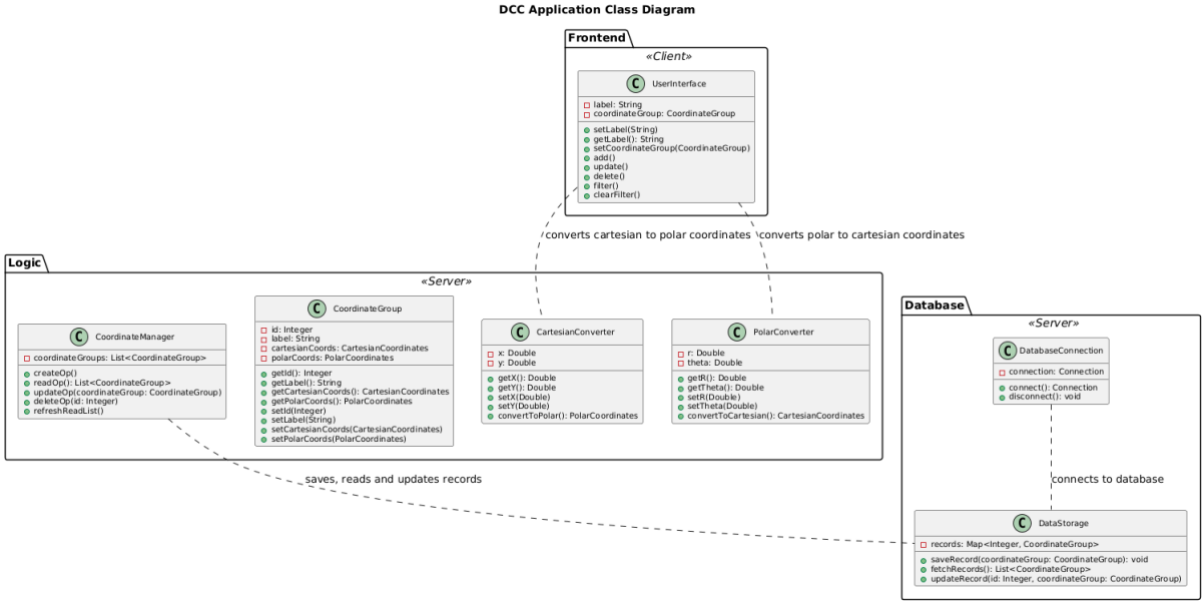


Figure 4.20 Diagram with ID = 349

4.2 Web-based presentation of the results

As detailed in the previous chapter, we used large language models (LLMs) to generate a total of **480 class diagrams** in PlantUML format. To streamline the evaluation process and provide a foundation for extending this project in the future, we developed a custom software application. This

tool was designed to present the generated diagrams in an organized and user-friendly interface while also offering functionalities to facilitate their evaluation by humans.

The application includes several core functionalities to support both diagram visualization and evaluation:

1. **Organized Display of Diagrams**
   The tool provides a clean, intuitive layout to browse through all 480 generated class diagrams. Diagrams are categorized and searchable by key parameters, such as architecture type (Client-Server, Three-Tier, Model-View-Controller), the selected LLM, the diagrams's unique identifier, or the used rag file. This organization allows users to quickly locate and review specific diagrams.
2. **Integrated Diagram Rendering**
   Using the PlantUML .jar file, the tool dynamically renders PlantUML code into class diagrams. This enables users to view the diagrams directly within the application without requiring additional software or manual conversion.
3. **Evaluation Interface**
   The tool includes an evaluation module that mirrors the criteria used in the experiment (adherence to architecture, class relationships, cohesion/coupling, and requirement consistency). Evaluators can input their scores for each diagram, storing the results in a a database for later analysis. After completing their evaluation, users can also view the scores assigned by each LLM for every key evaluation criterion.
4. **Comparison and Filtering Features**
   Users can compare diagrams generated under different scenarios (e.g., with and without RAG, or between different LLMs) side-by-side. Filtering options allow users to narrow down diagrams based on specific configurations, such as embedding models or chunking methods.
5. **Export and Reporting**
   The application supports exporting evaluation data, making it easy to summarize findings for presentations or further analysis.
6. **Extensibility for Future Work**
   The tool was built with scalability in mind. Its modular design allows for easy integration of additional features, such as new architectures, evaluation criteria, or analysis modules.

App Walkthrough

In the following section, we provide screenshots of the tool in action, demonstrating its major functionalities and layout. These visuals will illustrate how the application simplifies diagram management and evaluation, making it an indispensable resource for this project.

Main page : Evaluate diagrams

In the screenshots below, you can see the interfaces of our custom app designed to display, evaluate, and analyze the generated class diagrams. The main page consists of four main components, each serving a specific function:

1. **Filtering Component**

This section allows users or evaluators to filter the displayed results based on various parameters. Filters include options such as:

- **Model:** Select the LLM used for generating the diagram (e.g., Llama 3.1, Gemma 2:27b).
- **Architecture:** Choose the architectural pattern, such as Client-Server, Three-Tier, or MVC.
- **RAG Decision:** Filter diagrams based on whether RAG was applied.
- **Additional Parameters:** Filters for FR/NFR sets, prompts, embeddings, and chunking strategies.

This component ensures users can quickly narrow down results to focus on the specific diagrams or scenarios of interest.

2. Results Segment

This component displays a table of filtered results, reflecting the parameters selected in the filtering section. Each row in the table represents a diagram scenario, including details such as architecture, LLM model, RAG decision, and result path. Users can select a row to preview and evaluate the corresponding diagram in the component below. This streamlined integration ensures an intuitive workflow for diagram selection and analysis.

3. **Evaluation Form**

This section focuses on reviewing and evaluating the selected diagram. It consists of several interactive elements:

- **Diagram Carousel:** Displays the selected diagram and includes navigation arrows to allow users to browse through other diagrams in the dataset seamlessly.
- **Evaluation Sliders:** Below the diagram, evaluators can rate the diagram across the key criteria discussed earlier, including:
  - Adherence to architecture.
  - Correctness of class relationships.
  - Cohesion and coupling.
  - Consistency with software requirements.
- **Action Buttons:** Additional functionality is available through action buttons, which include:
  - **Edit Diagram:** Allows users to make modifications to the diagram.
  - **Compare Diagrams:** Provides a side-by-side view of two diagrams for comparative evaluation.
  - **Full-Screen Mode:** Enables a distraction-free, enlarged view of the diagram.
  - **Sidebar Navigation:** Opens a sidebar displaying a thumbnail preview of all diagrams, allowing users to navigate directly to any diagram of interest.
  - **Display Bar Chart**: If a registered user has already completed the assessment of diagram, this button enables the display of a bar chart. The chart provides a visual representation of the scores, facilitating the comparison and analysis of the results for the specific chart across different LLMs and the user's score.

4. Metadata Component

This section provides detailed metadata for the selected diagram. It includes:

- **Functional and Non-Functional Requirements (FR/NFR):** Displays the textual content of the requirements sets used as input for generating the diagram.
- **Prompt:** Shows the exact prompt text provided to the LLM for diagram generation.
- **Original LLM Response:** Displays the unprocessed response generated by the LLM.
- **RAG File (if applicable):** Indicates whether RAG was used and, if so, includes the content of the RAG file that supplemented the LLM's output.

This component ensures transparency and traceability, giving evaluators full context for each diagram. This interface was carefully designed to enhance usability and ensure that evaluators can interact with the diagrams efficiently

Second Page: LLM Assessments

The second screenshot showcases the **LLM Assessments** page of the app, dedicated to displaying the evaluations provided by large language models (LLMs). This page retains several key components from the previous interface, such as the **filtering**, **results**, and **metadata** sections, but introduces new functionality to present the LLMs' assessment details.

LLM Evaluation Details

Replacing the evaluation form from the previous page, this section displays the LLM's assessment for the selected scenario. The evaluation includes:

1. **Scores**: Numerical ratings (on a scale from 0 to 5) for each evaluation criterion:
    - Adherence to Architecture
    - Correctness of Class Relationships
    - Cohesion and Coupling
    - Consistency with Software Requirements
2. **Reasoning**: A detailed textual explanation for each score, providing insights into the LLM's evaluation logic. These justifications help users understand how the model interprets architectural principles and requirements, highlighting strengths and potential issues in the class diagrams. In addition to scoring, LLMs provide actionable feedback on how the class diagram could be improved.

This page provides a transparent view of the LLMs' capabilities in evaluating class diagrams. By displaying both quantitative scores and qualitative reasoning, it enables a direct comparison between human and AI evaluations.

This interface not only streamlines the analysis of LLM evaluations but also facilitates a deeper understanding of how different models interpret and assess architectural diagrams, making it a critical component of the overall app workflow.

Visualization Page

The app also features a third page dedicated to visualizing statistical analyses of the evaluation data through charts. These charts provide insights into patterns and trends in the evaluation results. We will discuss these charts in detail and present them later in the results section.

Figure 4.21 and Figure 4.22 show the two main pages of our app.

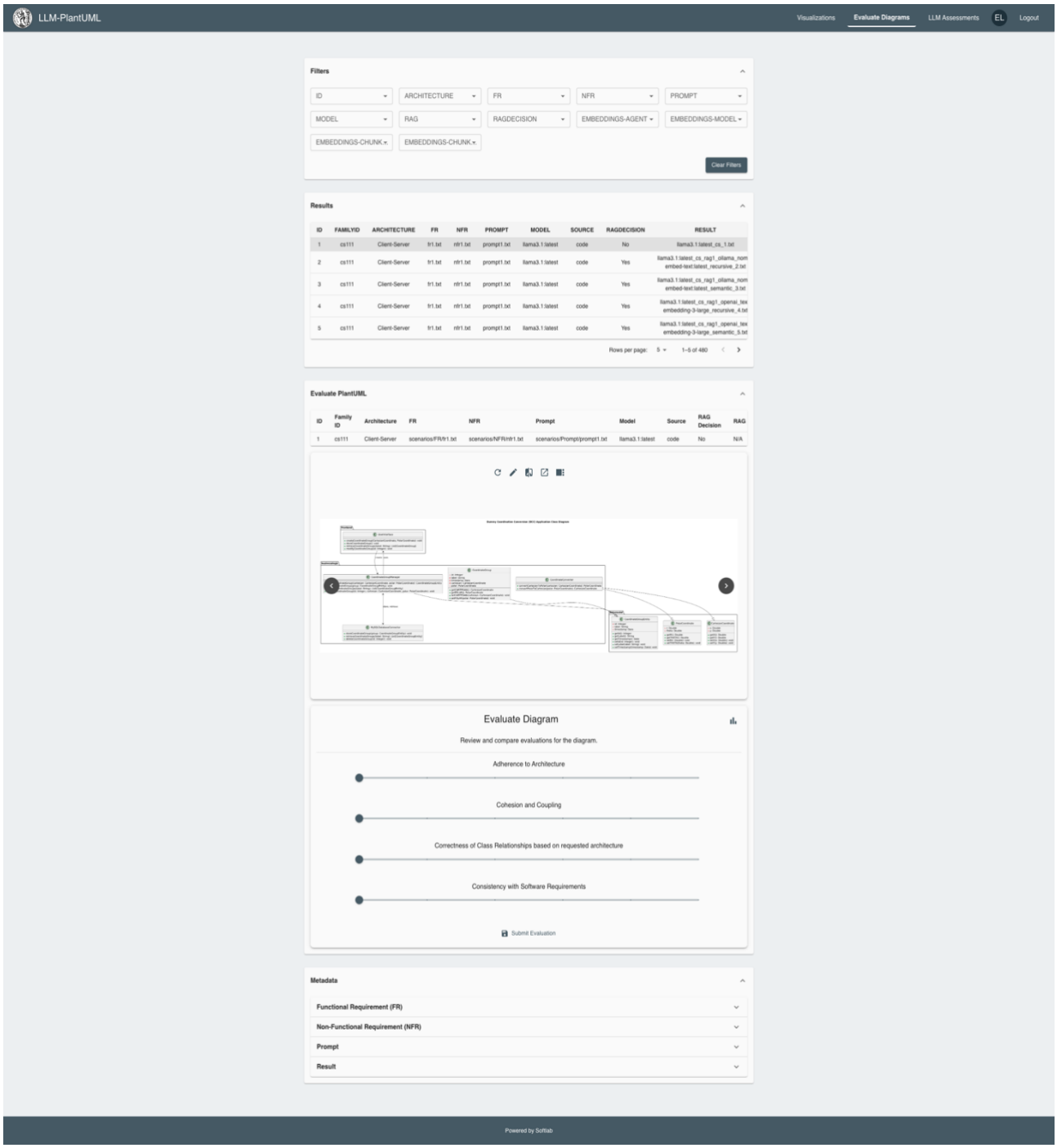


Figure 4.21 Evaluate diagrams page

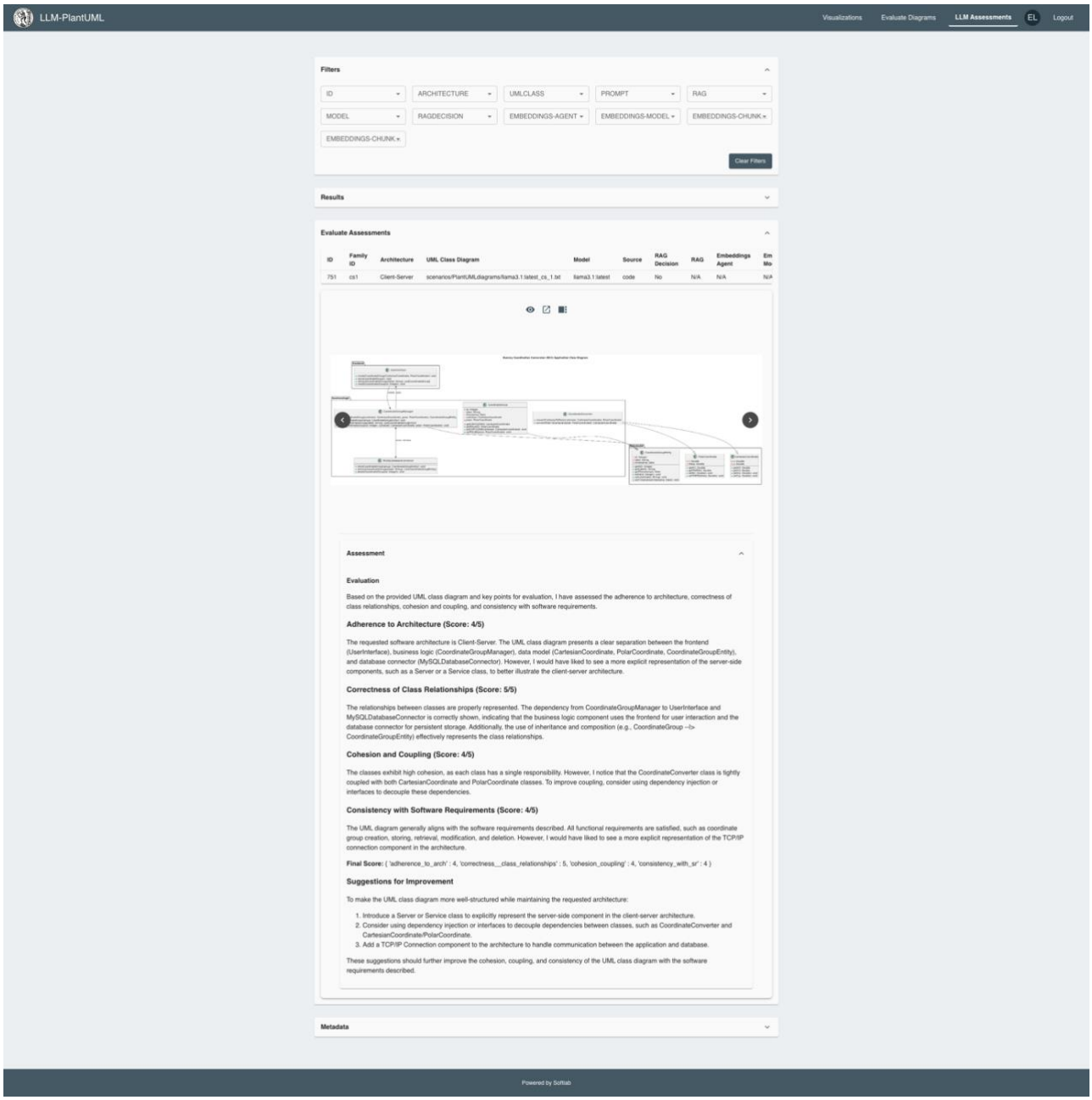


Figure 4.22 LLM assesssments page

4.3 Evaluation Results

In this section, we present charts generated from the evaluation data, focusing on insights derived from both human and LLM assessments of the class diagrams. All the charts presented in this section were created using the **Highcharts** library for clear and interactive visualization.

4.3.1 Overview of evaluations made by LLMs

We begin with an analysis of the scores provided by the LLMs. The following charts depict the **average scores assigned by all LLMs** across the evaluation categories, as well as a breakdown

of the scores given by each individual model. In addition, we compare these scores to those assigned by human evaluators to identify patterns and discrepancies.

From our observations during the evaluation process, it became evident that LLMs are not yet fully capable of evaluating class diagrams fairly and accurately. A notable trend is that LLMs consistently assign **higher evaluation scores** compared to human experts, even for diagrams that clearly fall short in multiple evaluation criteria. This tendency is captured in the following charts.

The chart in Figure 4.23 compares the **average scores of LLMs** to those of human evaluators, using a bar chart format. The data highlights significant differences, particularly in the categories of **adherence to architecture** and **correctness of class relationships**.

- **Adherence to Architecture**: LLMs often fail to critically analyze whether the principles of the requested architectural pattern are being followed, leading to inflated scores.
- **Correctness of Class Relationships**: This category also shows a notable gap, likely because accurately assessing class relationships requires a deeper understanding of the system's intended design logic—an area where human expertise is more decisive.

These discrepancies underline the limitations of LLMs in providing nuanced and accurate evaluations, particularly for criteria that demand critical thinking and architectural understanding.

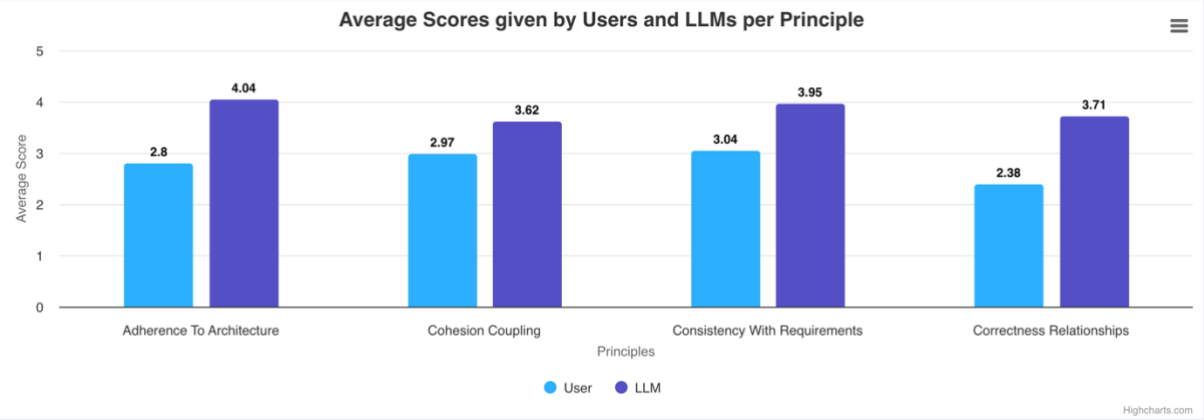


Figure 4.23 Average Score by Users vs LLMs

The chart in Figure 4.24 presents a comparison of the **average scores assigned by each LLM**. The results highlight certain models, such as **command-r**, **mixtral:8x7b**, and **phi3:medium-128k**, which appear unsuitable for class diagram evaluation. These models consistently assign **significantly high scores**, often exceeding **4/5 across almost all categories**, regardless of the quality of the diagrams.

Conversely, there are LLMs ,like gemma2:27b and gpt-4o, whose average scores seem more aligned with user evaluations. However, this apparent alignment is not consistently reflected in the detailed analysis of individual diagrams or in our observations of LLM scoring behavior.

Overall, the data reinforces the conclusion that LLMs generally demonstrate **poor performance in evaluating class diagrams**, with a tendency to overrate diagrams across various criteria. Consequently, for the remainder of the analysis, we shift our focus to **user evaluations** to better understand the impact of experimental parameters on the quality of generated diagrams.

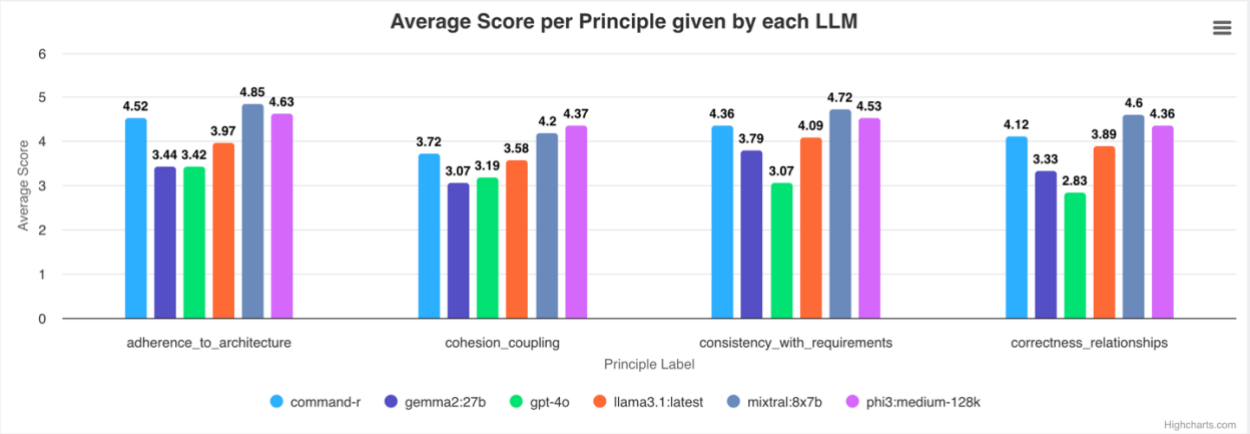


Figure 4.24 Average Score given by each LLM

4.3.2 Evaluation by humans

The chart in Figure 4.25 illustrates the average scores of class diagrams generated by each LLM, aggregated across all experimental parameters. It is important to note that online models (ClaudeSonnet3.5, Copilot, Gemini, GPT-4o, GPT-4o Software-Architecture-Visualizer) only generated diagrams without RAG. Therefore, the parameters that varied for these models were limited to the requested architecture and the FR/NFR sets. As a result, the number of class diagrams produced by online models was significantly smaller compared to local models.

Despite this limitation, the results clearly show that the state-of-the-art online models, particularly **ClaudeSonnet3.5**, **GPT-4o**, and its fine-tuned version **GPT-4o Software-Architecture-Visualizer**, achieved the highest average scores across all evaluation criteria. These models consistently outperformed local LLMs in adherence to architectural principles, class relationships, cohesion, and requirement alignment.

However, the performance of local models should not be overlooked. While **phi3:medium-128k** delivered **disappointing results**, with an average score of just 1.4/5, other local models demonstrated surprisingly strong performance. In particular**, Gemma2:27b and Command-r** stood out, with consistently high scores that rivaled or outperformed the online models in many cases. Additionally, Llama3.1:latest and Mixtral:8x7b showed respectable, though less prominent, performance.

Overall, GPT-4o dominated in terms of average performance, confirming its position as a leading model.. However, the noteworthy performance of Gemma2:27b, a local model, deserves special attention and further exploration for its potential in secure, on-premises applications.

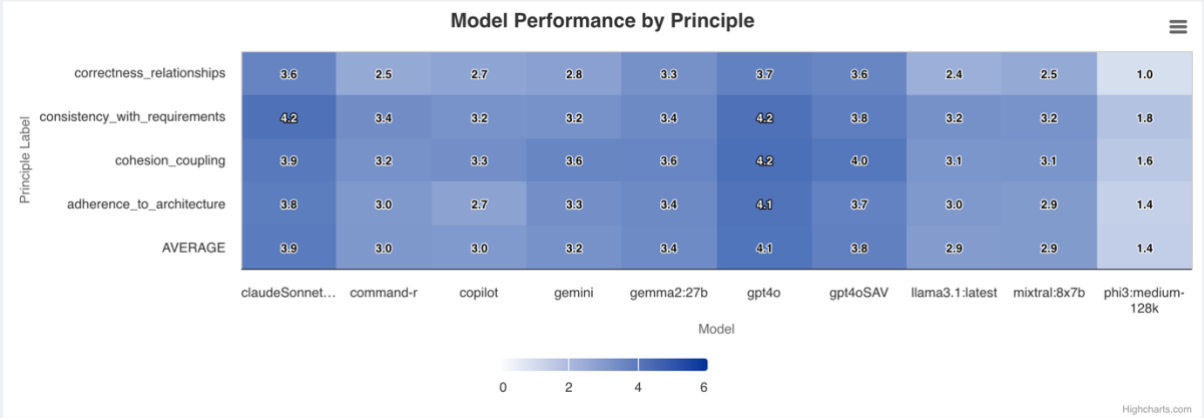


Figure 4.25 Model Performance based on human evaluations

To analyze the impact of different experimental parameters on the generated class diagrams, beginning with an analysis of the role of RAG (Retrieval-Augmented Generation).

The charts in Figure 4.26 and Figure 4.27 illustrate the average scores of class diagrams generated by each LLM, separated into those enhanced by RAG and those generated without RAG. The results are interesting and promising, offering insights into areas for future improvement.

Nearly all local models show a significant improvement in evaluation scores when their outputs are enhanced by RAG. However, phi3:medium-128k is an exception, as its average evaluation score decreases from 1.6 to 1.4 when RAG is applied. This decline can be attributed to the model's tendency to become confused by the additional context provided in the RAG files. As highlighted in Section 4.1 on weak diagrams, phi3:medium-128k often produces irrelevant or inconsistent diagrams, influenced by the examples and text in the RAG process.

For the other LLMs, the use of RAG leads to notable improvements in diagram quality. Specific examples include:

- **Command-r**: Average score improves from **2.8** to **3.1**.
- **Gemma2:27b**: Average score rises from **3.1** to an impressive **3.5**, even outperforming Gemini and Copilot in this setup.
- **Llama3.1:latest**: Average score increases from **2.3** to **3.1**.
- **Mixtral:8x7b**: Average score improves from **2.6** to **3.0**.

The results underline the **potential of RAG** to enhance the performance of local LLMs in generating class diagrams. While RAG helps models better understand and structure context, its effectiveness depends on the model's ability to process the additional information. For most local models, RAG provides a clear advantage, with **Gemma2:27b** emerging as a standout performer, achieving competitive results against some online models. This reinforces the importance of fine-tuning both the RAG process and the choice of embedding models to further optimize outputs.

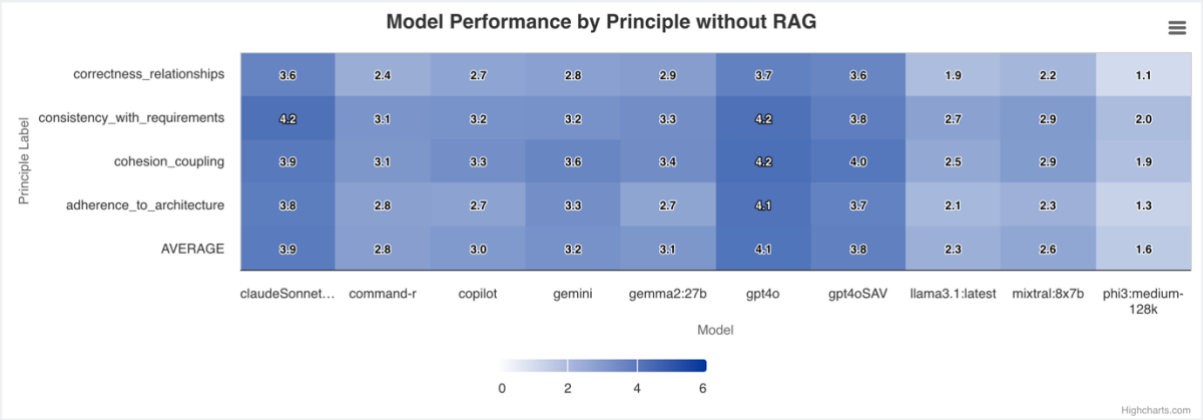


Figure 4.26 Model Performance without RAG based on human evaluations

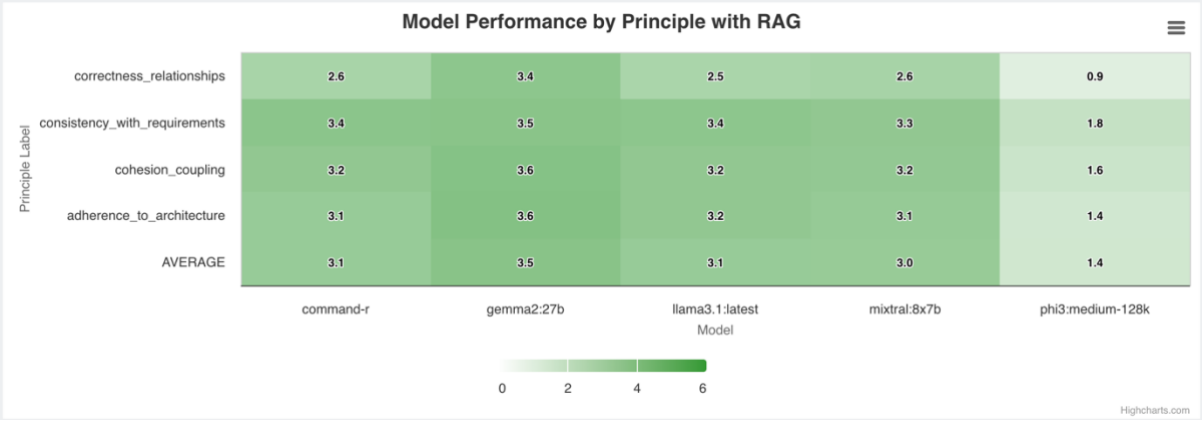


Figure 4.27 Model Performance with RAG based on human evaluations

The **RAG technique** itself involves multiple parameters that significantly influence its effectiveness in enhancing LLM responses. These parameters include the **embedding model**, the **chunking method**, and the design of the **RAG documents** that make up the corpus data. Each of these components plays a critical role in how well the additional context provided by RAG aligns with the requirements and improves LLM outputs.

The analysis starts with the **embedding model** and **chunking method**, which are fundamental to the RAG process. The following chart provides a direct comparison of the two chunking methods used: **semantic chunking** and **recursive chunking**.

The chart in Figure 4.28 clearly illustrates that **semantic chunking** outperforms **recursive chunking** across all evaluation criteria. This outcome aligns with expectations, as semantic chunking is a more advanced technique that dynamically segments data based on content and meaning. By grouping semantically related information into cohesive chunks, it ensures that the retrieved context is relevant and aligned with the query.

In contrast, **recursive chunking** takes a more rigid approach, splitting text based on structural elements such as characters or paragraphs. While simpler to implement, this method does not account for the underlying meaning of the text, leading to less precise and contextually relevant chunks. As a result, diagrams generated using recursive chunking tend to take lower scores.

This finding reinforces the need to prioritize advanced chunking strategies in future implementations of RAG to maximize its potential for enhancing LLM performance.

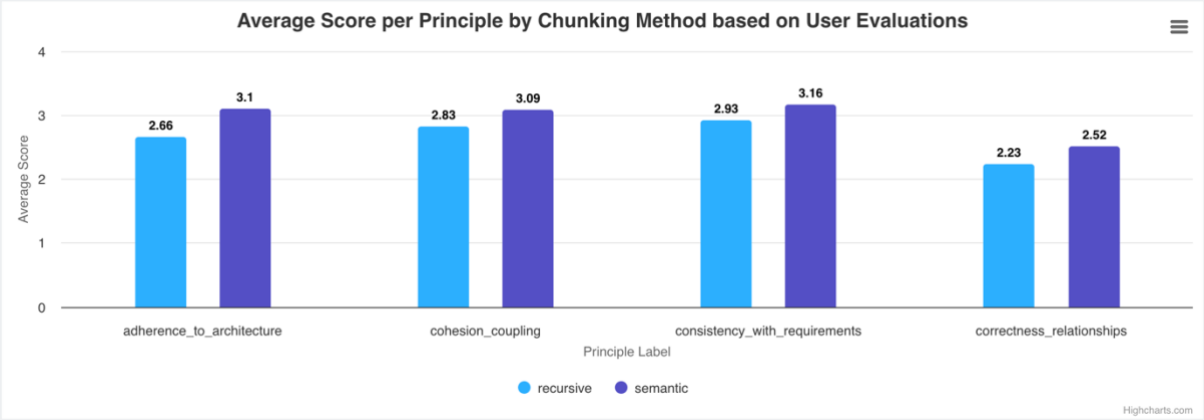


Figure 4.28 Average Score by chunking method based on human evaluation

In contrast to the clearer excellence observed with the **chunking methods**, the comparison of **embedding models** reveals a more balanced performance. The results in Figure 4.29 show that **nomic-embed-text:latest** presents a surprising level of competitiveness with the cloud-based, commercial **text-embedding-3-large** model.

Specifically, **nomic-embed-text:latest** achieves slightly higher scores in the **adherence to architecture** category, while **text-embedding-3-large** demonstrates a marginally better performance across the remaining three evaluation criteria. Despite these subtle differences, the results highlight an important point: **nomic-embed-text:latest**, as a free, open-source, locally deployable embedding model, provides a viable alternative to the commercial, paid **text-embedding-3-large**.

This comparison underscores the growing potential of local, open-source models to rival commercial counterparts in embedding tasks, offering a cost-effective and secure solution for those prioritizing privacy and independence from cloud-based services.

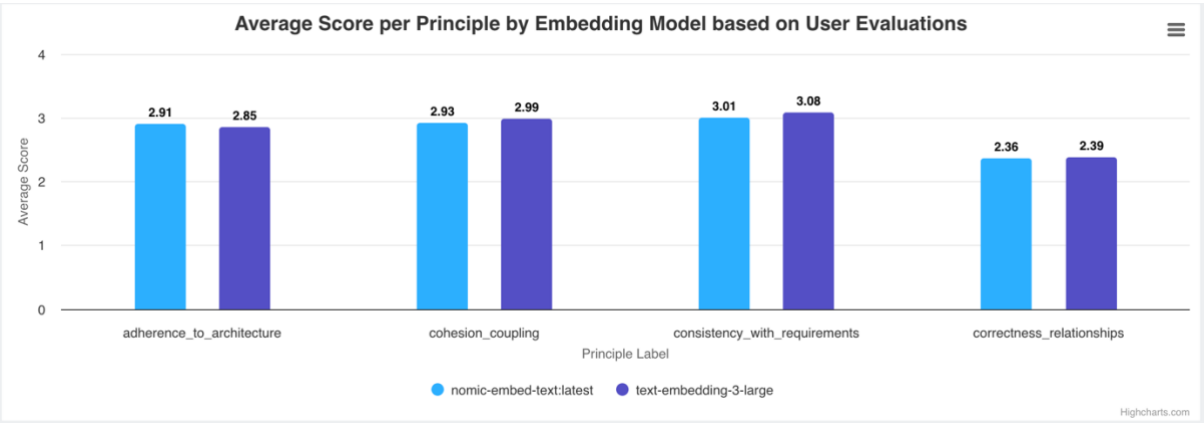


Figure 4.29 Average Score by embedding model based on human evaluations

The chart in Figure 4.30 illustrates the scores achieved by diagrams grouped according to the combination of embedding model and chunking method. The results reveal a relatively balanced performance across all combinations, with no significant discrepancies observed. The scores align with the trends noted in the previous two charts, reflecting consistent tendencies in how these parameters impact the quality of the generated diagrams.

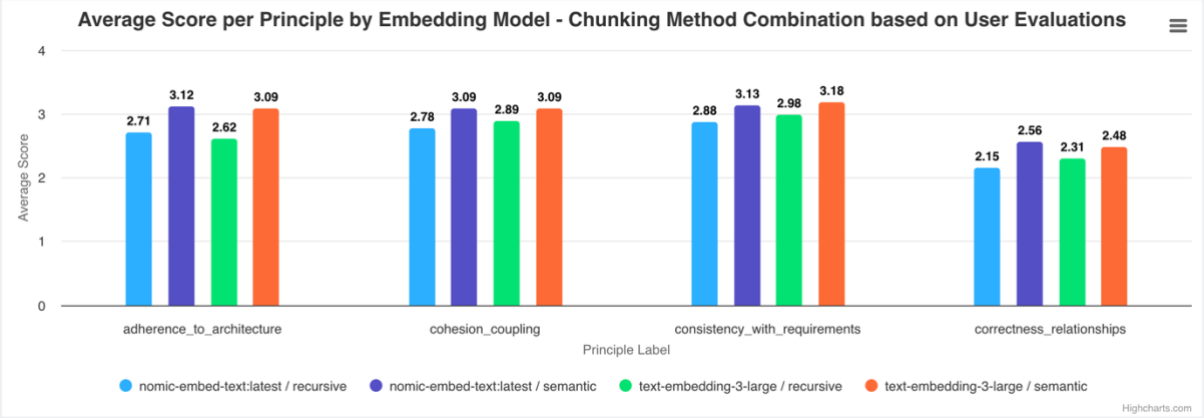


Figure 4.30 Average Score by embedding model – chunking method based on human evaluations

The focus now turns to the impact of the **RAG files parameter** on the performance of the generated class diagrams. As discussed in the section 3.1.2, we experimented with three distinct types of RAG files or approaches:

1. **RAG1**: This file contains comprehensive knowledge about the selected architectures extracted from the foundational text, *Software Engineering, 10th Edition* by Ian Sommerville. It serves as a high-quality, authoritative source of information.
2. **RAG2, RAG3, RAG4**: In this approach, we split the content of RAG1 into separate, architecture-specific files. For example:
    - **RAG2** contains only the content relevant to the Client-Server architecture.
    - **RAG3** focuses exclusively on the Three-Tier architecture.
    - **RAG4** is specific to the Model-View-Controller (MVC) architecture.

    This targeted approach ensures that LLMs receive information explicitly relevant to the requested architecture.
3. **RAG5**: This file compiles information from publicly available sources such as Wikipedia and GeeksforGeeks, representing a more generalized and less curated dataset.

The chart in Figure 4.31 reveals several noteworthy trends regarding the impact of RAG file types on the quality of the generated class diagrams. The approach that yielded the best results was separating the content into architecture-specific files (RAG2, RAG3, RAG4). By providing the LLMs with targeted and relevant information, this method minimized confusion and ensured that the context directly aligned with the requested architecture independently from RAG accuracy. This specificity appears to enhance the models' ability to produce accurate and coherent class diagrams.

Probably surprisingly, the second-best performance came from RAG5, which used publicly available sources such as Wikipedia and GeeksforGeeks. Despite being less curated and potentially less reliable, this approach outperformed RAG1, which was based on the authoritative textbook by Ian Sommerville. This result suggests that the diversity and accessibility of information from web-based sources may offer LLMs a broader, more practical context for understanding the architectural requirements, even if the material is less formal.

In contrast, RAG1, derived from a foundational academic text, showed relatively weaker performance. While the information was of high quality, its broader scope and less targeted structure

may have diluted its effectiveness. The results indicate that even high-quality data must be delivered in a precise and focused manner to maximize its utility for LLMs.

These findings underscore the critical role of **targeted and contextually relevant information** in guiding LLM performance. Moreover, the strong showing of web-based sources highlights the potential for leveraging accessible and diverse datasets to improve AI-assisted tasks, suggesting a promising direction for future research.

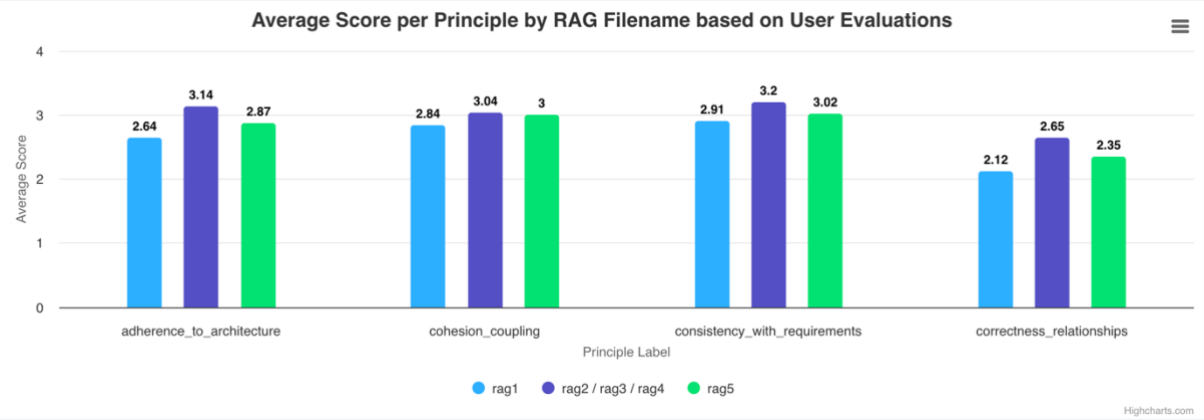


Figure 4.31 Average Score by RAG file based on human evaluations

Another critical parameter that significantly impacts LLM responses is the FR/NFR sets. Their structure, level of detail, and the language used as part of the prompt appear to play a decisive role in the quality of the resulting class diagrams. In the next chart, we revisit this observation with scores data, a concept briefly discussed earlier in 4.1 when we analyzed specific class diagrams demonstrating this behavior.

The chart in Figure 4.32 illustrates the **average scores achieved by class diagrams generated using the two FR/NFR sets** presented in earlier section. As seen, diagrams generated using the **first set** consistently outperform those generated with the **second set** across all evaluation criteria. Notably, the differences in scores are more pronounced in certain categories, while in others, they are relatively marginal. Specifically in the categories **"Cohesion and Coupling"** and **"Consistency with Requirements,"** the differences in scores are slight, and conversely, the chart highlights significant differences in the categories **"Adherence to Architecture"** and **"Correctness of Class Relationships,"** where diagrams generated with the second set fall noticeably short.

While this initial observation suggests a performance gap between the two sets, the underlying reasons for this behavior require further exploration. For additional insights we have created one more diagram for this analysis.

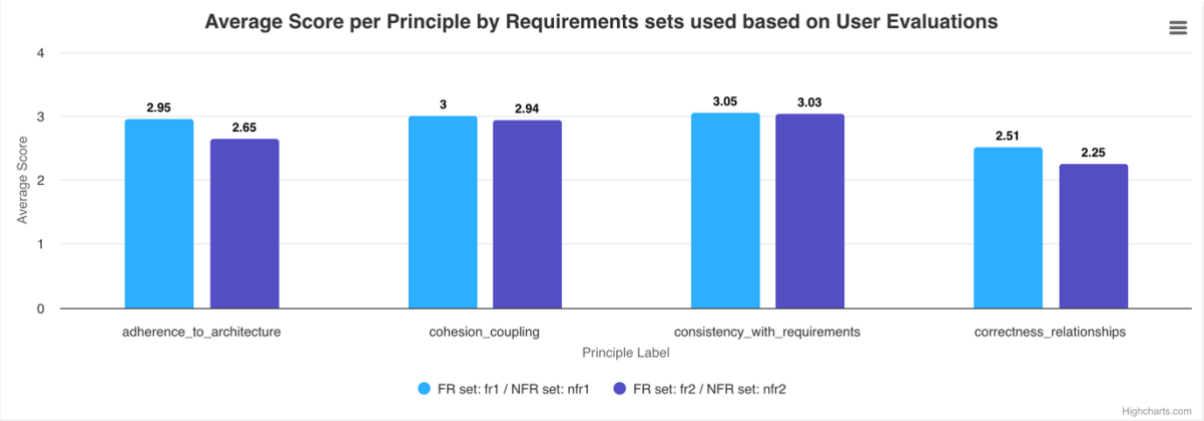


Figure 4.32 Average Score by requirements set based on human evaluations

The chart in Figure 4.33 offers a deeper perspective by breaking down the **average scores of diagrams by architecture type**(Client-Server, Model-View-Controller, and Three-Tier) and FR/NFR set used. This detailed view provides clarity on the patterns observed in Figure 4.32.

Key observations include:

- Client-Server and Model-View-Controller (MVC) Architectures:
    - In these architectures, diagrams generated with the **second set** show a significant decline in scores across all evaluation criteria, especially in **"Adherence to Architecture"** and **"Correctness of Class Relationships."**
- Three-Tier Architecture:
    - Interestingly, the second set performs better for this architecture, with higher average scores across all criteria.

This initially unexpected behavior can be attributed to the structure of the **second FR/NFR set (FR2)**. As detailed earlier, FR2 divides requirements into three distinct segments: **User Interactions**, **Logic**, and **Data Operations.** This segmentation closely aligns with the principles of the **Three-Tier architecture**, where responsibilities are typically divided into three layers: **Presentation Layer, Logic Layer, and Data Access Layer.** As a result, the LLMs naturally follow this separation in their generated diagrams, leading to higher scores for Three-Tier architecture.

In contrast, the **Client-Server** and **MVC architectures** do not inherently align with this segmented approach. Their design principles rely on different separations of responsibilities, and the structure of FR2 seems to reduce the LLMs' ability to generate diagrams that adhere to these architectural patterns. This misalignment is particularly evident in categories tied closely to architecture, such as **Adherence to Architecture** and **Correctness of Class Relationships,** where scores see significant deterioration.

These findings underscore the importance of tailoring FR/NFR sets to align with the principles of the requested architecture or ensuring that they are neutral and not biased towards any specific architectural pattern. The structure and phrasing of requirements can significantly influence the LLM's ability to generate accurate and relevant class diagrams. For a clearer understanding, refer back to the structure of the two FR/NFR sets as presented in 3.1.2, as well as the specific diagrams discussed in section 4.1 that illustrate this behavior.

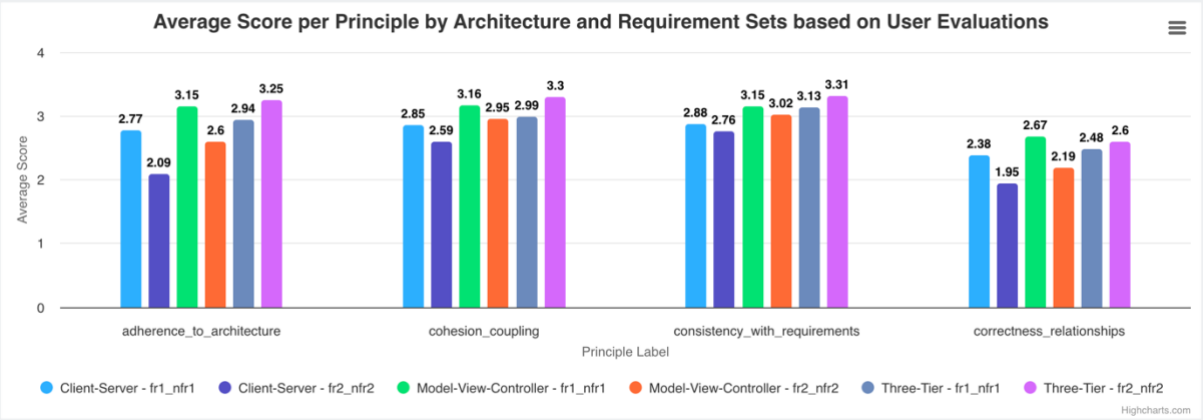


Figure 4.33 Average Score by architecture and requirements set based on human evaluations

Finally, we analyze whether the **requested architecture itself** introduces discrepancies in the average scores of the generated class diagrams. The next charts display the **average scores grouped by each requested architecture**.

The chart in Figure 4.34 reveals that diagrams with the **Client-Server** architecture as the requested pattern achieve the **lowest average scores**. The other two architectures, **Three-Tier** and **Model-View-Controller (MVC)**, show smaller differences in their scores, with Three-Tier diagrams slightly outperforming those of MVC. However, it is important to note that this chart reflects the combined influence of the FR set selection, particularly the poor alignment of the second FR set with the Client-Server and MVC architectural patterns.

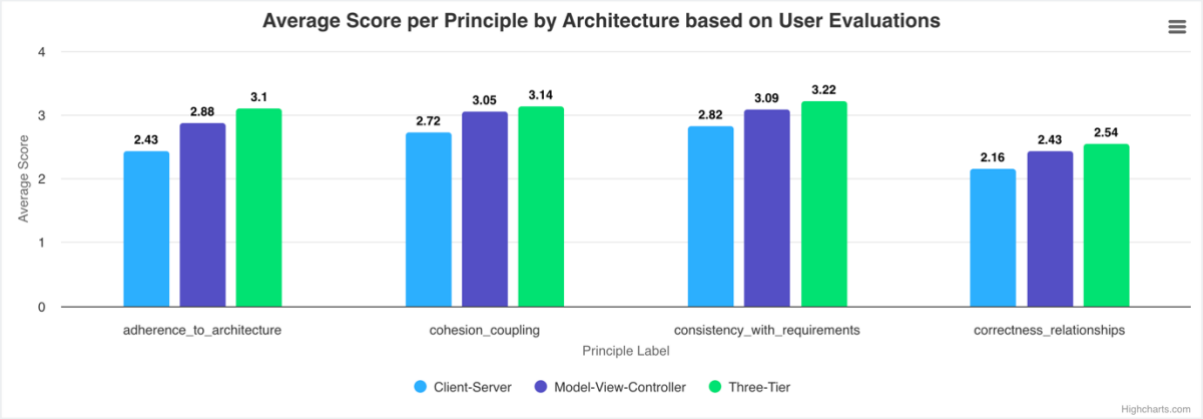


Figure 4.34 Average Score by architecture based on human evaluations

To isolate the impact of the requested architecture, the chart in Figure 4.35 presents the same analysis but considers only diagrams generated using the **FR1 set**, which, as observed during our exploration of class diagrams, exhibits more balanced and consistent behavior across architectures.

As expected, the results show reduced discrepancies in the scores. While Client-Server remains the architecture with the lowest scores, the relative performance of MVC and Three-Tier changes: MVC diagrams now surpass those with Three-Tier.

The findings suggest that Client-Server architecture likely presents an inherent challenge for LLMs, making it more difficult for them to adhere to its principles compared to the other two architectures. In contrast, the relative performance of MVC and Three-Tier appears to depend

heavily on the FR set used. This highlights the importance of both the requested architecture and the design of the accompanying FR/NFR sets in determining the quality of the generated class diagrams.

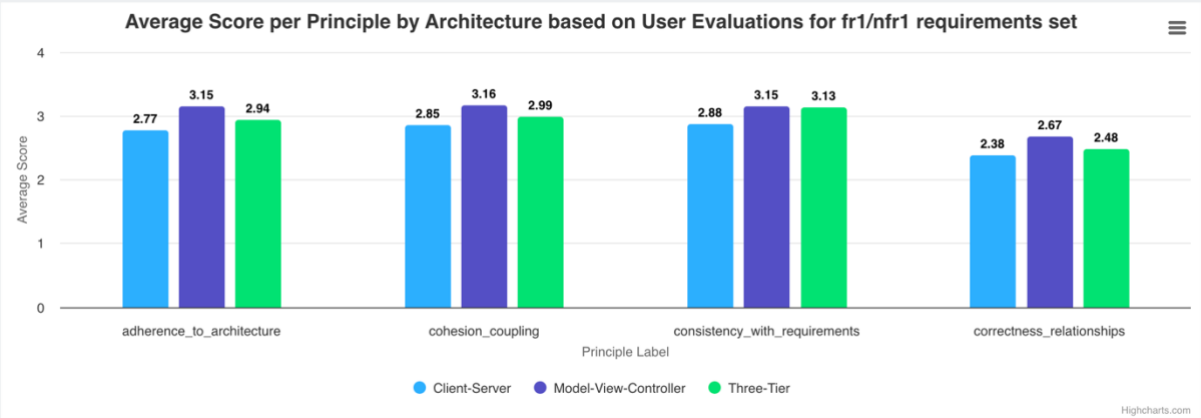


Figure 4.35 Average Score by architecture for FR1 set based on human evaluations

Similar insights can be drawn from the following **score distribution analysis**, which examines the evaluation scores for each requested architecture separately. The distribution charts in Figure 4.36 , Figure 4.37 and Figure 4.38 reveal that diagrams generated with the **Client-Server architecture** have a higher proportion of lower average scores across all criteria compared to the other two architectures. It is observed that as we transition from Client-Server to Model-View-Controller, and subsequently to Three-Tier, the peak of the distribution charts progressively shifts to the right.

Besides we can detect that diagrams with **average scores between 3.5 and 5** account for:

- Less than **20%** for Client-Server,
- Approximately **30%** for MVC, and
- Nearly **40%** for Three-Tier.

These distributions highlight the relative difficulty LLMs face in adhering to the Client-Server architecture, as opposed to the more consistent performance observed with Three-Tier and MVC patterns.

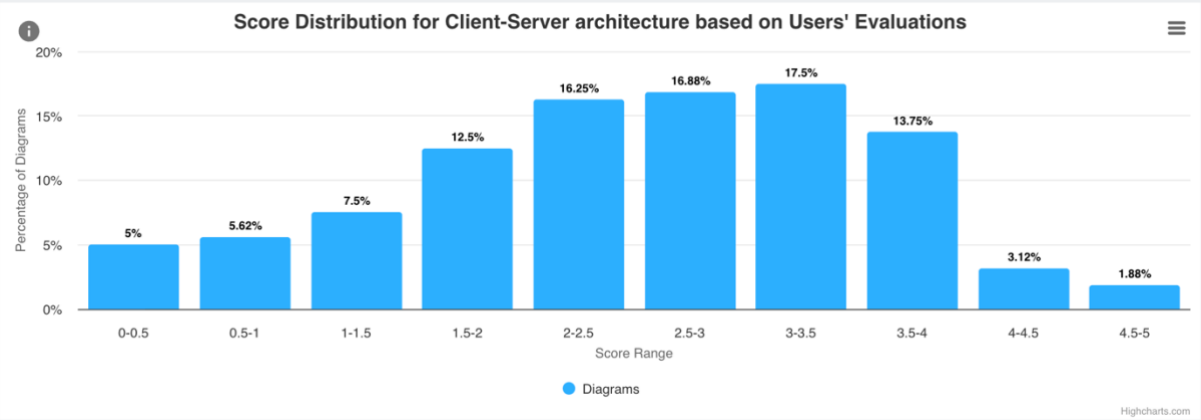


Figure 4.36 Score distribution for Client-Server architecture based on human evaluations

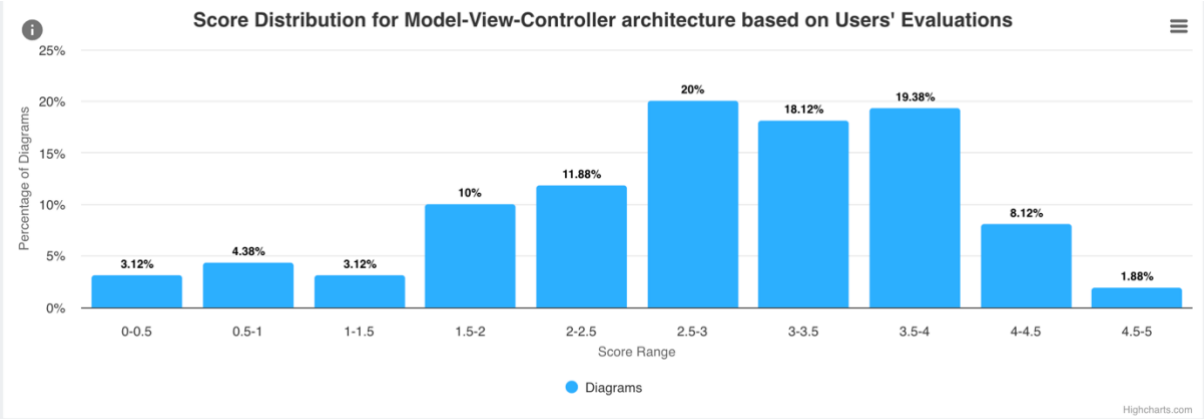


Figure 4.37 Score distribution for MVC architecture based on human evaluations

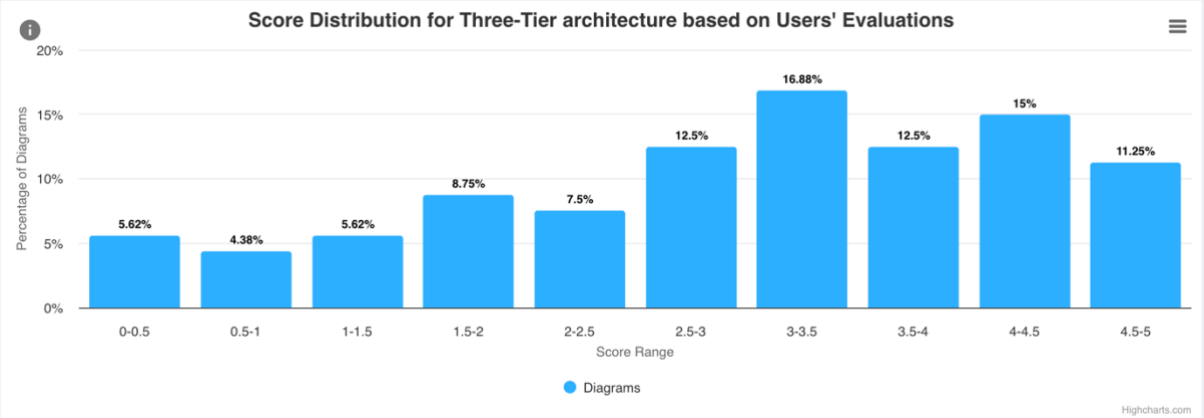


Figure 4.38 Score distribution for Three Tier architecture based on human evaluations

4.3.3 Discussion of results

The analysis of results highlights several findings about the factors influencing the quality of LLM-generated class diagrams for the context and the architectures examined. First of all, it is important to emphasize that the numeric values of grading by humans are subjective, and that a more extensive evaluation by human experts would offer even higher confidence to the findings. Still, the relative ranking of models and techniques, supports the generic feeling of the human experts who participated. **RAG techniques** proved to offer a notable improvement, with **semantic chunking** outperforming recursive methods and embedding models showing balanced yet competitive results between open-source and commercial LLMs. The structure of **FR/NFR sets of textual descriptions** included in LLM prompts significantly impacted diagram quality, with architecture-neutral sets yielding the best results, while more biased sets like FR2 posed challenges for architectures such as Client-Server and MVC.

Surprisingly, among the requested architectures, LLMs' performance in generating **Client-Server consistently demonstrated lower scores**, suggesting inherent difficulties for LLMs in adhering to its principles, while **Three-Tier and MVC** showed stronger alignment, particularly when FR1 was used. Additionally, the choice of **RAG file content** revealed that targeted-architecture-specific files drove the highest performance; also, web-sourced RAG materials surprisingly outperformed academic ones in some scenarios.

These findings underscore the critical role of precise inputs, targeted context, and optimized processes in improving LLM performance for software architecture tasks. Future research should focus on refining these parameters to further enhance diagram quality and explore the adaptability of LLMs to diverse architectural patterns.

5 Discussion

This thesis focused on the investigation of the capabilities and limitations of LLMs in generating architectures as UML class diagrams that follow specific architectural patterns, namely Client-Server, Three-Tier, and MVC. Through a systematic experimentation involving 480 scenarios, we analyzed the impact of various selected parameters, including the textual expression of functional and non-functional requirements sets, RAG techniques, embedding models, and chunking methods. A custom online presentation tool was developed to streamline the visualization and evaluation of architectures, further enabling the assessment by human experts.

5.1 By architecture

The results of this study demonstrate a promising potential of LLMs in generating architectures as UML class diagrams. Among the generated diagrams, a significant portion showcased LLMs' understanding of architectural principles, accurately reflecting the requested patterns and adhering to key evaluation criteria. However, the Client-Server architecture surprisingly posed particular challenges, often resulting in lower evaluation scores compared to Three-Tier and MVC. The latter showed stronger adherence when provided with well-structured inputs, emphasizing the importance of input quality and architecture alignment. This can be explained by the diversity of materials that have trained LLMs: the early conceptions of "client-server" are about thick clients that implemented all business logic and database-only servers, which has evolved into a more even sharing of responsibilities, and application servers implementing a share or even all of business logic

Despite the high-quality diagrams, some outputs scored at medium levels, reflecting a reasonable understanding of requirements but with shortcomings in specific areas like class relationships or adherence to the architecture. On the other hand, weaker results included "total failure diagrams" that failed to respect simple architectural patterns or lacked cohesion, highlighting the challenges posed by misaligned inputs, less effective techniques and, most importantly, materials that have been used in training LLMs.

5.2 By RAG method

RAG techniques and targeted context files significantly enhanced LLM performance in most cases. Semantic chunking and architecture-specific RAG files delivered the best results, illustrating the importance of tailoring retrieval methods to the specific requests to LLMs. The study also revealed that web-sourced RAG files outperformed academic resources in some cases, suggesting that diverse, accessible datasets can sometimes yield better outcomes than traditional content, largely acceptable as reference software engineering BOK materials. This insight highlights the impact of such data sources in improving LLM outputs.

5.3 By LLM

Local, open-source models like **Gemma2:27b** demonstrated competitive performance against state-of-the-art cloud-based models such as GPT-4o. This finding underscores the potential for secure, cost-effective, and privacy-preserving solutions in software architectural design automation. The analysis also revealed clear distinctions in model performance, emphasizing the importance of selecting an acceptable LLM, especially regarding open source models. These findings provide a basis for future improvements, focusing on optimizing LLM processes and employing proven techniques to enhance the overall quality of outputs.

Overall, the results provide valuable insights into what works and what doesn't, offering a clear direction for future investigations. They emphasize the importance of selecting appropriate models, mitigating biases in prompts that may influence LLM behavior, avoiding consistently underperforming models, and leveraging proven techniques to optimize performance of capable models. This comprehensive analysis not only highlights the current capabilities of LLMs but also lays the groundwork for targeted advancements in AI-assisted software design.

5.4 Future Work

The results of this study not only highlight the current capabilities of large language models in generating UML class diagrams but also pave the way for several promising directions to refine and enhance their effectiveness.

One potential area of exploration is the expansion of input complexity. Rather than using prompts with functional and non-functional requirements (FR/NFR) and the requested architecture as separate inputs, future work could investigate providing the entire **Software Requirements Specification (SRS)** document as input. This comprehensive approach would enable LLMs to process a more detailed and integrated representation of the system's requirements, potentially leading to more accurate and contextually aligned outputs. However, image formats should be shifted from static PNG diagrams to structured descriptions such as PlantUML code, offering greater flexibility for modifications and evaluations. An additional parameter that could be explored deeper in future work is the architecture set. Building on the experience and knowledge gained from foundational architectures such as Client-Server, Three-Tier, and Model-View-Controller, we could potentially transition to more complex architectures like Microservices and SOA.

Focusing on further improving local models, such as **Gemma2**, which demonstrated strong performance in this study, represents another key direction. Efforts could be directed at improving RAG techniques tailored specifically for these models, including optimizing chunking strategies and embedding model configurations. Moreover, fine-tuning such local models through **few-shot learning** could be explored. By utilizing the top-performing diagrams from the current experiment as part of the fine-tuning dataset, it may be possible to further enhance the model's understanding of architectural patterns and diagram generation.

These efforts, combined with a deeper investigation into the interplay between architectural patterns, FR/NFR sets, and RAG processes, would help build a more robust and reliable framework for AI-assisted software architecture design. By leveraging the insights from this study, future work can continue to push the boundaries of what LLMs can achieve in this critical domain.

6 References

[1] Shaveta, “A review on machine learning,” *International Journal of Science and Research Archive*, vol. 9, no. 1, pp. 281–285, May 2023, doi: 10.30574/ijsra.2023.9.1.0410.

[2] R. Mu and X. Zeng, “A Review of Deep Learning Research,” *KSII Transactions on Internet and Information Systems*, vol. 13, no. 4, pp. 1738–1764, Apr. 2019, doi: 10.3837/tiis.2019.04.001.

[3] X. Wang, “The application of NLP in information retrieval,” *Applied and Computational Engineering*, vol. 42, no. 1, pp. 290–297, Feb. 2024, doi: 10.54254/2755-2721/42/20230795.

[4] H. Li, G. K. Rajbahadur, and C.-P. Bezemer, “Studying the Impact of TensorFlow and PyTorch Bindings on Machine Learning Software Quality,” *ACM Transactions on Software Engineering and Methodology*, Jul. 2024, doi: 10.1145/3678168.

[5] “ETHICAL CONSIDERATION IN AI,” *International Research Journal of Modernization in Engineering Technology and Science*, May 2024, doi: 10.56726/IRJMETS55881.

[6] H. W. Marar, “Advancements in software engineering using AI,” *Computer Software and Media Applications*, vol. 6, no. 1, p. 3906, Feb. 2024, doi: 10.24294/csma.v6i1.3906.

[7] C. M. Hicks, C. S. Lee, and K. L. Foster-Marks, “The New Developer Executive Summary The New Developer AI Skill Threat, Identity Change & Developer Thriving in the Transition to AI-Assisted Software Development.” [Online]. Available: https://www.pluralsight.com/product/flow/developer-success-lab/dsl-navigate-toolkit

[8] I. Ozkaya, “Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications,” May 01, 2023, *IEEE Computer Society*. doi: 10.1109/MS.2023.3248401.

[9] P. Tembhekar, M. Devan, and J. Jeyaraman, “Role of GenAI in Automated Code Generation within DevOps Practices: Explore how Generative AI,” *Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online)*, vol. 2, no. 2, pp. 500–512, Oct. 2023, doi: 10.60087/jklst.vol2.n2.p512.

[10] Z. Gao, “A review on statistical language and neural network based code completion,” *Applied and Computational Engineering*, vol. 22, no. 1, pp. 233–239, Oct. 2023, doi: 10.54254/2755-2721/22/20231222.

[11] M. Atemkeng, S. Hamlomo, B. Welman, N. Oyetunji, P. Ataei, and J. L. K. E. Fendji, “Ethics of Software Programming with Generative AI: Is Programming without Generative AI always radical?,” Aug. 2024, doi: https://doi.org/10.48550/arXiv.2408.10554.

[12] N. M. Dr. Naveenkumar Jayakumar, “Role of Machine Learning & Artificial Intelligence Techniques in Software Testing,” *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 6, pp. 2913–2921, Apr. 2021, doi: 10.17762/turcomat.v12i6.5800.

[13] M. A. Job, “Automating and Optimizing Software Testing using Artificial Intelligence Techniques,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 5, 2021, doi: 10.14569/IJACSA.2021.0120571.

[14] S. Gautam, A. Khunteta, and P. Sharma, “A Review on Software Testing Using Machine Learning Techniques,” *ECS Trans*, vol. 107, no. 1, pp. 3393–3406, Apr. 2022, doi: 10.1149/10701.3393ecst.

[15] Dusica Marijan; Arnaud Gotlieb, “Software Testing for Machine Learning,” 2022, doi: 10.48550/arxiv.2205.00210.

[16] L. Kharb, “Automated Testing in Machine Learning Systems,” *International Journal of Progressive Research in Engineering Management and Science*, Nov. 2023, doi: 10.58257/IJPREMS32282.

[17] J. Calle and C. Zapata, “QUARE: Towards a Question-Answering Model for Requirements Elicitation,” Jul. 29, 2022. doi: 10.21203/rs.3.rs-1872151/v1.

[18] C. Cheligeer, J. Huang, G. Wu, N. Bhuiyan, Y. Xu, and Y. Zeng, “Machine learning in requirements elicitation: a literature review,” *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 36, p. e32, Oct. 2022, doi: 10.1017/S0890060422000166.

[19] Zarina Che Embi, Khaleduzzaman, and Ng Kok Why, “A Systematic Review on Natural Language Processing and Machine Learning Approaches to Improve Requirements Specification in Software Requirements Engineering,” *International Journal of Membrane Science and Technology*, vol. 10, no. 2, pp. 1563–1577, Sep. 2023, doi: 10.15379/ijmst.v10i2.1828.

[20] W. Alhoshan, R. Batista-Navarro, and L. Zhao, “Towards a Corpus of Requirements Documents Enriched with Semantic Frame Annotations,” in *2018 IEEE 26th International Requirements Engineering Conference (RE)*, IEEE, Aug. 2018, pp. 428–431. doi: 10.1109/RE.2018.00055.

[21] C. Liu, Z. Zhao, L. Zhang, and Z. Li, “Automated Conditional Statements Checking for Complete Natural Language Requirements Specification,” *Applied Sciences*, vol. 11, no. 17, p. 7892, Aug. 2021, doi: 10.3390/app11177892.

[22] Q. Lu, L. Zhu, J. Whittle, and J. B. Michael, “Software Engineering for Responsible AI,” *Computer (Long Beach Calif)*, vol. 56, no. 4, pp. 13–16, Apr. 2023, doi: 10.1109/MC.2023.3242055.

[23] R. Khankhoje, “Quality Challenges and Imperatives in Smart AI Software,” in *Soft Computing, Artificial Intelligence and Applications*, Academy & Industry Research Collaboration Center, Dec. 2023, pp. 143–154. doi: 10.5121/csit.2023.132412.

[24] A. Barredo Arrieta *et al.*, “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI,” *Information Fusion*, vol. 58, pp. 82–115, Jun. 2020, doi: 10.1016/j.inffus.2019.12.012.

[25] E. A. Abdelnabi, A. M. Maatuk, and M. Hagal, “Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques,” in *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA*, IEEE, May 2021, pp. 288–293. doi: 10.1109/MI-STA52233.2021.9464433.

[26] O. MacMillan-Scott and M. Musolesi, “(Ir)rationality and cognitive biases in large language models,” *R Soc Open Sci*, vol. 11, no. 6, Jun. 2024, doi: 10.1098/rsos.240255.

[27] R. Dhar, K. Vaidhyanathan, and V. Varma, “Can LLMs Generate Architectural Design Decisions? -An Exploratory Empirical study,” Mar. 2024, [Online]. Available: http://arxiv.org/abs/2403.01709

[28] T. Eisenreich, S. Speth, and S. Wagner, “From Requirements to Architecture: An AI-Based Journey to Semi-Automatically Generate Software Architectures,” Jan. 2024, doi: 10.1145/3643660.3643942.

[29] S. Yang and H. Sahraoui, “Towards automatically extracting UML class diagrams from natural language specifications,” in *Proceedings - ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems, MODELS 2022: Companion Proceedings*, Association for Computing Machinery, Inc, Oct. 2022, pp. 396–403. doi: 10.1145/3550356.3561592.

[30] I. Altawaiha and A. Al-Hgaish, “ClassDiagGen Tool: Fine-Tuning the GPT-3 Model for Automated Class Diagram Generation from Textual Descriptions,” May 02, 2024. doi: 10.21203/rs.3.rs-4350615/v1.

[31] Ian. Sommerville, *Software engineering*. Pearson, 2011.

[32] P. Kumar and Y. Singh, “A Software Reliability Growth Model for Three-Tier Client Server System.”

[33] M. Yener and A. Theedom, “Model View Controller Pattern,” 2015. [Online]. Available: www.wrox.com/go/

[34] Eric Karlsson, “The evolution and erosion of a service-oriented architecture in enterprise software”.

[35] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, “From monolithic systems to Microservices: An assessment framework,” *Inf Softw Technol*, vol. 137, p. 106600, Sep. 2021, doi: 10.1016/j.infsof.2021.106600.