ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Ανάπτυξη RAG Chatbot για βοήθεια χρηστών εφαρμογής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Ε. Κίτσιος

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π

Αθήνα, Οκτώβριος 2025

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Ανάπτυξη RAG Chatbot για βοήθεια χρηστών εφαρμογής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Ε. Κίτσιος

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 3η Νοεμβρίου 2025

..... Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	..... Γεώργιος Γκούμας Καθηγητής Ε.Μ.Π.	..... Διονύσιος Πνευματικάτος Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2025

.....

Γεώργιος Ε. Κίτσιος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

Περίληψη

Τα chatbot συστήματα αποτελούν αναδυόμενο εργαλείο με πάρα πολλές πρακτικές εφαρμογές, προσφέροντας άμεση πληροφόρηση και βελτιώνοντας την αλληλεπίδραση με ψηφιακές υπηρεσίες. Η συγκεκριμένη εργασία εστιάζει στην ανάπτυξη ενός RAG (Retrieval-Augmented Generation) chatbot, σχεδιασμένου για την εφαρμογή "ΑΘΗΝΑ" που χρησιμοποιείται από εργαζόμενους τραπεζικού οργανισμού για την δημιουργία διαγραμμάτων με βάση οικονομικά δεδομένα.

Το chatbot που μελετάται στη συγκεκριμένη διπλωματική δημιουργήθηκε με σκοπό να απλοποιήσει την εμπειρία των χρηστών κατά την πλοήγηση και αξιοποίηση των δυνατοτήτων της εφαρμογής. Ειδικότερα, το σύστημα αντλεί πληροφορία από οδηγούς χρήσης και τεχνικά εγχειρίδια σε μορφή PDF, τα οποία αναλύονται και μετατρέπονται σε μορφή κατάλληλη για ανάκτηση πληροφορίας. Μέσα από αυτήν την προσέγγιση, ο χρήστης μπορεί να θέσει ερωτήματα σε φυσική γλώσσα και να λάβει κατατοπιστικές απαντήσεις που σχετίζονται με τη διαδικασία δημιουργίας διαγραμμάτων.

Η αρχιτεκτονική του συστήματος περιλαμβάνει μια διαδικασία αποθήκευσης των δεδομένων σε μορφή κατάλληλη για ανάκτηση καθώς και έναν μηχανισμό ανάκτησης των δεδομένων και παραγωγή της απάντησης με βάση τα σχετικά αποσπάσματα. Η συνδυαστική αυτή λειτουργία εξασφαλίζει ακρίβεια και συνάφεια στην πληροφόρηση που παρέχεται στον χρήστη.

Κατά την ανάπτυξη του συστήματος πραγματοποιήθηκαν πειραματικές αξιολογήσεις για την συνολική απόδοση του chatbot με διάφορες παραλλαγές σε βασικούς παράγοντες του συστήματος, χρησιμοποιώντας ειδικές μετρικές ποιότητας. Τα αποτελέσματα ανέδειξαν τις δυνατότητες του συστήματος να υποστηρίζει αποτελεσματικά τους χρήστες στην εργασία τους.

Η εργασία καταλήγει με μια αποτίμηση των πλεονεκτημάτων και των περιορισμών της προσέγγισης αυτής, καθώς και προτάσεις για μελλοντικές επεκτάσεις που θα μπορούσαν να ενισχύσουν περαιτέρω τη λειτουργικότητα του chatbot στην συγκεκριμένη εφαρμογή.

Λέξεις Κλειδιά:

Retrieval-Augmented Generation, Large Language Models, RAG chatbot, AI chatbot, Knowledge retrieval, Context-aware chatbot, Intelligent assistant, Generative AI, Vector search, Semantic search, Embedding-based retrieval, Custom knowledge base, LangChain, LlamaIndex, FAISS / Chroma / Pinecone

Abstract

Chatbot systems are an emerging tool with numerous practical applications, offering instant information and enhancing interaction with digital services. This particular thesis focuses on the development of a RAG (Retrieval-Augmented Generation) chatbot designed for the “ATHENA” application, which is used by employees of a banking organization to create charts based on financial data.

The chatbot studied in this thesis was created with the goal of simplifying the user experience when navigating and utilizing the features of the application. Specifically, the system retrieves information from user guides and technical manuals in PDF format, which are analyzed and converted into a form suitable for information retrieval. Through this approach, users can pose questions in natural language and receive informative answers related to the chart creation process.

The system’s architecture includes a pipeline for storing data in a format suitable for retrieval, as well as a retrieval mechanism for the data and the generation of a response based on the relevant chunks. This combined functionality ensures accuracy and relevance in the information provided to the user.

During the development of the system, experimental evaluations were conducted to assess the chatbot’s overall performance, testing various configurations of key system components using specialized quality metrics. The results highlighted the system’s potential to effectively support users in their tasks.

The thesis concludes with an assessment of the advantages and limitations of this approach, along with suggestions for future enhancements that could further improve the chatbot’s functionality within the specific application.

KeyWords:

Retrieval-Augmented Generation, Large Language Models, RAG chatbot, AI chatbot, Knowledge retrieval, Context-aware chatbot, Intelligent assistant, Generative AI, Vector search, Semantic search, Embedding-based retrieval, Custom knowledge base, LangChain, LlamaIndex, FAISS / Chroma / Pinecone

Ευχαριστίες

Καταρχάς, θα ήθελα να ευχαριστήσω την οικογένεια μου που ήταν δίπλα μου καθ' όλη τη διάρκεια των σπουδών μου και κατά τη διάρκεια περάτωσης αυτής της εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω την ερευνήτρια της σχολής Κατερίνα Δόκα και τον συνάδελφό της Γεώργιο Σταυρουλάκη που σκέφτηκαν την ιδέα της συγκεκριμένης εφαρμογής.

Τέλος, θα ήθελα να ευχαριστήσω όλους τους καθηγητές της σχολής, που χάρη σε αυτούς απέκτησα πάρα πολλές σημαντικές γνώσεις, οι οποίες θα μου φανούν χρήσιμες στη μετέπειτα επαγγελματική μου πορεία. Φυσικά θα ήθελα να ευχαριστήσω και τους τρεις επιβλέποντες της διπλωματικής Νεκτάριο Κοζύρη, Γεώργιο Γκούμα και Διονύσιο Πνευματικάτο.

Table of Contents

Table of Figures

Εκτενής Περίληψη στα Ελληνικά

1 Εισαγωγή

Τα τελευταία χρόνια, η τεχνητή νοημοσύνη και ιδιαίτερα η επεξεργασία φυσικής γλώσσας (NLP) έχουν γνωρίσει σημαντική πρόοδο, οδηγώντας στην ανάπτυξη chatbots που προσομοιάζουν την ανθρώπινη κατανόηση και ανταπόκριση.
Ένα χαρακτηριστικό παράδειγμα αποτελούν τα μοντέλα RAG (Retrieval-Augmented Generation), τα οποία συνδυάζουν τις δυνατότητες των μεγάλων γλωσσικών μοντέλων (LLMs) με την ανάκτηση πληροφοριών από εξωτερικές πηγές. Έτσι, παρέχουν πιο ακριβείς και τεκμηριωμένες απαντήσεις, ξεπερνώντας τους περιορισμούς των παραδοσιακών chatbots.
Η παρούσα εργασία εστιάζει στην ανάπτυξη ενός RAG chatbot για την εφαρμογή “ΑΘΗΝΑ”, που χρησιμοποιείται για τη δημιουργία διαγραμμάτων με οικονομικά δεδομένα τράπεζας. Το chatbot θα διευκολύνει τους εργαζόμενους, προσφέροντας άμεσες απαντήσεις σε απορίες χρήσης, χωρίς την ανάγκη αναζήτησης στον οδηγό, ενισχύοντας έτσι την παραγωγικότητα και την εμπειρία χρήσης.

2 Μηχανική Μάθηση

Η Μηχανική Μάθηση [1] περιγράφεται ως το πεδίο μελέτης που παρέχει στους υπολογιστές την ικανότητα να μαθαίνουν χωρίς να έχουν προγραμματιστεί ρητά. Αποτελεί έναν υποκλάδο της Τεχνητής Νοημοσύνης (Artificial Intelligence) που εστιάζει στην ανάπτυξη αλγορίθμων και στατιστικών μοντέλων, τα οποία επιτρέπουν σε υπολογιστικά συστήματα να "μαθαίνουν" από δεδομένα, χωρίς να είναι ρητά προγραμματισμένα για κάθε συγκεκριμένη εργασία. Ουσιαστικά, πρόκειται για μια διαδικασία κατά την οποία ο υπολογιστής αναγνωρίζει πρότυπα σε μεγάλα σύνολα δεδομένων και, βασιζόμενος σε αυτά, πραγματοποιεί προβλέψεις ή λαμβάνει αποφάσεις. Η θεμελιώδης αρχή της μηχανικής μάθησης είναι ότι τα συστήματα μπορούν να μάθσουν από τα δεδομένα, να εντοπίσουν μοτίβα και να λάβουν αποφάσεις με ελάχιστη ανθρώπινη παρέμβαση.

2.1 Κατηγορίες Μηχανικής Μάθησης

Οι αλγόριθμοι μηχανικής μάθησης ταξινομούνται σε τρεις κύριες κατηγορίες, ανάλογα με τον τρόπο με τον οποίο "μαθαίνουν" από τα δεδομένα: την Επιβλεπόμενη Μάθηση, τη Μη Επιβλεπόμενη Μάθηση και την Ενισχυτική Μάθηση.

2.1.1 Επιβλεπόμενη Μάθηση (Supervised Learning)

Στην επιβλεπόμενη μάθηση [2], ο αλγόριθμος εκπαιδεύεται σε ένα σύνολο δεδομένων που είναι "επισημασμένο" (labeled). Αυτό σημαίνει ότι για κάθε είσοδο δεδομένων, η σωστή έξοδος είναι ήδη γνωστή. Ο αλγόριθμος μαθαίνει να χαρτογραφεί τη σχέση μεταξύ των εισόδων και των εξόδων, ώστε να μπορεί να προβλέψει την έξοδο για νέα, μη επισημασμένα δεδομένα. Η επιβλεπόμενη μάθηση διακρίνεται περαιτέρω σε δύο τύπους προβλημάτων: την ταξινόμηση (classification), όπου η έξοδος είναι μια κατηγορία (π.χ., "spam" ή "όχι spam"), και την παλινδρόμηση (regression), όπου η έξοδος είναι μια συνεχής τιμή (π.χ., η τιμή ενός σπιτιού).

2.1.2 Μη Επιβλεπόμενη Μάθηση (Unsupervised Machine Learning)

Στη μη επιβλεπόμενη μάθηση , ο αλγόριθμος λαμβάνει δεδομένα που δεν είναι επισημασμένα. Ο στόχος δεν είναι να προβλέψει μια συγκεκριμένη έξοδο, αλλά να ανακαλύψει κρυμμένες

δομές, πρότυπα και σχέσεις μέσα στα ίδια τα δεδομένα. Οι πιο συνηθισμένες τεχνικές μη επιβλεπόμενης μάθησης περιλαμβάνουν τη συστοίχιση (clustering), που ομαδοποιεί παρόμοια δεδομένα, και τη μείωση διαστασιμότητας (dimensionality reduction), που απλοποιεί τα δεδομένα μειώνοντας τον αριθμό των μεταβλητών.

2.1.3 Ενισχυτική Μάθηση (Reinforcement Learning)

Η ενισχυτική μάθηση [3], [4] είναι μια προσέγγιση όπου ένας "πράκτορας" (agent) μαθαίνει να λαμβάνει αποφάσεις αλληλεπιδρώντας με ένα "περιβάλλον" (environment). Ο πράκτορας λαμβάνει "ανταμοιβές" (rewards) ή "ποινές" (penalties) για τις ενέργειες που εκτελεί. Ο στόχος του είναι να μάθει μια στρατηγική (policy), δηλαδή μια σειρά από ενέργειες, που θα μεγιστοποιήσει τη συνολική ανταμοιβή του με την πάροδο του χρόνου. Αυτή η προσέγγιση είναι εμπνευσμένη από τη συμπεριφορική ψυχολογία και χρησιμοποιείται ευρέως στη ρομποτική, στα παιχνίδια και στη βελτιστοποίηση συστημάτων.

3 Βαθιά Μάθηση (Deep Learning)

3.1 Εισαγωγή στη Βαθιά Μάθηση

Η Βαθιά Μάθηση αποτελεί έναν εξειδικευμένο και προηγμένο υποτομέα της μηχανικής μάθησης, ο οποίος αντλεί την έμπνευσή του από τη δομή και τη λειτουργία του ανθρώπινου εγκεφάλου, και συγκεκριμένα από τα βιολογικά νευρωνικά δίκτυα. Θεμελιώδης αρχή της είναι η χρήση πολυεπίπεδων τεχνητών νευρωνικών δικτύων (Artificial Neural Networks - ANNs) για την προοδευτική εξαγωγή χαρακτηριστικών υψηλότερου επιπέδου από τα δεδομένα εισόδου. Σε αντίθεση με τις παραδοσιακές μεθόδους μηχανικής μάθησης, όπου η διαδικασία της εξαγωγής χαρακτηριστικών (feature engineering) απαιτεί σημαντική ανθρώπινη παρέμβαση και εξειδικευμένη γνώση, τα μοντέλα βαθιάς μάθησης αυτοματοποιούν αυτή τη διαδικασία. Κάθε επίπεδο του δικτύου μαθαίνει να αναγνωρίζει όλο και πιο σύνθετα μοτίβα, μετασχηματίζοντας την αρχική, ακατέργαστη πληροφορία σε μια πιο αφηρημένη και σύνθετη αναπαράσταση. Αυτή η ιεραρχική προσέγγιση επιτρέπει την επίλυση εξαιρετικά πολύπλοκων προβλημάτων σε τομείς όπως η επεξεργασία φυσικής γλώσσας, η αναγνώριση εικόνας και η αυτόνομη οδήγηση, όπου τα παραδοσιακά μοντέλα συχνά αποτυγχάνουν να συλλάβουν την πολυπλοκότητα των δεδομένων.

3.2 Perceptron

Η έννοια του Perceptron [1] αποτελεί τον ακρογωνιαίο λίθο των τεχνητών νευρωνικών δικτύων. Πρόκειται για τον απλούστερο τύπο ενός τεχνητού νευρώνα, έναν αλγόριθμο επιβλεπόμενης μάθησης σχεδιασμένο για δυαδική ταξινόμηση (binary classification). Το Perceptron δέχεται ένα σύνολο εισόδων, καθεμία από τις οποίες πολλαπλασιάζεται με ένα αντίστοιχο βάρος (weight), το οποίο υποδηλώνει τη σημασία της συγκεκριμένης εισόδου. Στη συνέχεια, υπολογίζεται το σταθμισμένο άθροισμα αυτών των εισόδων. Το αποτέλεσμα διέρχεται από μια συνάρτηση ενεργοποίησης (activation function), συνήθως μια βηματική συνάρτηση, η οποία παράγει μια δυαδική έξοδο (π.χ., 1 ή 0, -1 ή 1). Η μαθηματική του διατύπωση είναι η εξής:

$$y = \varphi\left(\sum_{i=1}^{n} w_i\, x_i + b\right)$$

όπου y είναι η έξοδος, xi οι είσοδοι, wi τα βάρη, b η πόλωση (bias) και f η συνάρτηση ενεργοποίησης. Μέσω μιας διαδικασίας εκπαίδευσης, τα βάρη προσαρμόζονται επαναληπτικά ώστε το μοντέλο να μάθει να ταξινομεί σωστά τα δεδομένα. Αν και ένα μόνο Perceptron μπορεί να επιλύσει μόνο γραμμικά διαχωρίσιμα προβλήματα, η διασύνδεση πολλαπλών τέτοιων μονάδων σε πολυεπίπεδες αρχιτεκτονικές αποτέλεσε τη βάση για την ανάπτυξη των πολύπλοκων νευρωνικών δικτύων που γνωρίζουμε σήμερα.

3.3 Αρχιτεκτονικές Βαθιάς Μάθησης

Η ισχύς της βαθιάς μάθησης έγκειται στην ποικιλία των αρχιτεκτονικών της, καθεμία από τις οποίες είναι σχεδιασμένη για να επιλύει συγκεκριμένες κατηγορίες προβλημάτων. Η επιλογή της κατάλληλης αρχιτεκτονικής εξαρτάται από τη φύση των δεδομένων και τον επιδιωκόμενο στόχο.

3.3.1 Νευρωνικά Δίκτυα Πρόσθιας Τροφοδότησης (FNNs)

Τα Τεχνητά Νευρωνικά Δίκτυα Πρόσθιας Τροφοδότησης (Feedforward Neural Networks - FNN) [5] αποτελούν μία από τις πιο βασικές και ευρέως χρησιμοποιούμενες αρχιτεκτονικές στη βαθιά μάθηση. Η δομή τους είναι απλή αλλά ισχυρή, οργανωμένη σε διαδοχικά στρώματα που επεξεργάζονται τις πληροφορίες με συστηματικό τρόπο, επιτρέποντας τη μοντελοποίηση πολύπλοκων σχέσεων μεταξύ εισόδου και εξόδου.

Δομή του FNN:

1. **Στρώμα Εισόδου**: Δέχεται τα αρχικά δεδομένα, όπου κάθε νευρώνας αντιπροσωπεύει ένα χαρακτηριστικό.
2. **Κρυφά Στρώματα**: Μεταξύ εισόδου και εξόδου, τα κρυφά στρώματα μετασχηματίζουν τα δεδομένα μέσω γραμμικών και μη γραμμικών συναρτήσεων ενεργοποίησης, επιτρέποντας την ανίχνευση σύνθετων προτύπων.
3. **Στρώμα Εξόδου**: Παράγει την τελική πρόβλεψη, όπως μια κατηγορία για προβλήματα ταξινόμησης ή μια συνεχής τιμή για προβλήματα παλινδρόμησης.

Η κύρια ιδιότητα των FNN είναι η μονόδρομη ροή πληροφοριών, όπου τα δεδομένα περνούν διαδοχικά από το ένα στρώμα στο επόμενο χωρίς κύκλους ή ανατροφοδοτήσεις. Κατά τη διάρκεια αυτής της διαδικασίας, το δίκτυο δημιουργεί ολοένα και πιο αφηρημένες αναπαραστάσεις των δεδομένων. Για παράδειγμα, σε εφαρμογές αναγνώρισης εικόνων, τα αρχικά στρώματα μπορεί να εντοπίζουν απλά χαρακτηριστικά, όπως ακμές, ενώ τα βαθύτερα στρώματα αναγνωρίζουν πιο σύνθετα χαρακτηριστικά, όπως σχήματα ή ολόκληρα αντικείμενα.

3.3.2 Συνελικτικά Νευρωνικά Δίκτυα (CNNs)

Τα Συνελικτικά Νευρωνικά Δίκτυα (CNNs) [6] αποτελούν μια κατηγορία βαθιών αρχιτεκτονικών νευρωνικών δικτύων, ειδικά σχεδιασμένων για δομημένα δεδομένα με μορφή

πλέγματος, όπως εικόνες και χρονοσειρές. Η φιλοσοφία τους βασίζεται στον οπτικό φλοιό του ανθρώπινου εγκεφάλου, όπου οι νευρώνες ανταποκρίνονται σε τοπικά πεδία υποδοχής.
Με τη χρήση μαθησιακών φίλτρων που εφαρμόζονται με συνέλιξη, τα CNNs εντοπίζουν τοπικά χωρικά μοτίβα, επαναχρησιμοποιώντας αποδοτικά τα βάρη. Έτσι, χτίζουν ιεραρχικές αναπαραστάσεις χαρακτηριστικών: από απλά στοιχεία (όπως ακμές και υφές) έως πιο σύνθετες έννοιες (όπως αντικείμενα και σχήματα).
Παρά την εμφάνιση νέων μοντέλων, όπως οι Vision Transformers, τα CNNs παραμένουν κεντρικό εργαλείο της Τεχνητής Νοημοσύνης λόγω της αποδοτικότητάς τους, της ανθεκτικότητάς τους και της υψηλής τους επίδοσης σε προβλήματα όρασης υπολογιστών, όπως η αναγνώριση εικόνας, η τμηματοποίηση και η ανίχνευση αντικειμένων.
Η τυπική αρχιτεκτονική ενός CNN περιλαμβάνει συνελικτικά στρώματα για εξαγωγή χαρακτηριστικών, μη γραμμικές συναρτήσεις ενεργοποίησης (π.χ. ReLU), πράξεις pooling για μείωση υπολογιστικού κόστους και ανθεκτικότητα, στρώματα κανονικοποίησης για σταθεροποίηση εκπαίδευσης και πλήρως συνδεδεμένα στρώματα για λήψη αποφάσεων.

3.3.3 Αναδρομικά Νευρωνικά Δίκτυα (RNNs)

Τα Αναδρομικά Νευρωνικά Δίκτυα (Recurrent Neural Networks - RNNs) [7] αποτελούν μια κατηγορία νευρωνικών δικτύων ειδικά σχεδιασμένη για την επεξεργασία ακολουθιακών δεδομένων. Σε αντίθεση με τα παραδοσιακά δίκτυα πρόσθιας τροφοδότησης (feed-forward), τα RNNs διαθέτουν εσωτερικούς βρόχους που τους επιτρέπουν να διατηρούν μια μορφή «μνήμης» από προηγούμενες εισόδους. Αυτό επιτυγχάνεται μέσω μιας εσωτερικής κρυφής κατάστασης (hidden state), η οποία ενημερώνεται σε κάθε χρονικό βήμα, καθιστώντας τα ιδανικά για εφαρμογές όπου το περιεχόμενο και η σειρά των δεδομένων είναι κρίσιμης σημασίας, όπως η επεξεργασία φυσικής γλώσσας, η ανάλυση ήχου και η πρόβλεψη χρονοσειρών.

Η βασική αρχιτεκτονική ενός RNN επεξεργάζεται σε κάθε βήμα την τρέχουσα είσοδο μαζί με την κρυφή κατάσταση του προηγούμενου βήματος. Ο συνδυασμός αυτός μετασχηματίζεται για να παραγάγει μια ενημερωμένη κρυφή κατάσταση και, προαιρετικά, μια έξοδο. Με αυτόν τον τρόπο, η κρυφή κατάσταση λειτουργεί ως μια δυναμική σύνοψη του ιστορικού της ακολουθίας, επιτρέποντας στο δίκτυο να ανιχνεύει χρονικές εξαρτήσεις.

Παρά την κυριαρχία των αρχιτεκτονικών τύπου Transformer σε πολλές σύγχρονες εφαρμογές, τα RNNs παραμένουν εξαιρετικά χρήσιμα σε εξειδικευμένους τομείς, ιδιαίτερα όπου τα δεδομένα επεξεργάζονται τμηματικά ή όπου η υπολογιστική αποδοτικότητα είναι πιο σημαντική από την απόλυτη απόδοση.

3.3.4 Μετασχηματιστές (Transformers)

Η αρχιτεκτονική Transformer, που παρουσιάστηκε το 2017 [8], σηματοδότησε μια θεμελιώδη αλλαγή στα μοντέλα επεξεργασίας ακολουθιών (όπως η μηχανική μετάφραση), αντικαθιστώντας τα παραδοσιακά αναδρομικά (RNN) και συνελικτικά (CNN) δίκτυα. Η βασική καινοτομία του Transformer είναι ότι αποφεύγει πλήρως την αναδρομή (recurrence) και βασίζεται αποκλειστικά σε μηχανισμούς προσοχής (attention mechanisms). Ο πυρήνας της αρχιτεκτονικής είναι η αυτο-προσοχή (self-attention), ένας μηχανισμός που επιτρέπει στο μοντέλο να σταθμίζει δυναμικά τη σημασία διαφορετικών λέξεων ή τμημάτων της ακολουθίας εισόδου, ανεξάρτητα από την απόστασή τους. Αυτή η προσέγγιση επιλύει το κρίσιμο πρόβλημα των μακρινών εξαρτήσεων (long-range dependencies) και, ταυτόχρονα, επιτρέπει πολύ μεγαλύτερη παραλληλοποίηση κατά την εκπαίδευση σε σχέση με τα σειριακά RNNs.
Ο μετασχηματιστής διατηρεί την κλασική δομή κωδικοποιητή-αποκωδικοποιητή (encoder-decoder). Τόσο ο κωδικοποιητής όσο και ο αποκωδικοποιητής αποτελούνται από μια στοίβα πολλαπλών πανομοιότυπων στρωμάτων. Κάθε ένα από αυτά τα στρώματα περιέχει δύο βασικά υπο-στρώματα: έναν μηχανισμό Προσοχής Πολλαπλών Κεφαλών (Multi-Head Attention) και ένα απλό, τοπικό νευρωνικό δίκτυο τροφοδότησης (Position-wise Feed-Forward Network). Ο αποκωδικοποιητής διαθέτει επίσης ένα τρίτο υπο-στρώμα που εκτελεί προσοχή πάνω στην έξοδο του κωδικοποιητή. Δεδομένου ότι η αρχιτεκτονική δεν επεξεργάζεται τις λέξεις σειριακά, δεν έχει εγγενή αντίληψη της σειράς. Για να αντιμετωπιστεί αυτό, ο Transformer εισάγει Κωδικοποιήσεις Θέσης (Positional Encodings), οι οποίες προστίθενται στα embeddings της εισόδου για να παρέχουν στο μοντέλο πληροφορίες σχετικά με τη σειρά των στοιχείων στην ακολουθία.
Η επιτυχία του Transformer απέδειξε ότι οι μηχανισμοί προσοχής από μόνοι τους είναι επαρκείς για την επίτευξη υψηλών επιδόσεων σε εργασίες με ακολουθίες. Τα πλεονεκτήματα της αυτο-προσοχής (ικανότητα μοντελοποίησης μακρινών εξαρτήσεων με σταθερό αριθμό λειτουργιών και υψηλός βαθμός παραλληλισμού) την κατέστησαν κυρίαρχη. Η αρχιτεκτονική αυτή δεν αποτέλεσε απλώς μια βελτίωση, αλλά τη βάση πάνω στην οποία χτίστηκαν τα σύγχρονα μεγάλα γλωσσικά μοντέλα (LLMs), όπως το BERT (που βασίζεται στον κωδικοποιητή) και το GPT (που βασίζεται στον αποκωδικοποιητή).

4 Επεξεργασία φυσικής γλώσσας

4.1 Εισαγωγή στην Επεξεργασία Φυσικής Γλώσσας (NLP)

Το βασικό ζήτημα που έρχεται να αντιμετωπίσει η Επεξεργασία Φυσικής Γλώσσας είναι η κατανόηση της ανθρώπινης γλώσσας από τους υπολογιστές. Επειδή οι υπολογιστές «καταλαβαίνουν» μόνο δυαδικό κώδικα, η επικοινωνία μαζί τους μέσω της φυσικής γλώσσας κατέστη εφικτή χάρη στο NLP. Πρόκειται για έναν ουσιαστικό «συνδετικό κρίκο» ανάμεσα στην ανθρώπινη επικοινωνία και την κατανόηση από τις μηχανές, επιτρέποντας στους υπολογιστές να ερμηνεύουν, να παράγουν και να μαθαίνουν από τη γλώσσα, ώστε να εκτελούν χρήσιμες λειτουργίες. Είναι η τεχνολογία που βρίσκεται πίσω από πολλές από τις «έξυπνες» εφαρμογές που χρησιμοποιούμε καθημερινά, καθιστώντας την αλληλεπίδρασή μας με τις μηχανές πιο φυσική, αποδοτική και ουσιαστική.

4.2 Ορισμός της Επεξεργασίας Φυσικής Γλώσσας (NLP)

Η Επεξεργασία Φυσικής Γλώσσας [9] είναι κλάδος της Τεχνητής Νοημοσύνης (AI), ο οποίος δίνει τη δυνατότητα στους υπολογιστές να «διαβάζουν», να κατανοούν, να ερμηνεύουν και να παράγουν ανθρώπινη γλώσσα. Συνδυάζει τη γλωσσολογία με στατιστικά μοντέλα, μηχανική μάθηση και βαθιά μάθηση. Ο κύριος στόχος της είναι να διευκολύνει την επεξεργασία και ανάλυση μεγάλων ποσοτήτων γλωσσικών δεδομένων, εξάγοντας χρήσιμα συμπεράσματα και διευκολύνοντας την αβίαστη επικοινωνία ανθρώπου-μηχανής. Με πιο απλά λόγια, το NLP είναι η τεχνολογία που επιτρέπει σε λογισμικά να καταλαβαίνουν τι λέμε ή γράφουμε, αντιλαμβανόμενα το νόημα και την πρόθεση, όπως θα έκανε ένας άνθρωπος.

4.3 Η συμβολοποίηση στην Επεξεργασία Φυσικής Γλώσσας (NLP)

4.3.1 Ορισμός και Σημασία

Η συμβολοποίηση (tokenization) [10] αποτελεί ένα από τα βασικότερα στάδια στη διαδικασία της Επεξεργασίας Φυσικής Γλώσσας (Natural Language Processing – NLP). Πρόκειται για τον διαχωρισμό του ακατέργαστου κειμένου σε μικρότερες μονάδες, τα λεγόμενα tokens. Τα tokens μπορεί να είναι λέξεις, υπολέξεις, χαρακτήρες ή ακόμη και ολόκληρες προτάσεις, και αποτελούν τις βασικές δομικές μονάδες πάνω στις οποίες στηρίζονται οι επόμενες φάσεις επεξεργασίας.
Κάθε token θεωρείται ως μια σημασιολογική μονάδα, η οποία μπορεί να αναλυθεί, να επεξεργαστεί ή να μετατραπεί σε χαρακτηριστικά που αξιοποιούνται από αλγορίθμους μηχανικής μάθησης. Ο ρόλος της συμβολοποίησης είναι καθοριστικός, καθώς διευκολύνει τη διάσπαση του μη δομημένου κειμένου σε διαχειρίσιμα μέρη και θέτει τα θεμέλια για την αποτελεσματική λειτουργία μεγάλων γλωσσικών μοντέλων (LLMs).

4.3.2 Είδη Συμβολοποίησης

Ανάλογα με τη γλώσσα και το εκάστοτε υπολογιστικό έργο, εφαρμόζονται διαφορετικές προσεγγίσεις. Οι πιο συνηθισμένες μορφές είναι οι εξής:

α) Συμβολοποίηση σε λέξεις (Word Tokenization)

Η πιο απλή μέθοδος, η οποία χωρίζει το κείμενο με βάση τα κενά ή τα σημεία στίξης. Είναι ιδιαίτερα αποτελεσματική σε γλώσσες που χρησιμοποιούν σαφή διαχωριστικά μεταξύ των λέξεων, όπως τα Αγγλικά.
Παράδειγμα:
"Natural Language Processing" → ["Natural", "Language", "Processing"]

β) Συμβολοποίηση σε χαρακτήρες (Character Tokenization)

Διαχωρίζει το κείμενο σε μεμονωμένους χαρακτήρες. Χρησιμοποιείται συχνά για εργασίες όπως η διόρθωση ορθογραφικών λαθών, για γλώσσες χωρίς ξεκάθαρα όρια μεταξύ λέξεων ή για περιπτώσεις με περιορισμένο λεξιλόγιο. Ωστόσο, η μέθοδος αυτή παράγει πολύ

μεγαλύτερες ακολουθίες και οδηγεί σε απώλεια υψηλότερων σημασιολογικών δομών.
Παράδειγμα:
"hello" → ["h", "e", "l", "l", "o"]

γ) Συμβολοποίηση σε υπολέξεις (Subword Tokenization)

Διαχωρίζει τις λέξεις σε μικρότερες μονάδες, μεγαλύτερες από χαρακτήρες αλλά μικρότερες από λέξεις. Χρησιμοποιεί αλγόριθμους όπως Byte-Pair Encoding (BPE), WordPiece ή SentencePiece. Η προσέγγιση αυτή επιτρέπει την αποδοτικότερη αναπαράσταση σπάνιων ή σύνθετων λέξεων.
Παράδειγμα:
"unhappiness" → ["un", "happi", "ness"] (ανάλογα με το εκάστοτε μοντέλο εκπαίδευσης).

δ) Συμβολοποίηση σε προτάσεις (Sentence Tokenization)

Διαχωρίζει το κείμενο σε προτάσεις, κάτι που είναι ιδιαίτερα χρήσιμο σε αναλύσεις κειμένων μεγάλης κλίμακας ή σε συστήματα που απαιτούν κατανόηση της πρότασης ως αυτόνομης σημασιολογικής ενότητας.
Παράδειγμα:
"This is one sentence. This is another." → ["This is one sentence.", "This is another."]

4.4 Προεπεξεργασία Κειμένου

Πριν ένα κείμενο αξιοποιηθεί από ένα μοντέλο μηχανικής μάθησης, πρέπει να καθαριστεί και να ομογενοποιηθεί. Η διαδικασία αυτή, γνωστή ως προεπεξεργασία κειμένου, μετατρέπει το μη δομημένο κείμενο σε πιο οργανωμένη μορφή. Τρεις βασικές τεχνικές είναι η διαχείριση των stopwords, το stemming και η lemmatization.

Stopwords

Σε κάθε γλώσσα υπάρχουν λέξεις που εμφανίζονται πολύ συχνά αλλά από μόνες τους δεν προσθέτουν ουσιαστική σημασία. Τέτοιες είναι τα άρθρα (π.χ. «το», «η»), οι σύνδεσμοι (π.χ.

«και», «αλλά») και οι προθέσεις (π.χ. «σε», «με»). Ονομάζονται stopwords [11]Η αφαίρεσή τους συμβάλλει στη μείωση της πολυπλοκότητας των δεδομένων και στην εστίαση στις λέξεις που φέρουν μεγαλύτερο σημασιολογικό βάρος. Για παράδειγμα, από τη φράση *«The quick brown fox jumps over the lazy dog»* μένουν οι λέξεις: *[“quick”, “brown”, “fox”, “jumps”, “lazy”, “dog”]*.
Ωστόσο, η απομάκρυνση stopwords δεν είναι πάντα επωφελής. Σε εργασίες όπως η ανάλυση συναισθήματος ή η αυτόματη μετάφραση, οι «μικρές» αυτές λέξεις μπορεί να παίζουν καθοριστικό ρόλο στη διατήρηση του νοήματος και της ροής του κειμένου.

Stemming

Το stemming [12] είναι μια απλοϊκή μέθοδος που αποκόπτει καταλήξεις από τις λέξεις, ώστε να τις μειώσει σε μια κοινή ρίζα (stem). Η ρίζα αυτή συχνά δεν αντιστοιχεί σε υπαρκτή λέξη, αλλά λειτουργεί ως ενδιάμεση μορφή για την ομαδοποίηση όρων. Ο αλγόριθμος εφαρμόζει απλούς κανόνες (π.χ. αφαίρεση καταλήξεων όπως *-ing, -ed, -s*) χωρίς να λαμβάνει υπόψη το συμφραζόμενο.
Ένας από τους πιο γνωστούς και ευρέως χρησιμοποιούμενους αλγόριθμους είναι ο Porter Stemmer, που ξεχωρίζει για την απλότητα και την ταχύτητά του.

Παραδείγματα:

- *“jumps”, “jumping”, “jumped”* → *“jump”*
- *“running”, “runner”, “runs”* → *“run”*
- *“university”, “universal”* → *“univers”* (μη υπαρκτή λέξη)

Το stemming είναι αποτελεσματικό για τη γρήγορη μείωση της πολυπλοκότητας και την ενοποίηση όρων, ωστόσο μπορεί να οδηγήσει σε παραμορφώσεις ή απώλεια νοήματος λόγω της μη ακριβούς προσέγγισής του.

Lemmatization

Το lemmatization [13], [14] είναι μια πιο εξελιγμένη και υπολογιστικά απαιτητική διαδικασία σε σχέση με το stemming. Στόχος της είναι η μείωση μιας λέξης στη βασική της μορφή, γνωστή ως λήμμα (lemma), το οποίο αποτελεί υπαρκτή λέξη του λεξικού.
Σε αντίθεση με το stemming,το lemmatization λαμβάνει υπόψη το συμφραζόμενο και το μέρος του λόγου (ρήμα, ουσιαστικό κ.λπ.) στο οποίο ανήκει η λέξη. Βασίζεται σε εκτενή λεξιλόγια και μορφολογική ανάλυση, ώστε να επιστρέψει την κανονική μορφή της λέξης.

Παραδείγματα:

- *“better” (επίθετο) → “good”*
- *“running” (ρήμα) → “run”*
- *“meeting” ως ουσιαστικό → “meeting”*, ενώ *ως ρήμα → “meet”*

Το lemmatization είναι πιο ακριβές από το stemming, καθώς διατηρεί το σωστό νόημα, αλλά είναι πιο αργή και απαιτεί περισσότερους γλωσσολογικούς πόρους, όπως εργαλεία αναγνώρισης μέρους του λόγου (POS taggers), που αναφέρεται παρακάτω.

4.5 Επισήμανση Μερών του Λόγου

Η επισήμανση μερών του λόγου (Part-of-Speech Tagging) είναι η διαδικασία κατά την οποία σε κάθε λέξη μιας πρότασης αποδίδεται ο γραμματικός της ρόλος (όπως ουσιαστικό, ρήμα, επίθετο, επίρρημα), με βάση το νόημα και τα συμφραζόμενα. Για παράδειγμα, στην πρόταση «The quick brown fox jumps over the lazy dog», οι λέξεις χαρακτηρίζονται ως άρθρο, επίθετο, ουσιαστικό, ρήμα κ.λπ. Αυτή η διαδικασία είναι θεμελιώδης στη Φυσική Επεξεργασία Γλώσσας (NLP), καθώς δίνει δομή στο ακατέργαστο κείμενο και υποστηρίζει πιο σύνθετες εργασίες, όπως η συντακτική ανάλυση, η μηχανική μετάφραση και η αυτόματη απάντηση σε ερωτήσεις.

4.6 Αναγνώριση Ονομαστικών Οντοτήτων (NER)

Η Αναγνώριση Ονομαστικών Οντοτήτων (Named Entity Recognition[15], [16] είναι βασικό μέρος στην Επεξεργασία Φυσικής Γλώσσας (NLP) που εντοπίζει και κατηγοριοποιεί οντότητες σε κείμενο, όπως πρόσωπα, τοποθεσίες, οργανισμούς, ημερομηνίες και προϊόντα. Με αυτόν τον τρόπο, επιτρέπει την εξαγωγή ουσιαστικής πληροφορίας από μη δομημένα δεδομένα, διευκολύνοντας εφαρμογές όπως η εξαγωγή πληροφορίας, τα chatbots και η δημιουργία knowledge graphs.
Οι πρώτες προσεγγίσεις βασίζονταν σε κανόνες και λεξικά, αλλά σήμερα κυριαρχούν οι μέθοδοι μηχανικής και βαθιάς μάθησης, όπως τα transformer-based μοντέλα (π.χ. BERT), που λαμβάνουν υπόψη το συμφραζόμενο και τις σχέσεις των λέξεων. Έτσι, η NER μετατρέπει το

αδόμητο κείμενο σε αξιοποιήσιμη γνώση για ανάλυση και αλληλεπίδραση με μεγάλους όγκους δεδομένων.

4.7 Μέθοδοι Αναπαράστασης Χαρακτηριστικών

Μια σημαντική λειτουργία της Επεξεργασίας Φυσικής Γλώσσας είναι η αναπαράσταση λέξεων, κομματιών λέξεων ή και φράσεων σε μορφή που αναγνωρίζουν οι μηχανές. Παρακάτω θα παρουσιάσουμε κάποιες από τις μεθόδους αυτής της αναπαράστασης.

Η **One-Hot Encoding** [17], [18]αποτελεί μια βασική μέθοδο αναπαράστασης κατηγορικών δεδομένων, κατά την οποία κάθε ξεχωριστή κατηγορία μετατρέπεται σε μία δυαδική στήλη (0/1). Για κάθε δείγμα, μόνο μία στήλη φέρει την τιμή 1, ενώ όλες οι υπόλοιπες παραμένουν μηδενικές. Με τον τρόπο αυτό, τα μοντέλα αποφεύγουν να αποδώσουν ψευδή ιεραρχία μεταξύ των κατηγοριών, όπως θα συνέβαινε αν χρησιμοποιούνταν απλές αριθμητικές τιμές. Έτσι, για παράδειγμα, τα χρώματα "κόκκινο", "μπλε" και "πράσινο" αναπαρίστανται ισότιμα, χωρίς να υπονοείται ότι κάποιο έχει μεγαλύτερη "αξία" από τα υπόλοιπα.

Η μέθοδος **Bag of Words (BoW)** [19], [20] χρησιμοποιείται κυρίως στην επεξεργασία φυσικής γλώσσας. Αναπαριστά κάθε κείμενο με βάση τη συχνότητα εμφάνισης των λέξεών του, χωρίς να λαμβάνει υπόψη τη σειρά τους. Με αυτόν τον τρόπο, δημιουργείται ένα "σακί" λέξεων όπου η σημασία προκύπτει από το πόσες φορές εμφανίζεται κάθε όρος. Η μέθοδος αυτή λύνει τον περιορισμό του One-Hot Encoding, καθώς επιτρέπει τη σύγκριση και κατανόηση περιεχομένου κειμένων με βάση τις λέξεις που περιέχουν. Ωστόσο, η βασική της αδυναμία είναι ότι αγνοεί τα συμφραζόμενα και τη σειρά με την οποία εμφανίζονται οι λέξεις.
Για να αντιμετωπιστεί αυτός ο περιορισμός, χρησιμοποιούνται τα **n-grams** [21], [22], τα οποία λαμβάνουν υπόψη ακολουθίες n λέξεων (π.χ. bigrams για ζεύγη λέξεων, trigrams για τριάδες κ.ο.κ.). Με αυτόν τον τρόπο, διατηρείται μέρος της πληροφορίας της σειράς, κάτι που επιτρέπει την καλύτερη κατανόηση φράσεων ή εκφράσεων με ειδικό νόημα, όπως "Νέα Υόρκη". Ωστόσο, η μέθοδος αυτή αυξάνει σημαντικά τον αριθμό των χαρακτηριστικών και μπορεί να παράγει πολλές άχρηστες ή σπάνιες συνδυαστικές μορφές, οδηγώντας σε μεγαλύτερη πολυπλοκότητα.

Τέλος, η μέθοδος **TF-IDF (Term Frequency – Inverse Document Frequency)** [23] επιδιώκει να βελτιώσει το Bag of Words με το να αποδίδει διαφορετικά βάρη στις λέξεις. Συγκεκριμένα, η συχνότητα εμφάνισης μιας λέξης σε ένα κείμενο (TF) συνδυάζεται με τον αντίστροφο βαθμό συχνότητας εμφάνισής της σε ολόκληρη τη συλλογή κειμένων (IDF). Έτσι, λέξεις που εμφανίζονται πολύ συχνά σε όλα τα κείμενα, όπως τα άρθρα ή οι σύνδεσμοι ("και", "το", "είναι"), λαμβάνουν χαμηλό βάρος, ενώ πιο σπάνιες και πληροφοριακές λέξεις αναδεικνύονται. Με τον τρόπο αυτό, το TF-IDF λύνει τον περιορισμό του BoW που τείνει να δίνει την ίδια βαρύτητα σε όλες τις λέξεις, ανεξάρτητα από τη σημασία τους.

Ενσωματώσεις Λέξεων (Word Embeddings)

Οι Ενσωματώσεις Λέξεων (Word Embeddings) [24], [25] αποτελούν σύγχρονη μέθοδο αναπαράστασης λέξεων στην Επεξεργασία Φυσικής Γλώσσας (NLP). Σε αντίθεση με παραδοσιακές τεχνικές όπως το bag-of-words και το TF-IDF, που αντιμετωπίζουν τις λέξεις ως ανεξάρτητα σύμβολα χωρίς να αποτυπώνουν νοηματικές σχέσεις, οι ενσωματώσεις λέξεων απεικονίζουν κάθε λέξη ως διάνυσμα σε πολυδιάστατο συνεχές χώρο. Έτσι, λέξεις με παρόμοιο νόημα βρίσκονται γεωμετρικά πιο κοντά μεταξύ τους (π.χ. *king* και *queen*). Αυτή η ιδιότητα επιτρέπει στα μοντέλα μηχανικής μάθησης να αξιοποιούν τις σημασιολογικές κανονικότητες της γλώσσας και να βελτιώνουν την απόδοσή τους σε εφαρμογές όπως ανάλυση συναισθήματος, μηχανική μετάφραση και ανάκτηση πληροφορίας.

Το Word2Vec [26], [27] αποτελεί μία από τις πιο σημαντικές τεχνικές στην Επεξεργασία Φυσικής Γλώσσας (NLP) για την αναπαράσταση λέξεων σε μορφή διανυσμάτων (embeddings). Επέτρεψε την δημιουργία «πυκνών» και χαμηλών διαστάσεων αναπαραστάσεων, οι οποίες συλλαμβάνουν τόσο τις συντακτικές όσο και τις σημασιολογικές σχέσεις μεταξύ των λέξεων. Σε αντίθεση με τα one-hot vectors, που δεν εμπεριέχουν πληροφορία για τη σημασία, το Word2Vec οργανώνει τις λέξεις σε έναν πολυδιάστατο χώρο, όπου οι λέξεις με παρόμοιο νόημα βρίσκονται κοντά.

Η βασική αρχή του Word2Vec είναι ότι η σημασία μιας λέξης μπορεί να συναχθεί από τα συμφραζόμενά της. Με βάση αυτήν τη λογική, προτάθηκαν δύο διαφορετικές νευρωνικές αρχιτεκτονικές: το Continuous Bag-of-Words (CBOW) και το Skip-gram.

CBOW

Το CBOW επιχειρεί να προβλέψει τη λέξη-στόχο με βάση τα συμφραζόμενα. Οι λέξεις του περιβάλλοντος εισάγονται στο δίκτυο ως one-hot vectors, συνδυάζονται σε μια ενιαία διανυσματική αναπαράσταση και χρησιμοποιούνται για την πρόβλεψη της κεντρικής λέξης. Το πλεονέκτημά του είναι η αποδοτικότητα και η ταχύτητα εκπαίδευσης, γεγονός που το καθιστά κατάλληλο για μικρότερα σύνολα δεδομένων και συχνές λέξεις. Ωστόσο, αυτή η μέθοδος μπορεί να περιορίσει την δυνατότητα να αναγνωρίζονται πληροφορίες από σπάνιες αλλά σημαντικές λέξεις.

Skip-gram

Αντίθετα, το Skip-gram αναστρέφει τη διαδικασία: χρησιμοποιεί τη λέξη-στόχο για να προβλέψει τα συμφραζόμενα. Κάθε συνδυασμός λέξης-στόχου και λέξης περιβάλλοντος λειτουργεί ως ξεχωριστό εκπαιδευτικό παράδειγμα, γεγονός που το καθιστά πιο αποτελεσματικό στη μάθηση αναπαραστάσεων για σπάνιες λέξεις και στην αποτύπωση λεπτών σημασιολογικών σχέσεων. Το Skip-gram συχνά αποδίδει καλύτερα σε σημασιολογικά έργα (π.χ. αναλογίες λέξεων), αλλά απαιτεί περισσότερο χρόνο και μεγαλύτερο όγκο δεδομένων για εκπαίδευση.

Αφού μετατραπούν οι λέξεις σε ενσωματώσεις, μπορούν να βρεθούν σημασιολογικά κοντινές λέξεις με απλές μεθόδους σύγκρισης αποστάσεων διανυσμάτων, όπως ευκλείδεια απόσταση, απόσταση συνημιτόνου ή εσωτερικό γινόμενο.

5 Γενετική Τεχνητή Νοημοσύνη

5.1 Εισαγωγή στην Γενετική Τεχνητή Νοημοσύνη

Η Γενετική Τεχνητή Νοημοσύνη (Generative AI) είναι ένας τομέας της μηχανικής μάθησης που επικεντρώνεται στη δημιουργία νέου, πρωτότυπου περιεχομένου, όπως κείμενα, εικόνες, μουσική ή κώδικας, μέσω της εκμάθησης προτύπων από μεγάλα σύνολα δεδομένων. Σε αντίθεση με την παραδοσιακή ΤΝ, που αναλύει δεδομένα και κάνει προβλέψεις, η γενετική ΤΝ παράγει δημιουργικές εξόδους που μοιάζουν με ανθρώπινα έργα. Βασίζεται κυρίως σε βαθιά νευρωνικά δίκτυα, όπως τα Generative Adversarial Networks (GANs) και τα μοντέλα τύπου Transformer, και έχει σημειώσει σημαντική πρόοδο σε τομείς όπως η επεξεργασία φυσικής γλώσσας, η υπολογιστική όραση και οι δημιουργικές βιομηχανίες. Στο κεφάλαιο αυτό αναλύεται η Γενετική ΤΝ, με έμφαση στα Μεγάλα Γλωσσικά Μοντέλα (LLMs), τα οποία αποτελούν το βασικό εργαλείο της εφαρμογής που δημιουργήθηκε για την παρούσα διπλωματική εργασία.

5.2 Μεγάλα Γλωσσικά Μοντέλα

Τα Μεγάλα Γλωσσικά Μοντέλα (Large Language Models – LLMs) [28] αποτελούν εξέλιξη της αρχιτεκτονικής των μετασχηματιστών (transformers) και κυρίως στη μορφή *decoder-only*. Το χαρακτηριστικό τους στοιχείο είναι η κλίμακα: τεράστιος αριθμός παραμέτρων, όγκος δεδομένων εκπαίδευσης και τεράστια υπολογιστική ισχύς. Η διεύρυνση αυτή δεν είναι απλώς ποσοτική, αλλά οδηγεί σε νέες, αναδυόμενες δυνατότητες που δεν εμφανίζονται σε μικρότερα μοντέλα.

Η κατασκευή ενός LLM στηρίζεται σε τρεις πυλώνες:

1. **Αρχιτεκτονική**: κυριαρχεί το *decoder-only* μοντέλο, ιδανικό για παραγωγή κειμένου και εφαρμογές όπως συνομιλία, συγγραφή ή προγραμματισμός.

2. **Δεδομένα**: η εκπαίδευση βασίζεται σε τρισεκατομμύρια tokens από ποικίλες πηγές (διαδίκτυο, βιβλία, άρθρα, κώδικας). Η ποιότητα και η καθαρότητα των δεδομένων είναι κρίσιμες.

3. **Εκπαίδευση**: περιλαμβάνει δύο φάσεις. Στην *προ-εκπαίδευση*, το μοντέλο μαθαίνει τα βασικά στατιστικά της γλώσσας και αναπτύσσει "γνώση" για τον κόσμο. Στην

εξατομίκευση (fine-tuning), μέσω τεχνικών όπως το *instruction tuning*, προσαρμόζεται ώστε να είναι χρήσιμο, ασφαλές και συνεπές.

Η ευθυγράμμιση με τις ανθρώπινες αξίες επιτυγχάνεται με το Reinforcement Learning from Human Feedback (RLHF) [3], [4], μια διαδικασία πολλαπλών σταδίων που συνδυάζει ανθρώπινες υποδείξεις, εκπαίδευση μοντέλου ανταμοιβής και ενίσχυση μέσω αλγορίθμων. Στόχος είναι η αποφυγή επιβλαβών ή ακατάλληλων απαντήσεων και η παραγωγή αποτελεσμάτων που ικανοποιούν τις προτιμήσεις των χρηστών.

Τα LLMs ξεχωρίζουν για δυνατότητες όπως: in-context learning (εκμάθηση από παραδείγματα μέσα στην προτροπή), εκτέλεση πολύπλοκων οδηγιών, αλυσιδωτή συλλογιστική (chain-of-thought reasoning) και παραγωγή/κατανόηση κώδικα. Ωστόσο, συνοδεύονται από σημαντικούς περιορισμούς:

- Παράγουν πειστικό αλλά ψευδές περιεχόμενο (*hallucinations*).
- Δεν έχουν πραγματική κατανόηση του κόσμου, καθώς λειτουργούν μόνο πάνω σε στατιστικά πρότυπα.
- Ενσωματώνουν κοινωνικές προκαταλήψεις που υπάρχουν στα δεδομένα τους.
- Έχουν υψηλό υπολογιστικό και περιβαλλοντικό κόστος.

Ανακεφαλαιώνοντας, τα LLMs συνιστούν μια τομή στην Τεχνητή Νοημοσύνη, καθώς μετατρέπουν τη γλώσσα σε πεδίο δυναμικής αλληλεπίδρασης με μηχανές. Είναι ιδιαίτερα χρήσιμα εργαλεία αλλά παραμένουν στατιστικά μοντέλα με όρια και αδυναμίες, και η αξιοποίησή τους απαιτεί κριτική επίγνωση των δυνατοτήτων και περιορισμών τους.

5.3 Τελική Πρόσβαση Χρηστών στα Μεγάλα Γλωσσικά Μοντέλα

Η ανάπτυξη μεγάλων γλωσσικών μοντέλων (LLMs) αποτελεί μια ιδιαίτερα απαιτητική διαδικασία, που προϋποθέτει τεράστιους υπολογιστικούς πόρους και σημαντική οικονομική επένδυση. Γι’ αυτό τον λόγο, την κατασκευή και βελτιστοποίησή τους αναλαμβάνουν κυρίως μεγάλες οργανώσεις με την απαραίτητη υποδομή και κεφάλαια. Η πρόσβαση των χρηστών σε αυτά τα μοντέλα πραγματοποιείται μέσα από δύο βασικές κατηγορίες: τα ανοικτού κώδικα και τα κλειστού κώδικα LLMs.
Τα ανοικτού κώδικα LLMs (όπως τα LLaMA, Mistral ή Falcon) διατίθενται ελεύθερα στο κοινό, με δημοσιευμένα τα αρχιτεκτονικά τους στοιχεία και, σε ορισμένες περιπτώσεις, τα δεδομένα εκπαίδευσης. Η διαθεσιμότητά τους σε πλατφόρμες όπως το GitHub ή το Hugging

Face επιτρέπει σε ερευνητές, ακαδημαϊκούς και μικρές επιχειρήσεις να τα κατεβάσουν και να τα αξιοποιήσουν, εφόσον διαθέτουν τον κατάλληλο εξοπλισμό ή υπηρεσίες cloud. Η διαφάνεια αυτής της προσέγγισης ενισχύει την καινοτομία και τη συνεργασία, ενώ διευκολύνει την ανάπτυξη εξειδικευμένων εφαρμογών. Ωστόσο, η αξιοποίησή τους απαιτεί τεχνικές γνώσεις και σημαντικούς υπολογιστικούς πόρους, ενώ το κόστος λειτουργίας και οι κίνδυνοι κακής χρήσης αποτελούν σημαντικούς περιορισμούς.
Αντίθετα, τα κλειστού κώδικα LLMs (όπως τα ChatGPT, Claude ή Grok) παραμένουν ιδιοκτησία των εταιρειών που τα ανέπτυξαν και διατίθενται κυρίως ως υπηρεσίες μέσω APIs, διαδικτυακών εφαρμογών ή κινητών εφαρμογών. Η πρόσβαση συνήθως προσφέρεται μέσω συνδρομών ή μοντέλων χρέωσης ανά χρήση, ενώ παρέχονται και δωρεάν εκδόσεις με περιορισμούς. Αυτή η προσέγγιση δίνει έμφαση στην ευκολία χρήσης, μειώνοντας την ανάγκη για τεχνικές γνώσεις, ενώ η φιλοξενία σε ισχυρή υποδομή εξασφαλίζει αξιοπιστία και κλιμάκωση. Παράλληλα, ενσωματώνει μηχανισμούς ασφαλείας για τη μείωση επιβλαβών απαντήσεων. Ωστόσο, η έλλειψη διαφάνειας δυσκολεύει την κατανόηση πιθανών μεροληψιών, ενώ το κόστος συνδρομής και η εξάρτηση από έναν πάροχο εγείρουν ζητήματα ισότητας και προστασίας δεδομένων.

5.4 Σχεδιασμός Προτροπών (Prompt engineering)

Ο Σχεδιασμός Προτροπών (Prompt engineering) ορίζεται ως η συστηματική διαδικασία διαμόρφωσης ακριβών εντολών που επιτρέπουν την αποτελεσματική προσαρμογή και λειτουργία ενός μεγάλου γλωσσικού μοντέλου (LLM). Για να κατανοηθεί η πρακτική αυτή, είναι χρήσιμο να περιγραφεί η τυπική δομή εισόδων προς το μοντέλο σε συνομιλιακές εφαρμογές: τα μηνύματα οργανώνονται σε ακολουθία με διακριτούς ρόλους — System, User και Assistant — που καθορίζουν αντίστοιχα τους κανόνες/περιορισμούς, το αίτημα του ανθρώπου και την απόκριση του μοντέλου.

Zero-shot prompting [29], [30]: η απλούστερη προσέγγιση, όπου η προτροπή δίνεται χωρίς παραδείγματα και το μοντέλο βασίζεται αποκλειστικά στην προεκπαίδευσή και την έμφυτη ικανότητα γενίκευσης. Το πλεονέκτημα αυτής της προσέγγισης είναι η απλότητα και το χαμηλό κόστος σύνταξης. Το μειονέκτημα είναι ότι είναι ευπαθές σε πολύπλοκα, πολυβηματικά ή ειδικά αιτήματα, όπου το μοντέλο μπορεί να κάνει λανθασμένες υποθέσεις.

One-shot prompting [30]: περιλαμβάνει ένα ενδεικτικό παράδειγμα εισόδου–εξόδου μέσα στο prompt, προσφέροντας καλύτερη κατανόηση της εργασίας που δίνεται στο μοντέλο. Είναι χρήσιμο όταν χρειάζεται συγκεκριμένο σχήμα αλλά το πρόβλημα δεν είναι υπερβολικά σύνθετο. Ο περιορισμός είναι ότι ένα μόνο παράδειγμα συχνά δεν καλύπτει όλα τα όρια ή τις ιδιαιτερότητες.

Few-shot prompting: παρέχει πολλά παραδείγματα ώστε το μοντέλο να μάθει το πρότυπο, τη στρατηγική σκέψης ή το ύφος. Αυξάνει σημαντικά την αξιοπιστία σε σύνθετες εργασίες (πολλαπλές κατηγορίες, μετασχηματισμοί κειμένου, συγκεκριμένο ύφος). Το κόστος είναι η χρήση μεγάλου μέρους του context window και η ανάγκη επιμερούς επιλογής των παραδειγμάτων.

Negative prompting [31]: ο χρήστης δηλώνει ρητά τι πρέπει να αποφευχθεί στο αποτέλεσμα (στυλιστικά στοιχεία, ευαίσθητο περιεχόμενο, κλισέ). Χρησιμεύει στον έλεγχο περιεχομένου και στην απομάκρυνση ανεπιθύμητων μοτίβων, αλλά μπορεί να έρχεται σε σύγκρουση με τον κύριο στόχο ή να είναι λιγότερο αποτελεσματικό έναντι ισχυρών προκαταλήψεων του μοντέλου.

Iterative prompting : μεθοδολογία επαναληπτικού σχεδιασμού όπου το αρχικό prompt αξιολογείται, διορθώνεται και βελτιώνεται μέσα από κύκλους μέχρι να επιτευχθεί η επιθυμητή ποιότητα. Είναι πρακτική για ρεαλιστικές, δύσκολα ορισμένες εργασίες, απαιτεί όμως χρόνο και αναλυτική ικανότητα από τον χρήστη.

Prompt chaining: η αποσύνθεση ενός σύνθετου έργου σε διαδοχικά υπο-βήματα, όπου κάθε βήμα τροφοδοτεί το επόμενο, δημιουργώντας αλληλουχία επεξεργασίας. Μειώνει το γνωστικό φορτίο σε κάθε βήμα και περιορίζει συσσωρευτικά σφάλματα, αλλά συνήθως απαιτεί εξωτερική ορχήστρωση (κώδικα) για τη ροή πληροφοριών.

Chain-of-Thought (CoT) prompting [32]: τεχνική που προτρέπει το μοντέλο να παράγει ρητή, βηματική αιτιολόγηση πριν ή μαζί με την απάντηση, είτε μέσω παραδειγμάτων είτε με οδηγίες όπως «ας σκεφτούμε βήμα-βήμα». Υπερέχει σε αριθμητικά και λογικά προβλήματα καθώς και σε σύνθετη συλλογιστική, με το κόστος όμως της ανάγκης για καλά σχεδιασμένα παραδείγματα και αυξημένων υπολογιστικών πόρων.

Η επιλογή τεχνικής εξαρτάται από τη φύση και την πολυπλοκότητα του έργου, τους περιορισμούς του context window και την ανάγκη για έλεγχο ή επεξήγηση. Για απλά αιτήματα αρκούν zero- ή one-shot προσεγγίσεις, ενώ για πολύπλοκες ή πολυβηματικές εργασίες συνιστώνται few-shot, CoT, prompt chaining και iterative prompting.

5.5 Χρησιμότητα των Μεγάλων Γλωσσικών Μοντέλων

Τα Μεγάλα Γλωσσικά Μοντέλα (LLMs) αποτελούν σημαντική πρόοδο στην τεχνητή νοημοσύνη, με εφαρμογές που μετασχηματίζουν πολλούς τομείς. Στην εκπαίδευση, προσφέρουν εξατομικευμένη μάθηση και υποστηρίζουν τους εκπαιδευτικούς με δημιουργία υλικού. Στην υγεία, διευκολύνουν τη διάγνωση, την επικοινωνία με ασθενείς και την έρευνα, αν και απαιτείται προσεκτική ενσωμάτωση. Στον επιχειρηματικό και χρηματοοικονομικό τομέα, βελτιώνουν την αποδοτικότητα με ανάλυση κινδύνων, ανίχνευση απάτης και αυτοματοποίηση διαδικασιών. Στην επιστημονική έρευνα, επιταχύνουν τη συλλογή και ανάλυση δεδομένων, αλλά χρειάζονται ανθρώπινη επίβλεψη. Στη δημιουργική βιομηχανία ενισχύουν την παραγωγή περιεχομένου και την εξατομίκευση εμπειριών, ενώ στο μάρκετινγκ και την εξυπηρέτηση πελατών προσφέρουν αυτοματοποιημένη και προσωποποιημένη αλληλεπίδραση. Καταλήγωντας, τα LLMs ανοίγουν νέους δρόμους καινοτομίας, με ταυτόχρονες προκλήσεις αξιοπιστίας και ηθικής χρήσης.

5.6 Περιορισμοί των Μεγάλων Γλωσσικών Μοντέλων

Τα Μεγάλα Γλωσσικά Μοντέλα (LLMs) έχουν φέρει επανάσταση στην τεχνητή νοημοσύνη και την επεξεργασία φυσικής γλώσσας, όμως παρουσιάζουν σοβαρούς περιορισμούς. Κατ' αρχάς, δεν διαθέτουν πραγματική κατανόηση ή λογική σκέψη, αλλά παράγουν κείμενο βάσει στατιστικών συσχετισμών, γεγονός που οδηγεί σε «παραισθήσεις» (ανακριβείς ή ψευδείς πληροφορίες). Επίσης, εμφανίζουν αστάθεια στη συλλογιστική, περιορισμένες ικανότητες αφηρημένης σκέψης, καθώς και έμφυτες προκαταλήψεις που αντικατοπτρίζουν τα δεδομένα εκπαίδευσής τους. Η γνώση τους είναι στατική και περιορίζεται χρονικά, ενώ η ανάπτυξη και λειτουργία τους απαιτούν τεράστιους υπολογιστικούς και περιβαλλοντικούς πόρους. Επιπλέον, λειτουργούν ως «μαύρα κουτιά» χωρίς επαρκή δυνατότητα εξήγησης των αποφάσεών τους. Τέλος, παρά τη φαινομενική τους ευφυΐα, στερούνται συνείδησης και πραγματικής κατανόησης, αποτελώντας εργαλεία παραγωγής γλώσσας και όχι νοήμονα όντα.

6 Retrieval Augmented Generation (RAG)

6.1 Εισαγωγή στο RAG

Η Retrieval-Augmented Generation (RAG) [33] αποτελεί μια σύγχρονη αρχιτεκτονική που συνδυάζει τα γλωσσικά μοντέλα μεγάλης κλίμακας με εξωτερικές βάσεις γνώσης, με στόχο την ενίσχυση της ακρίβειας, της επικαιρότητας και της αξιοπιστίας των παραγόμενων απαντήσεων. Σε αντίθεση με τα παραδοσιακά LLMs, τα οποία βασίζονται αποκλειστικά στη γνώση που έχει ενσωματωθεί κατά την εκπαίδευσή τους, το RAG επιτρέπει τη δυναμική αναζήτηση πληροφοριών από εξωτερικές πηγές πριν την παραγωγή κειμένου.

Η ανάγκη για τέτοιου τύπου αρχιτεκτονικές προκύπτει από τις εγγενείς αδυναμίες των LLMs: η γνώση τους σταματά στη χρονική στιγμή που ολοκληρώνεται η εκπαίδευση (knowledge cutoff), συχνά παράγουν «παραισθήσεις» δηλαδή ψευδείς αλλά πειστικές απαντήσεις, παρουσιάζουν έλλειψη εξειδίκευσης σε στενά ή εξειδικευμένα πεδία και λειτουργούν ως «μαύρα κουτιά» χωρίς δυνατότητα ανίχνευσης της προέλευσης των πληροφοριών.

Το RAG έρχεται να αντιμετωπίσει αυτά τα ζητήματα μέσω του συνδυασμού δύο ειδών γνώσης: της παραμετρικής, που εμπεριέχεται στο ίδιο το μοντέλο, και της μη παραμετρικής, που προέρχεται από εξωτερικές πηγές (π.χ. βάσεις δεδομένων ή σώματα κειμένων). Η διαδικασία θυμίζει «ανοιχτό βιβλίο»: το σύστημα χρησιμοποιεί έναν μηχανισμό ανάκτησης για να βρει σχετικά αποσπάσματα και τα ενσωματώνει στην είσοδο που δίνεται στο μοντέλο, ώστε η παραγόμενη απάντηση να είναι περισσότερο ακριβής, επίκαιρη και τεκμηριωμένη.

Με αυτόν τον τρόπο, το RAG ξεπερνά τις βασικές αδυναμίες των παραδοσιακών LLMs, μειώνοντας τις ψευδείς απαντήσεις, ενσωματώνοντας συνεχώς νέα γνώση, επιτρέποντας εξειδίκευση χωρίς επανεκπαίδευση και παρέχοντας δυνατότητα παραπομπής στις πηγές. Πρόκειται για μια αρχιτεκτονική που ανοίγει τον δρόμο για αξιόπιστες εφαρμογές σε τομείς όπου η ακρίβεια και η τεκμηρίωση είναι κρίσιμες.

6.2 Παραδοτέα Εφαρμογής

Προτού προχωρήσουμε στη θεωρητική ανάλυση του RAG, θα περιγράψουμε τις απαιτήσεις της εφαρμογής. Η εργασία εστιάζει στην ανάπτυξη ενός RAG chatbot για την εφαρμογή “ΑΘΗΝΑ”, η οποία χρησιμοποιείται από τραπεζικούς υπαλλήλους για τη δημιουργία χρηματοοικονομικών διαγραμμάτων. Στόχος είναι να καλυφθεί η ανάγκη άμεσης πρόσβασης σε πληροφορίες για τις λειτουργίες της εφαρμογής, καθώς η αναζήτηση στον οδηγό χρήσης είναι χρονοβόρα και μειώνει την αποδοτικότητα.

Το chatbot θα λειτουργεί ως άμεση βάση γνώσης, επιτρέποντας ερωτήσεις σε φυσική γλώσσα και γρήγορες, αξιόπιστες απαντήσεις. Δεδομένου ότι ο οδηγός περιλαμβάνει σύνθετες δομές όπως πίνακες και εικόνες, απαιτούνται εξειδικευμένες τεχνικές αποθήκευσης και ανάκτησης, οι οποίες θα αναλυθούν στη συνέχεια.

6.3 Οικοσύστημα LangChain

Το οικοσύστημα LangChain [34] αποτελεί ένα από τα σημαντικότερα εργαλεία για την ανάπτυξη εφαρμογών που βασίζονται σε γλωσσικά μοντέλα μεγάλης κλίμακας (LLMs). Προσφέρει ένα ενιαίο πλαίσιο που συνδέει τα LLMs με εξωτερικές πηγές δεδομένων, APIs και μηχανισμούς συλλογισμού, καθιστώντας τα πιο ευέλικτα και πρακτικά σε πραγματικές εφαρμογές.

Η βασική βιβλιοθήκη παρέχει αφαιρέσεις για *prompts*, *memory*, *chains* και *agents*, βοηθώντας στη διαχείριση εισόδου, συμφραζομένων και εκτέλεσης εργασιών. Παράλληλα, η ευρεία γκάμα από ενσωματώσεις επιτρέπει σύνδεση με βάσεις δεδομένων, vector stores, APIs και cloud υπηρεσίες, προσφέροντας μεγάλη ευελιξία στην αρχιτεκτονική των εφαρμογών.

Επιπλέον, εργαλεία όπως το LangSmith διευκολύνουν τον έλεγχο, την παρακολούθηση και τον εντοπισμό σφαλμάτων, ενώ το LangServe επιτρέπει την εύκολη ανάπτυξη και ενσωμάτωση αλυσίδων και πρακτόρων σε μεγαλύτερα συστήματα. Η ανοιχτή κοινότητα του LangChain ενισχύει συνεχώς το οικοσύστημα με νέες ενσωματώσεις και τεκμηρίωση.

Ανακεφαλαιώνοντας, το LangChain προσφέρει τη δομή και τα εργαλεία για να μετατραπούν οι δυνατότητες των LLMs σε πλήρεις, αξιόπιστες και επεκτάσιμες εφαρμογές, κάτι που αποτέλεσε και τη βάση για την υλοποίηση της παρούσας εφαρμογής.

6.4 Προετοιμασία Δεδομένων

Η προετοιμασία των δεδομένων αποτελεί το θεμελιώδες στάδιο εκτός σύνδεσης σε ένα σύστημα Retrieval-Augmented Generation (RAG), με σκοπό τη μετατροπή ακατέργαστων δεδομένων σε μια δομημένη, αναζητήσιμη βάση γνώσης. Αυτή η διαδικασία εξασφαλίζει ότι εξωτερικές πληροφορίες είναι εύκολα προσβάσιμες κατά τη φάση ανάκτησης, περιλαμβάνοντας τα εξής κρίσιμα βήματα: φόρτωση και προεπεξεργασία εγγράφων, τεμαχισμός τους σε διαχειρίσιμα τμήματα, δημιουργία ενσωματώσεων (embeddings) και αποθήκευση αυτών σε μια βάση δεδομένων διανυσμάτων για αποδοτικές αναζητήσεις

ομοιότητας. Κάθε βήμα στοχεύει στη βελτιστοποίηση της ποιότητας και προσβασιμότητας της βάσης γνώσης, επιτρέποντας στο σύστημα RAG να παρέχει ακριβείς και συναφείς απαντήσεις.

Φόρτωση και Προεπεξεργασία Δεδομένων: Η διαδικασία ξεκινά με τη συλλογή εγγράφων από ποικίλες πηγές, όπως PDF, ιστοσελίδες ή βάσεις δεδομένων. Στην περίπτωσή μας, χρησιμοποιήθηκαν δύο PDF για την εφαρμογή «ΑΘΗΝΑ». Η προεπεξεργασία περιλαμβάνει την εξαγωγή κειμένου με εργαλεία όπως το PyMuPDF4LLM μέσω του πλαισίου LangChain, το οποίο υποστηρίζει την ανάλυση πινάκων και εικόνων. Για τις εικόνες, εφαρμόστηκαν δύο μέθοδοι: η πρώτη, πιο αποδοτική, χρησιμοποίησε ένα πολυτροπικό (multimodal) μοντέλο ('gpt-4o-mini') για λεπτομερή περιγραφή εικόνων, ενώ η δεύτερη, μέσω OCR, είχε χαμηλότερη ακρίβεια. Επιπλέον, η προεπεξεργασία περιλαμβάνει καθαρισμό (αφαίρεση θορύβου, όπως ειδικοί χαρακτήρες), κανονικοποίηση (επίτευξη ομοιομορφίας) και έλεγχο ποιότητας, με στόχο τη δημιουργία καθαρού, τυποποιημένου κειμένου.

Τεμαχισμός (Chunking) Εγγράφων: Λόγω του περιορισμένου παραθύρου περιεχομένου των γλωσσικών μοντέλων, τα έγγραφα τεμαχίζονται σε μικρότερα, σημασιολογικά συνεκτικά τμήματα. Κάποιες από τις μεθόδους περιλαμβάνουν: α) σταθερού μεγέθους τεμαχισμό (Fixed-Size Chunking), που είναι απλός αλλά μπορεί να διασπά τη σημασιολογική συνέχεια, β) αναδρομικό τεμαχισμό (Recursive Chunking), που διατηρεί λογικές μονάδες (π.χ., παραγράφους, προτάσεις) και εφαρμόστηκε στην περίπτωσή μας με δοκιμές διαφορετικών μεγεθών και επικαλύψεων, και γ) σημασιολογικό τεμαχισμό (Semantic Chunking), που βασίζεται στη σημασία του κειμένου, προσφέροντας υψηλή ποιότητα αλλά με μεγαλύτερο υπολογιστικό κόστος.

Δημιουργία Ενσωματώσεων (Embeddings) και Αποθήκευση: Μετά τον τεμαχισμό, το κείμενο μετατρέπεται σε διανυσματικές αναπαραστάσεις (embeddings) μέσω μοντέλων όπως το text-embedding-3-large, που επιλέχθηκε για την υψηλή του ακρίβεια στην ελληνική γλώσσα. Οι ενσωματώσεις αποθηκεύονται σε βάσεις δεδομένων διανυσμάτων, όπως οι Pinecone και ChromaDB, που υποστηρίζουν γρήγορες αναζητήσεις ομοιότητας. Η ποιότητα των ενσωματώσεων είναι κρίσιμη για την αποτελεσματικότητα του RAG, καθώς κακές ενσωματώσεις οδηγούν σε μη σχετικές ανακτήσεις.
Η διαδικασία προετοιμασίας των δεδομένων, μέσω αυτών των βημάτων, εξασφαλίζει ότι το σύστημα RAG μπορεί να ανακτήσει και να αξιοποιήσει πληροφορίες με ακρίβεια και αποτελεσματικότητα, αποτελώντας τη βάση για την παραγωγή ποιοτικών απαντήσεων.

6.5 Διαδικασία Ανάκτησης και Παραγωγής

Η διαδικασία ανάκτησης και παραγωγής (Retrieval and Generation Pipeline) περιγράφει τη διαδικασία που εκτελείται σε πραγματικό χρόνο όταν ο χρήστης υποβάλλει ένα ερώτημα στο σύστημα. Βασίζεται στα δεδομένα που έχουν προηγουμένως εισαχθεί κατά την offline φάση εισαγωγής και εξασφαλίζει ότι η απάντηση του LLM στηρίζεται σε έγκυρες και σχετικές πληροφορίες. Η διαδικασία αποτελείται από τέσσερα στάδια: *ενσωμάτωση ερωτήματος*

(Query Embedding), *σημασιολογική αναζήτηση και ανάκτηση (Retrieval)*, *εμπλουτισμό περιεχομένου (Context Augmentation)* και *παραγωγή με εμπλουτισμό (Augmented Generation)*.
Στο πρώτο στάδιο, το ερώτημα του χρήστη μετατρέπεται σε αριθμητικό διάνυσμα μέσω του ίδιου μοντέλου ενσωμάτωσης που χρησιμοποιήθηκε στην offline διαδικασία (π.χ. Text-embedding-3-large στην περίπτωση μας). Έτσι, το ερώτημα και τα αποσπάσματα εγγράφων συνυπάρχουν στον ίδιο διανυσματικό χώρο, επιτρέποντας ουσιαστική σύγκριση με βάση τη σημασιολογική ομοιότητα.
Ακολουθεί η **σημασιολογική αναζήτηση**, όπου το ερώτημα-διάνυσμα συγκρίνεται με τα διανύσματα των εγγράφων στη βάση δεδομένων. Μετρικές όπως *Cosine Similarity* ή *Dot Product* καθορίζουν τη συνάφεια, και ανακτώνται τα *top-k* πιο σχετικά αποσπάσματα. Το κατάλληλο *k* καθορίζεται με βάση αξιολογητικές μετρικές, ενώ στην πράξη χρησιμοποιήθηκαν συστήματα όπως ChromaDB και Pinecone για αποδοτική αναζήτηση.
Στο στάδιο του **εμπλουτισμού περιεχομένου**, τα ανακτημένα αποσπάσματα ενσωματώνονται σε ένα προκαθορισμένο πρότυπο προτροπής (prompt template), μαζί με το αρχικό ερώτημα. Το πρότυπο αυτό καθοδηγεί το LLM να απαντήσει αποκλειστικά με βάση τα παρεχόμενα δεδομένα και να περιορίσει τυχόν "φανταστικές" πληροφορίες.
Τέλος, στη **φάση παραγωγής με εμπλουτισμό**, το πλήρες prompt αποστέλλεται στο LLM (στην περίπτωση μελέτης το gpt-4). Το μοντέλο δεν αντλεί πλέον από τη γενική του προεκπαίδευση, αλλά συνθέτει μια τεκμηριωμένη και φυσική απάντηση, στηριγμένη αποκλειστικά στο ανακτημένο περιεχόμενο. Με τον τρόπο αυτό μειώνεται ο κίνδυνος λανθασμένων απαντήσεων και διασφαλίζεται η ακρίβεια και η ιχνηλασιμότητα της πληροφορίας.
Η συνολική υλοποίηση, χάρη σε εργαλεία όπως το LangChain, μπορεί να εκτελεί τα στάδια ανάκτησης, εμπλουτισμού και παραγωγής με απλές κλήσεις συναρτήσεων, καθιστώντας τη διαδικασία πρακτικά εφαρμόσιμη και αποδοτική.

6.6 Colpali

Το ColPali αποτελεί μια καινοτόμο μέθοδο αναζήτησης και κατανόησης εγγράφων με πλούσιο οπτικό περιεχόμενο (όπως PDF με πίνακες, εικόνες ή διαγράμματα), η οποία υπερβαίνει τους περιορισμούς των παραδοσιακών συστημάτων. Αντί να στηρίζεται σε περίπλοκες διαδικασίες όπως OCR, ανάλυση διάταξης και αποσπασματική εξαγωγή κειμένου, το ColPali αξιοποιεί μοντέλα Όρασης-Γλώσσας (Vision-Language Models – VLMs) για να επεξεργάζεται απευθείας τις εικόνες των σελίδων, ενσωματώνοντας ταυτόχρονα οπτικές και γλωσσικές πληροφορίες. Με αυτό τον τρόπο, προσφέρει μια απλοποιημένη και πλήρως εκπαιδεύσιμη προσέγγιση για αναζήτηση εγγράφων, επιτυγχάνοντας υψηλότερη απόδοση σε εργασίες που απαιτούν κατανόηση τόσο του κειμένου όσο και της δομής της σελίδας.

Αρχιτεκτονική
Η μέθοδος βασίζεται στο PaliGemma-3B VLM, το οποίο συνδυάζει έναν οπτικό κωδικοποιητή (SigLIP) με ένα γλωσσικό μοντέλο (Gemma-2B) μέσω πολυτροπικής προβολής. Κάθε σελίδα εγγράφου μετατρέπεται σε εικόνα και χωρίζεται σε "patches", τα οποία λειτουργούν ως οπτικά tokens. Ο μηχανισμός προβολής μετατρέπει τις αναπαραστάσεις σε έναν χαμηλότερης διάστασης χώρο (D=128), δημιουργώντας πολλαπλά διανύσματα ανά σελίδα. Αυτό επιτρέπει λεπτομερή αντιστοίχιση ερωτημάτων και εγγράφων μέσα σε έναν κοινό πολυτροπικό χώρο. Με τεχνικές όπως η ιεραρχική μέση ομαδοποίηση μειώνεται ο πλεονασμός των δεδομένων, βελτιώνοντας αποδοτικότητα και μνήμη.

Εκπαίδευση
Το ColPali εκπαιδεύτηκε με περίπου 119.000 ζεύγη ερωτήσεων–σελίδων, προερχόμενα από ακαδημαϊκά σύνολα δεδομένων (DocVQA, InfoVQA) και συνθετικά δεδομένα. Η εκπαίδευση γίνεται με Low-Rank Adaptation (LoRA) και αντιθετική απώλεια (contrastive loss), ώστε το σύστημα να μαθαίνει να αντιστοιχίζει ερωτήματα με τις πιο σχετικές σελίδες. Η διαδικασία είναι end-to-end, βελτιστοποιώντας ταυτόχρονα την οπτική και γλωσσική κατανόηση.

Λειτουργία – Αναζήτηση Εγγράφων
Το ColPali λειτουργεί σε δύο βασικά στάδια που συνδυάζουν αποτελεσματικότητα, ακρίβεια και ερμηνευσιμότητα.

1. Δημιουργία ευρετηρίου (Indexing):

Σε αυτό το στάδιο, κάθε σελίδα εγγράφου αποδίδεται ως εικόνα και εισάγεται στο Vision-Language Model (VLM). Το μοντέλο αναλύει την οπτική πληροφορία (π.χ. κείμενο, πίνακες, διαγράμματα, σχήματα) και δημιουργεί πολυδιανυσματικές αναπαραστάσεις που αποτυπώνουν τόσο τη γλωσσική όσο και τη χωρική δομή της σελίδας. Κάθε διάνυσμα αντιστοιχεί σε ένα "patch" ή σε ένα τμήμα της σελίδας, επιτρέποντας την ακριβή κωδικοποίηση του περιεχομένου χωρίς ανάγκη για OCR, ανίχνευση διάταξης ή λεζάντες εικόνων. Τα δεδομένα αυτά αποθηκεύονται αποδοτικά, καθιστώντας το ευρετήριο ελαφρύ και εύχρηστο για μελλοντική αναζήτηση.

2. Αναζήτηση (Querying):

Όταν ο χρήστης υποβάλει ένα κείμενο-ερώτημα, αυτό μετατρέπεται από το ίδιο μοντέλο σε ένα σύνολο πολυδιανυσματικών αναπαραστάσεων, αντίστοιχης μορφής με εκείνες των σελίδων. Η ομοιότητα μεταξύ ερωτήματος και εγγράφων υπολογίζεται μέσω ενός μηχανισμού καθυστερημένης αλληλεπίδρασης (late interaction): για κάθε διάνυσμα του ερωτήματος εντοπίζεται το πιο σχετικό διάνυσμα από το ευρετήριο (με βάση το μέγιστο εσωτερικό γινόμενο), και το άθροισμα αυτών των επιμέρους ομοιοτήτων καθορίζει τη συνολική συνάφεια. Έτσι, το σύστημα μπορεί να εστιάζει στα πιο σημαντικά στοιχεία κάθε σελίδας, είτε αυτά είναι λέξεις, τίτλοι, πίνακες ή εικόνες.
Η διαδικασία είναι πολύ γρήγορη και επεκτάσιμη, απαιτώντας περίπου 0,39 δευτερόλεπτα ανά σελίδα για την ευρετηρίαση και μόλις 30 χιλιοστά του δευτερολέπτου για κάθε ερώτημα.

Επιπλέον, το ColPali προσφέρει ερμηνευσιμότητα μέσω θερμικών χαρτών (heatmaps), οι οποίοι απεικονίζουν τα σημεία της σελίδας που σχετίζονται περισσότερο με την αναζήτηση. Με αυτό τον τρόπο, ο χρήστης μπορεί να κατανοήσει πώς και γιατί το σύστημα εντόπισε συγκεκριμένες πληροφορίες, ενισχύοντας τη διαφάνεια και την εμπιστοσύνη στα αποτελέσματα.

Αποτελέσματα και Συμπεράσματα

Στο πρότυπο ViDoRe, το ColPali επιτυγχάνει υψηλότερες επιδόσεις (nDCG@5 = 81,3%) από τα παραδοσιακά συστήματα, ιδίως σε οπτικά απαιτητικές εργασίες (π.χ. πίνακες, διαγράμματα). Είναι γρηγορότερο, πιο αποδοτικό και ανθεκτικό σε σφάλματα OCR, ενώ γενικεύει καλά και σε άλλες γλώσσες.

Συνολικά, το ColPali σηματοδοτεί μια στροφή προς την οπτικο-κεντρική ανάκτηση εγγράφων, ενοποιώντας την κατανόηση εικόνας και κειμένου. Αποτελεί μια ευέλικτη, επεκτάσιμη λύση για εφαρμογές ακαδημαϊκής έρευνας και βιομηχανίας, θέτοντας τις βάσεις για πιο έξυπνα και αποτελεσματικά συστήματα Retrieval Augmented Generation (RAG).

7 Αξιολόγηση RAG συστήματος

7.1 Εισαγωγή στην Αξιολόγηση Συστημάτων RAG

Η αποτελεσματικότητα ενός συστήματος RAG (Retrieval-Augmented Generation) δεν είναι δεδομένη. Η απόδοσή του εξαρτάται από τη σωστή συνεργασία των δύο βασικών του τμημάτων: του μηχανισμού ανάκτησης πληροφοριών (retriever) και του μηχανισμού παραγωγής κειμένου (generator). Μια αδυναμία σε οποιοδήποτε από τα δύο μπορεί να οδηγήσει σε άσχετες, ανακριβείς ή ελλιπείς απαντήσεις. Για τον λόγο αυτό, η συστηματική αξιολόγηση είναι κρίσιμη.

7.2 Μετρικές Αξιολόγησης

Η αξιολόγηση γίνεται με δύο τρόπους σύμφωνα με το εργαλείο RAGAS [35]. Ο πρώτος εστιάζει στα επιμέρους στοιχεία του συστήματος, εξετάζοντας την ποιότητα των πληροφοριών που ανακτώνται και τη σχέση τους με την τελική απάντηση. Οι βασικές μετρικές είναι οι εξής:

- **Πιστότητα (Faithfulness):** Εξετάζει αν η απάντηση είναι συνεπής με τις πληροφορίες που ανακτήθηκαν, χωρίς να "εφευρίσκει" δεδομένα.

- **Συνάφεια Απάντησης (Answer Relevancy):** Μετρά κατά πόσο η απάντηση είναι σχετική με την αρχική ερώτηση του χρήστη.

- **Ανάκληση & Ακρίβεια Περιεχομένου (Context Recall & Precision):** Αξιολογούν αν το σύστημα βρήκε όλες τις απαραίτητες πληροφορίες (Ανάκληση) και αν οι πληροφορίες που βρέθηκαν ήταν όντως χρήσιμες και όχι περιττές (Ακρίβεια).

Ο δεύτερος τρόπος αξιολογεί το τελικό αποτέλεσμα συνολικά, συγκρίνοντας την απάντηση του συστήματος με μια ιδανική, "αληθινή" απάντηση (ground truth). Οι κυριότερες μετρικές είναι:

- **Σημασιολογική Ομοιότητα (Answer Semantic Similarity):** Ελέγχει αν η παραγόμενη απάντηση έχει το ίδιο νόημα με την ιδανική απάντηση, ακόμη κι αν χρησιμοποιεί διαφορετικές λέξεις.

- **Ορθότητα (Answer Correctness):** Συνδυάζει την πραγματολογική ακρίβεια με τη σημασιολογική ομοιότητα για μια πιο ισορροπημένη αξιολόγηση.

Τέλος, υπάρχουν και ποιοτικοί έλεγχοι (Aspect Critiques) που αξιολογούν πτυχές όπως η συνοχή, η ορθότητα και η απουσία επιβλαβούς περιεχομένου. Ο συνδυασμός όλων αυτών των μετρικών προσφέρει μια ολοκληρωμένη εικόνα για την απόδοση και την αξιοπιστία ενός συστήματος RAG.

7.3 Πρακτική Αξιολόγηση του RAG Pipeline

Σε συνέχεια της συζήτησης για τις διάφορες μετρικές αξιολόγησης, αυτή η ενότητα εστιάζει στην πρακτική αξιολόγηση της δικής μας εφαρμογής ως προς τις πιο σχετικές από αυτές τις μετρικές. Κατά τον σχεδιασμό της διαδικασίας αξιολόγησης, πειραματιστήκαμε με διαφορετικές παραμέτρους, συγκεκριμένα μεταβάλλοντας το μέγεθος των τμημάτων (chunk size), τον βαθμό αλληλεπικάλυψης (chunk overlap) και τον αριθμό των ανακτημένων τμημάτων.

Είναι σημαντικό να τονιστεί ότι θα μπορούσαν να εξεταστούν πολλές εναλλακτικές διαμορφώσεις και κριτήρια αξιολόγησης. Το RAG pipeline προσφέρει πολυάριθμες ρυθμιζόμενες πτυχές και η αξιολόγηση μπορεί να διεξαχθεί σε πολλαπλές διαστάσεις. Ωστόσο, για τους σκοπούς αυτής της μελέτης, επιλέξαμε σκόπιμα τις προαναφερθείσες παραμέτρους, καθώς τις θεωρήσαμε ως τους πιο κρίσιμους παράγοντες που επηρεάζουν την απόδοση του συστήματός μας.

Το πρώτο βήμα ήταν η επιλογή των διαμορφώσεων. Επιλέξαμε 4 διαφορετικές διαμορφώσεις και 3 μετρικές που κρίναμε ως τις πιο σημαντικές. Για την πρώτη διαμόρφωση, το αρχικό PDF φορτώθηκε με την προεπιλεγμένη συνάρτηση load_document, η οποία χρησιμοποιεί την κλάση LLMImageBlobParser για να αντικαταστήσει τις εικόνες με τις λεκτικές περιγραφές τους. Αντίθετα, το δεύτερο PDF φορτώθηκε με τη συνάρτηση load_document_OCR, όπου οι εικόνες αντικαταστάθηκαν με κείμενο που εξήχθη μέσω OCR. Σε αυτή τη ρύθμιση, το μέγεθος του chunk ορίστηκε σε 2800, χωρίς αλληλεπικάλυψη (overlap), και η παράμετρος ανάκτησης καθόρισε την ανάκτηση δύο τμημάτων. Στη συνέχεια, δοκιμάστηκαν τρεις επιπλέον διαμορφώσεις, όλες χρησιμοποιώντας την προεπιλεγμένη συνάρτηση load_document αλλά με διαφορετικές παραμέτρους:

- Η δεύτερη διαμόρφωση χρησιμοποίησε μέγεθος chunk 2000 χωρίς αλληλεπικάλυψη, ανακτώντας δύο τμήματα.

- Η τρίτη διαμόρφωση χρησιμοποίησε μέγεθος chunk 3500 με αλληλεπικάλυψη 100, ανακτώντας επίσης δύο τμήματα.
- Τέλος, η τέταρτη διαμόρφωση εφάρμοσε μέγεθος chunk 4200 με αλληλεπικάλυψη 100, ανακτώντας μόνο ένα τμήμα.

Μετά την εφαρμογή των διαμορφώσεων, δημιουργήσαμε ένα αρχείο CSV για καθεμία με τις στήλες: "user_input", "retrieved_contexts", "response", "reference", "faithfulness", "context_precision" και "answer_correctness". Οι πρώτες τέσσερις στήλες αποτέλεσαν το σύνολο δεδομένων δοκιμής, ενώ οι τρεις τελευταίες αντιπροσώπευαν τις μετρικές αξιολόγησης. Οι μετρικές αυτές ήταν: **Faithfulness (Πιστότητα)**, **Context Precision (Ακρίβεια Πλαισίου)** και **Answer Correctness (Ορθότητα Απάντησης)**. Τα αποτελέσματα ανέδειξαν σαφή υπεροχή της Διαμόρφωσης 1, η οποία παρουσίασε τις υψηλότερες τιμές σε όλες τις κατηγορίες.

Όσον αφορά τη **Faithfulness**, η Διαμόρφωση 1 πέτυχε τον υψηλότερο μέσο όρο (0.676), γεγονός που δείχνει ότι οι απαντήσεις της ήταν πιο συνεπείς με τις ανακτημένες πληροφορίες. Οι επόμενες διαμορφώσεις εμφάνισαν σταδιακά χαμηλότερες επιδόσεις, υποδηλώνοντας μειωμένη ικανότητα ενσωμάτωσης του παρεχόμενου περιεχομένου στη διαδικασία δημιουργίας απαντήσεων.

Στην **Context Precision**, η ίδια διαμόρφωση διατήρησε την πρωτιά (0.758), ενώ οι υπόλοιπες σημείωσαν φθίνουσες τιμές. Αυτό υποδηλώνει ότι το σύστημα ανάκτησης των άλλων διαμορφώσεων παρήγαγε περισσότερο άσχετες ή επαναλαμβανόμενες πληροφορίες, μειώνοντας τη συνάφεια και την ακρίβεια των απαντήσεων.

Αντίστοιχα, στην **Answer Correctness**, η Διαμόρφωση 1 απέδωσε τις πιο ακριβείς απαντήσεις (0.506), με τις υπόλοιπες να ακολουθούν πτωτικά. Η συνοχή των αποτελεσμάτων σε όλες τις μετρικές καταδεικνύει στενή σχέση ανάμεσα στην ποιότητα ανάκτησης, την πιστότητα στο πλαίσιο και την ορθότητα των παραγόμενων απαντήσεων.

Συνολικά, τα αποτελέσματα αναδεικνύουν μια ξεκάθαρη ιεραρχία απόδοσης, με τη Διαμόρφωση 1 να επιτυγχάνει τον βέλτιστο συνδυασμό συνάφειας, αξιοπιστίας και ακρίβειας. Για τον λόγο αυτό επιλέχθηκε ως η πλέον κατάλληλη για συνέχιση της ανάπτυξης και υλοποίησης της εφαρμογής RAG, καθώς εξασφαλίζει μειωμένα σφάλματα και αυξημένη αξιοπιστία. Στο επόμενο κεφάλαιο παρουσιάζεται η διαδικασία ανάπτυξης και δημιουργίας του τελικού περιβάλλοντος χρήστη.

8 Υλοποίηση Εφαρμογής Αλληλεπίδρασης με τον Τελικό Χρήστη

Το κεφάλαιο αυτό περιγράφει τη διαδικασία ανάπτυξης της τελικής εφαρμογής chatbot, εστιάζοντας στο Streamlit, το framework που χρησιμοποιήθηκε για τη δημιουργία του γραφικού περιβάλλοντος (front-end). Το Streamlit είναι ένα εργαλείο ανοιχτού κώδικα που απλοποιεί τη δημιουργία διαδραστικών web εφαρμογών για επιστήμη δεδομένων και μηχανική μάθηση, καθώς είναι βασισμένο στη γλώσσα Python.

Ένα βασικό του πλεονέκτημα είναι ότι επιτρέπει τη γρήγορη ανάπτυξη πρωτοτύπων (rapid prototyping) χωρίς να απαιτεί γνώσεις HTML, CSS ή JavaScript. Οι προγραμματιστές μπορούν με λίγες γραμμές κώδικα να προσθέσουν διαδραστικά στοιχεία, όπως κουμπιά και sliders, και να τα συνδέσουν με τα δεδομένα ή τα αποτελέσματα ενός μοντέλου.
Το Streamlit είναι ιδιαίτερα χρήσιμο για την ενσωμάτωση μοντέλων machine learning, επιτρέποντας τη μετατροπή τους σε web εφαρμογές χωρίς περίπλοκες υποδομές. Για εφαρμογές όπως το RAG (Retrieval-Augmented Generation), δημιουργεί ένα φιλικό προς τον χρήστη περιβάλλον, δίνοντας τη δυνατότητα στους χρήστες να αλληλεπιδρούν εύκολα με το σύστημα.

Η εφαρμογή υλοποιήθηκε και αναπτύχθηκε με το πλαίσιο Streamlit που προαναφέρθηκε, έπειτα από την επιλογή της κατάλληλης διαμόρφωσης RAG pipeline. Στον φάκελο του έργου εγκαταστάθηκε το Streamlit και δημιουργήθηκε ένα ειδικό αρχείο Python που περιλάμβανε συναρτήσεις για τη φόρτωση των embeddings, την παραγωγή απαντήσεων και την κατασκευή της διεπαφής χρήστη. Η διεπαφή ενσωμάτωνε τίτλο, sidebar για καταχώριση στοιχείων και ερωτημάτων από τον χρήστη, καθώς και στοιχεία για την απεικόνιση της συνομιλίας. Μετά από δοκιμές που επιβεβαίωσαν την ορθή λειτουργία, ο κώδικας ανέβηκε στο GitHub και μέσω της ενσωματωμένης δυνατότητας ανάπτυξης του Streamlit η εφαρμογή αναπτύχθηκε επιτυχώς. Είναι πλέον διαθέσιμη στη διεύθυνση “https://diplomaappdeployment.streamlit.app/”, με τελική μορφή που υποστηρίζει ερωτήσεις βασισμένες στον οδηγό χρήσης “ATHENA”.

9 Ανακαιφαλαίωση

Η διπλωματική εργασία παρουσίασε τον σχεδιασμό, την υλοποίηση και την αξιολόγηση ενός συστήματος chatbot βασισμένου σε RAG, με σκοπό την υποστήριξη των χρηστών της εφαρμογής «ΑΘΗΝΑ». Το σύστημα απέδειξε ότι μπορεί να απαντά αποτελεσματικά σε ερωτήσεις, προσφέροντας άμεση και στοχευμένη πρόσβαση στις πληροφορίες του οδηγού χρήσης και βελτιώνοντας σημαντικά την εμπειρία του χρήστη. Ωστόσο, λόγω της πιθανοκρατικής φύσης των γενετικών μοντέλων, δεν είναι δυνατόν να εξαλειφθεί πλήρως η πιθανότητα λανθασμένων απαντήσεων, γεγονός που αναδεικνύει την ανάγκη για συνεχή βελτίωση. Μελλοντικά, προτείνεται η αξιοποίηση πιο προηγμένων γλωσσικών μοντέλων (ιδίως για την ελληνική γλώσσα), η εφαρμογή εξελιγμένων τεχνικών επεξεργασίας δεδομένων, καθώς και ο εμπλουτισμός και η καλύτερη δομή του υλικού του οδηγού χρήσης. Με αυτά τα βήματα, η ακρίβεια, η πληρότητα και η χρησιμότητα του chatbot μπορούν να ενισχυθούν ακόμη περισσότερο.

Extended English Version

1 Introduction

In recent years, artificial intelligence technologies and, in particular, Natural Language Processing (NLP), have experienced rapid development, making it possible to create advanced dialogue systems (chatbots) that approach human levels of understanding and response. One of the most recent and powerful examples of such systems are RAG (Retrieval-Augmented Generation) models, which combine the capabilities of large language models (LLMs) with the search for information from external knowledge sources.

This thesis focuses on the development of a RAG Chatbot, which utilizes the Retrieval-Augmented Generation architecture to provide highly accurate answers, documented on the basis of relevant documents or databases. The combination of search and language production allows the system to overcome the limitations of traditional chatbots, which rely exclusively on the knowledge that has been integrated during their training phase.

The aim of the work is to design, implement and evaluate a RAG system, which will be used by users of the "ATHENA" application. "ATHENA" is an application through which financial data representation diagrams can be created for a bank. Employees who use this application often have questions about the correct use of this specific application. Therefore, creating a RAG chatbot will make things much easier for employees and increase productivity, as they will be able to ask the chatbot a question and get an answer immediately without having to search through the entire user guide.

1.1 Chatbot Definition

A chatbot is a software application, often powered by artificial intelligence (AI), designed to simulate and process human conversation, whether written or spoken. It enables users to interact with digital systems in a conversational manner, simulating a conversation with a human [36].

The term chatbot is directly related to the term LLM. A large language model (LLM) is a type of deep learning model that is trained on vast amounts of text data—such as books, websites, or articles—to learn patterns, grammar, facts, and relationships between words. Modern LLMs are typically based on transformer architecture, which uses self-attention mechanisms to process entire input sequences in parallel, making them robust and scalable. Once trained, LLMs can generate human-like text by predicting the next word, or "symbol", in a sentence. This capability allows them to perform many tasks without being trained on a specific task, a phenomenon known as zero-shot or few-shot learning [28].

Having been trained on a large amount of data, LLMs gain knowledge and can answer correctly in a wide range of domains. But what happens when the user asks for something that is not included in the training data, such as something that happened after the model was trained, the content of a website, a database, or, in our case, a user manual with text, images, and tables? This is where Retrieval Augmented Generation (RAG) comes in hand.

1.2 Retrieval Augmented Generation (RAG)

The Retrieval-Augmented Generation (RAG) technique is a modern approach to augmenting large language models (LLMs) with dynamic access to external information sources at runtime. Unlike traditional LLMs that rely solely on the parameters learned during their training phase, RAG models also incorporate mechanisms for retrieving knowledge from external data sources, with the aim of generating more accurate, timely and evidence-based answers.

The architecture of a RAG system is based on two main subsystems:

Retriever:

This is a system that, when receiving a query from the user, searches for relevant text fragments from an external knowledge base (usually a vector base, such as FAISS or Chroma). The texts in the database have been pre-transformed into vector representations (embeddings) using a separate model, to allow searching based on semantic relevance and not just exact word matching.

Text generator:

The LLM receives the retrieved chunks as information (context) along with the user's initial query. It then composes an answer that is based not only on the knowledge contained in the model itself, but also on information that came from outside. This process is called augmentation, as the prompt processed by the model is dynamically enriched.

The use of the RAG technique presents significant advantages:

- Reduction of false answers (hallucinations), as the model is based on external, documented data.
- Content update, without the need to retrain the model every time there is new information. This is very useful for cases like owr own where the user manual or app data might change.
- Transparency and documentation, as the user can be provided with the same sources used to produce the answer.
- Increased reliability in answers, especially in areas that require accurate and up-to-date knowledge (e.g. legal, medical or business data).

In this paper, we study ways in which we can produce the most correct and informative answers possible that are consistent with both the user's initial question and the retrieved data, reducing errors, inaccuracies and hallucinations.

2 Machine Learning

Before analyzing the architecture of the chatbot that we will develop, we must establish the theoretical foundations on which this technology is based. The general science in which an AI chatbot is classified is artificial intelligence. Analyzing concepts such as machine learning and deep learning will lay the foundation to understand concepts such as transformers and large language models (LLMs). Thus, we will have all the theoretical background to understand the composite architecture of a RAG chatbot.

2.1 Machine Learning Definition

Let's start our analysis with an introduction and definition of machine learning. Machine Learning is a subfield of artificial intelligence (AI) that focuses on developing algorithms and statistical models that allow computers to perform tasks without being explicitly programmed for each specific action. By learning from data, these systems can identify patterns, make predictions, and improve their performance over time [37].

At its core, Machine Learning harnesses the power of data and computational techniques to simulate aspects of human learning. Unlike traditional programming, where a programmer manually writes rules to perform tasks, machine learning systems extract rules and relationships directly from data. This has opened up new possibilities in areas as diverse as image and speech recognition, natural language processing, medical diagnosis, autonomous systems, economic modeling, and more.

2.2 Machine Learning Categories

Machine Learning can be categorized into supervised learning, unsupervised learning, and reinforcement learning, each of which is characterized by the nature of the learning process and the type of data available. Supervised learning involves learning from labeled data, where the algorithm is trained to predict an output based on input-output pairs. Unsupervised learning, on the other hand, deals with data without explicit labels, focusing on discovering hidden structures or patterns. Reinforcement learning is based on an agent learning to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties. Following we will shortly present some machine learning algorithms without going into too much detail, as they are not prerequisites for the subject of this diploma thesis.

2.2.1 Supervised Learning

Supervised learning is a type of machine learning that aims to classify or predict data, based on a set of examples provided in advance for training. These examples include input data (usually feature vectors) and the corresponding desired outputs or labels. Supervised learning algorithms study these examples and create a model, which can then be used to predict or classify new, unknown cases. The goal is for the model to be able to apply the knowledge it has acquired from the training data to new data in a way that makes sense, that is, to generalize well [2]. Following, we will make an introduction to som

2.2.1.1 Linear Regression

Linear regression is one of the most fundamental algorithms in machine learning and statistics, used to model the relationship between one or more input variables (often called features) and a continuous output variable. The goal is to find a linear equation that best describes how the inputs influence the output. For a single input, this takes the form of a straight line; for multiple inputs, it becomes a flat surface (or hyperplane) in higher dimensions.

The algorithm estimates the *weights* (or coefficients) for each input variable so that the predictions are as close as possible to the actual observed values. This is typically done by minimizing the overall difference between predicted and actual values, often measured as the average squared error. Once trained, the model can be used to make predictions for new data and to interpret the strength and direction of the influence of each input [38].

Below is shown the scatter plot for the result of the simple linear regression algorithm.

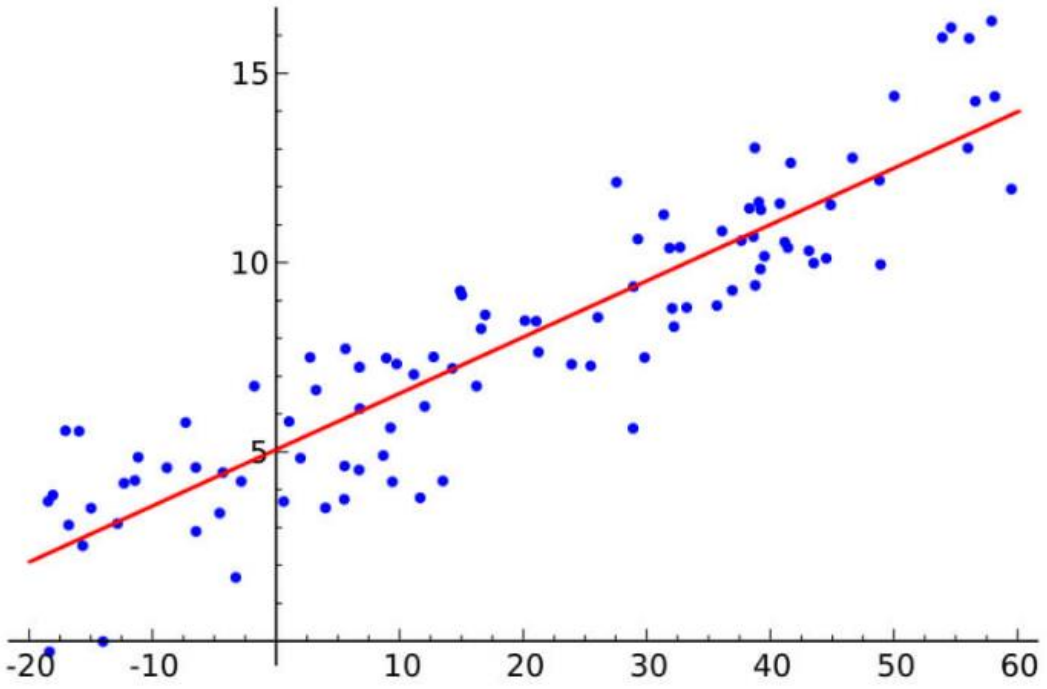

Figure 1 – Scatter Plot for simple linear regression [38]

Linear regression is used to predict continuous values based on one or multiple input values. For binary values, the algorithm that is used is logistic regression.

3.2.1.2 Logistic regression

Logistic regression is a **supervised machine learning algorithm** traditionally used for **classification tasks**, particularly **binary classification**—predicting outcomes such as "yes/no," "0/1," or "spam/not spam." Unlike linear regression, which predicts a continuous variable, logistic regression estimates the **probability** of class membership using the **logistic (sigmoid) function** to bound outputs between 0 and 1 [39], [40].[41]

Mathematical Foundation
The model computes a weighted sum of input features and applies the logistic function:

$$g(z) = \frac{1}{1 + e^{-z}}$$

where $z = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$. This maps the real-valued z to a probability between 0 and 1.

Variants

- **Binary Logistic Regression**: for two-class outcomes (most common)
- **Multinomial Logistic Regression**: for unordered multi-class targets
- **Ordinal Logistic Regression**: for ordered multi-class outcomes (e.g. ratings)[42]

2.2.1.3 Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a supervised machine learning algorithm widely used for classification and, with adaptations, for regression tasks. The fundamental idea behind an SVM is to find the optimal decision boundary, known as a hyperplane, that separates data points belonging to different categories. Unlike a simple dividing line, this hyperplane is chosen in such a way that it maximizes the margin, which is the distance between the boundary and the closest data points of each class. These critical points, called support vectors, play a central role in defining the position and orientation of the hyperplane. By focusing only on these support vectors rather than the entire dataset, SVMs achieve strong generalization capabilities and robustness against overfitting, even in high-dimensional spaces.

2.2.1.4 k-Nearest Neighbors (k-NN)

The k-Nearest Neighbors (k-NN) algorithm is one of the simplest and most intuitive machine learning methods used for classification and regression. It belongs to the family of instance-

based (lazy) learning algorithms, meaning it does not explicitly learn a model but instead stores all training data and makes predictions based on similarity [43].

The k-NN algorithm works in three main steps:

1. **Store the Training Data**: The algorithm memorizes the entire training dataset (features and labels).
2. **Compute Distances**: For a new data point, it calculates the distance (e.g., Euclidean, Manhattan) to all other points in the training set.
3. **Find Nearest Neighbors**: It selects the **k** closest data points (neighbors) and predicts the output based on their labels.

- **For Classification**: The most common class among the **k** neighbors is chosen.
- **For Regression**: The average (or weighted average) of the neighbors' values is taken.

3. Mathematical Formulation

The **Euclidean distance** between two points $\mathbf{x}_1$ and $\mathbf{x}_2$ in an n-dimensional space is:

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

We use this to calculate the distance from the neigbors. Other distances can be used also, like manhattan.

For classification, the predicted class $\hat{y}$ is determined by majority voting:

$$\hat{y} = \text{mode}(\{y_i \mid i \in \text{top } k \text{ neighbors}\})$$

For regression, the predicted value is the average of the neighbors:

$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y_i$$

While deciding on the right value of k to use for the algorithm it is important to note that:

- A small **k** (e.g., 1) leads to high variance (overfitting).
- A large **k** (e.g., 20) leads to high bias (underfitting).
- The optimal **k** is often found using **cross-validation**.

To sum up, k-NN is a fundamental algorithm in machine learning, useful for both classification and regression. While simple, its performance depends on proper tuning of **k** and distance metrics. It serves as a good baseline model before exploring more complex algorithms.

2.2.2 Unsupervised Machine Learning

Unsupervised machine learning is a branch of machine learning where algorithms work with data that is not labeled. Unlike supervised learning, where the goal is to predict known outcomes, unsupervised learning focuses on discovering hidden structures, relationships, or patterns within the data itself. The input consists only of features, without any predefined target values.

The central idea is that data often contains inherent groupings or underlying structures that can be uncovered without prior knowledge. Two of the most common tasks in this area are:

- **Clustering** – grouping data points into clusters based on similarity (e.g., k-means, hierarchical clustering).

- **Dimensionality reduction** – reducing the number of features while preserving essential information, often for visualization or preprocessing (e.g., Principal Component Analysis, t-SNE).

2.2.3 Reinforcement Learning

The last machine learning category mentioned in this diploma thesis is Reinforcement Learning. Reinforcement Learning (RL) is a core paradigm of machine learning where an agent learns to make decisions by interacting with a dynamic environment, aiming to maximize a notion of cumulative reward over time [3]. Unlike supervised or unsupervised learning, RL doesn't rely on labeled examples; instead, agents learn through trial and error, striking a balance between exploration (trying new actions) and exploitation (leveraging known rewarding actions).

Essential Components of RL

- **Agent**: the decision-maker learning from interaction.

- **Environment**: the context within which the agent operates and evolves.

- **State**: a representation of the current situation.

- **Action**: choices available to the agent.

- **Reward**: feedback signals that drive learning, guiding the agent toward desirable outcomes [4].

3 Deep learning

In order to build the knowledge required to learn about LLMs we have to first learn about deep learning and some of the architecture around this field. In this chapter we will mention some relevant concepts as the technology of the LLMs is highly based on deep learning architecture and neural networks.

3.1 Introduction to deep learning

Deep learning is a subfield of machine learning that focuses on training artificial neural networks with multiple layers (hence "deep") to model complex patterns in data. Inspired by the structure and function of the human brain, deep learning algorithms automatically learn hierarchical representations of data, enabling breakthroughs in computer vision, natural language processing (NLP), speech recognition, and many other domains.

Unlike traditional machine learning, which often requires manual feature extraction, deep learning models can automatically discover relevant features from raw data. This capability has made deep learning a dominant force in artificial intelligence (AI), driving innovations such as self-driving cars, virtual assistants, and medical diagnostics.

3.2 Perceptron: The basis of a neural Network

The basic building block of a neural network is the perceptron, which is a simplified representation of a biological neuron [1]. A perceptron calculates a weighted sum of its input features x, adds a bias term b, and then passes the result through a non-linear activation function φ to generate an output:

$$y = \varphi\left(\sum_{i=1}^{n} w_i\, x_i + b\right)$$

Here, the w_i values represent the adjustable weights. This formulation enables the perceptron to define linear decision boundaries. On its own, a single perceptron can only solve linearly separable tasks, but combining many perceptrons allows the network to capture more complex, non-linear patterns.

Activation functions are crucial because they introduce non-linearity into the network, making it possible to model sophisticated relationships in the data. Commonly used activation functions include:

- **Sigmoid:**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Hyperbolic tangent (tanh):**

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- **Rectified Linear Unit (ReLU):**

$$\mathrm{ReLU}(x) = \max(0, x)$$

Each of these functions comes with advantages and drawbacks. For instance, sigmoid and tanh are prone to vanishing gradient problems in deep networks, which hinders effective learning. ReLU alleviates this issue but can result in inactive neurons. The selection of an activation function strongly influences the network's ability to learn efficiently and capture expressive patterns.

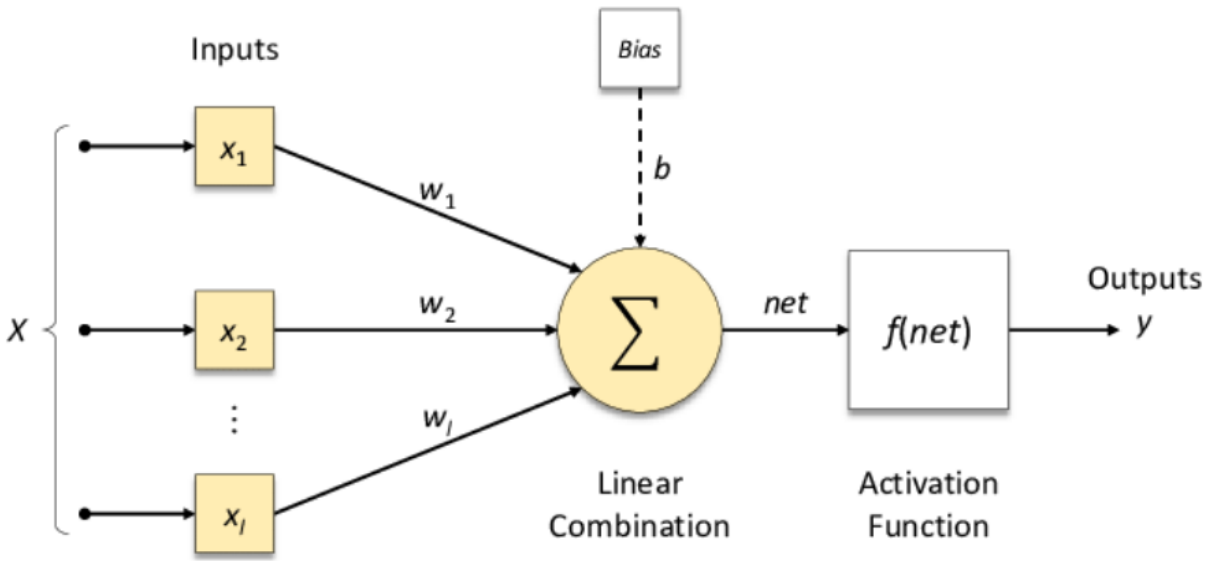

Figure 2 - Structure of a perceptron [44]

Neural networks are built from three main types of layers: the input layer, which takes in the raw features; one or more hidden layers, where the data is transformed and patterns are extracted; and the output layer, which delivers the final result or prediction.
Within each layer, neurons perform a combination of a linear operation and a non-linear activation. As data flows forward through the network, each layer refines the representation learned from the previous one. This step-by-step processing enables the network to capture deeper and more abstract patterns, making it capable of handling tasks ranging from simple classification to complex regression problems.

3.3 Deep Learning Architectures

In this chapter we will analyze some basic deep learning architectures which are crucial for the rest of this thesis. More specifically, we will analyze certain types of Artificial Neural Networks (ANNs) which are composed of interconnected layers of neurons.

More Specifically, we will mention:

- Feedforward Neural Networks (FNNs): Basic networks where data flows in one direction.
- Convolutional Neural Networks (CNNs): Specialized for grid-like data (e.g., images), using convolutional layers to detect spatial hierarchies.
- Recurrent Neural Networks (RNNs): Designed for sequential data (e.g., text, time series), with loops allowing information persistence.

Transformers, which is another architecture that leverages deep learning concepts and is also the basis of LLMs will be analyzed later after we introduce some other important concepts.

3.3.1 Feedforward Neural Networks (FNNs)

A Feedforward Neural Network (FNN) which we described in short in the previous chapter is one of the simplest and most widely used architectures in deep learning. Its structure is organized into layers, which form the foundation of how the network processes information.

An FNN is composed of three main types of layers:

- **Input layer**: This is where raw data first enters the network. Each node in this layer represents one feature from the input data.

- **Hidden layers**: Between the input and output, one or more hidden layers transform the data. Each hidden layer contains neurons that apply a linear transformation to the incoming values, followed by a nonlinear activation function. These nonlinearities are crucial, as they allow the network to capture complex patterns rather than just simple linear relationships.

- **Output layer**: This produces the final prediction, which may be a class label in classification problems or a continuous value in regression tasks.

The defining feature of an FNN is its feedforward flow of information. Data passes sequentially from one layer to the next, without any loops or feedback connections. At each step, the

network builds increasingly abstract representations of the input. For example, in image recognition, early layers may detect edges or simple textures, while deeper layers capture higher-level concepts such as shapes or objects.

By stacking transformations in this way, FNNs are able to map complex input-output relationships. Although modern neural network architectures (like recurrent or convolutional networks) extend beyond this basic idea, the feedforward model remains the conceptual backbone of deep learning.

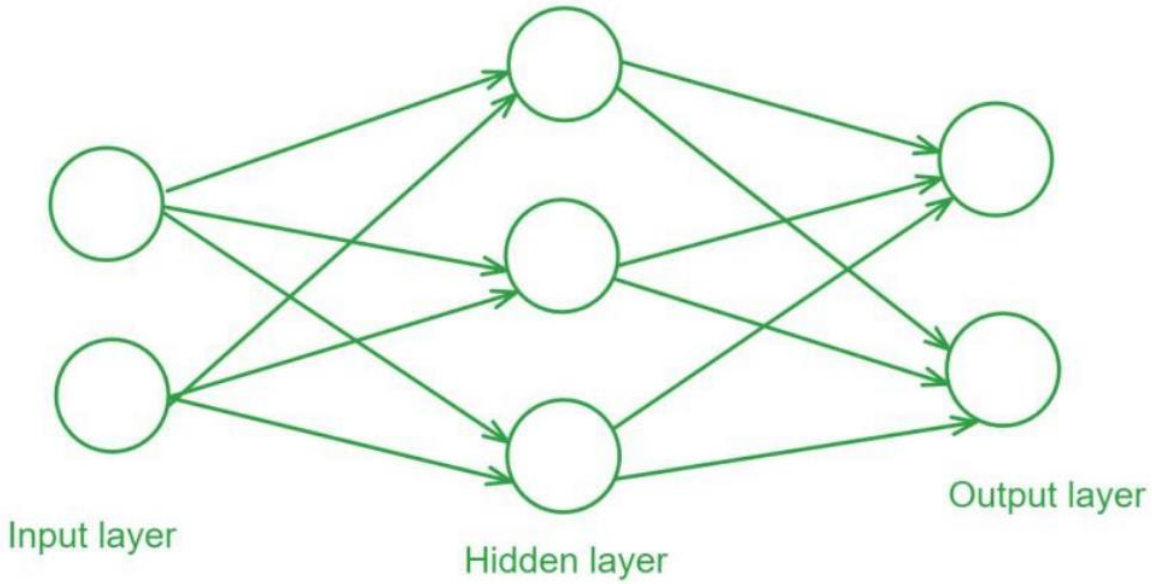

Figure 3 - Feed Forward Neural Network [5]

3.3.2 Convolutional Neural Networks (CNNs)

A Convolutional Neural Network (CNN) is designed to process spatial data, such as images, by mimicking the human visual system. Its core functionality relies on three key ideas: local connectivity, parameter sharing, and hierarchical learning.

The process begins with a **convolutional layer**. Instead of connecting every neuron to every pixel in the input image, this layer uses small filters (or kernels) that slide across the image. Each filter detects a specific local feature, like an edge or a blotch of color, producing a feature map that highlights where that feature occurs. This allows the network to learn patterns regardless of their position, a principle called translation invariance.

Next, **pooling layers** (typically max pooling) downsample these feature maps. They reduce their spatial size by taking the maximum value from small regions, preserving the most activated features while making the representation more manageable and robust to small shifts.

Through a stack of alternating convolutional and pooling layers, the network builds a hierarchical representation. Early layers learn simple low-level features (edges, corners), while deeper layers combine them into complex high-level patterns (eyes, wheels, entire objects).

Finally, the processed features are flattened and fed into traditional **fully connected layers** to perform the final classification, assigning a label based on the complex features extracted.[6]

Below is a figure depicting everything described above.

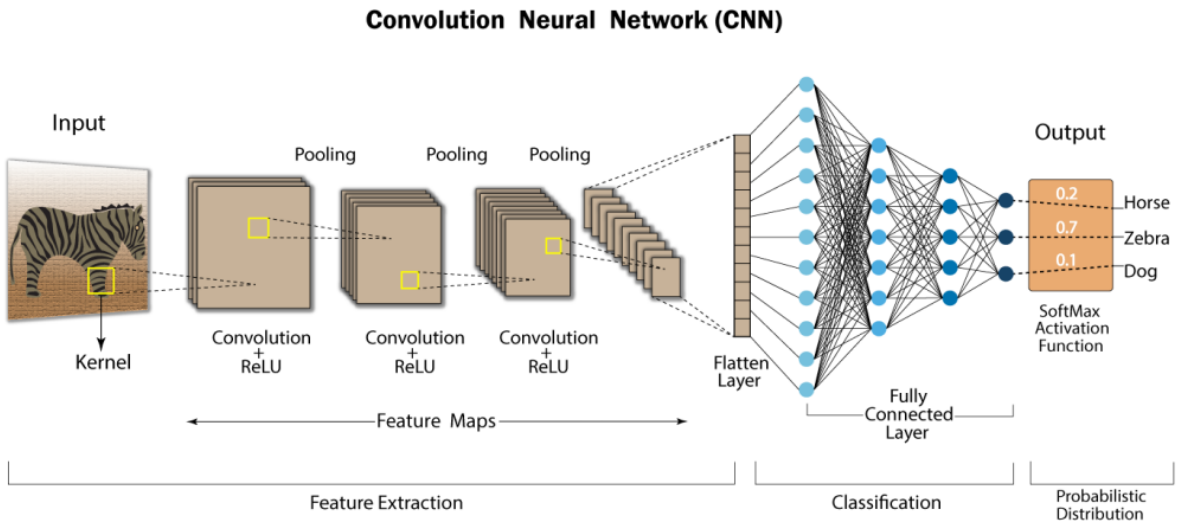

Figure 4 - Convolutional Neural Network[6]

3.3.3 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) [45] are a class of neural networks engineered for processing sequential data. Unlike traditional feed-forward networks, RNNs incorporate loops that enable them to retain a "memory" of previous inputs by updating an internal hidden state at each time step. This makes them ideal for tasks where context and order are critical, such as natural language processing, audio analysis, and time-series prediction.
At each step, an RNN processes an input alongside its hidden state from the prior step. This combination undergoes a transformation—typically non-linear—to produce an updated hidden state and, if needed, an output. The hidden state acts as a dynamic context, evolving as the

network processes the sequence, allowing it to capture temporal patterns and dependencies effectively.

RNN Layers
An RNN typically includes:

- **Input Layer**: Accepts sequential data, such as words in a sentence or values in a time series.

- **Recurrent Layers**: Maintains temporal context through hidden states, enabling the network to "remember" past information.
- **Output Layer**: Generates predictions, either as a single output for the entire sequence (e.g., sentiment classification) or as a sequence of outputs (e.g., machine translation).

Challenges in Training
Training RNNs using backpropagation through time (BPTT) can be problematic due to vanishing or exploding gradients. Vanishing gradients hinder learning of long-term dependencies, as updates become negligible, while exploding gradients can destabilize the training process. To address these, advanced architectures like Long Short-Term Memory (LSTM) [46] units and Gated Recurrent Units (GRUs) introduce gating mechanisms to selectively retain or discard information, enabling better handling of long sequences.

Strengths and Applications
Recurrent Neural Networks (RNNs) excel in scenarios that require sequential understanding. They are widely used in speech recognition, where they model audio signals for accurate transcription. In time-series analysis, RNNs are effective at predicting trends in financial markets or environmental data. They are also employed in text generation, where their ability to create coherent sequences makes them valuable for applications such as automated writing or dialogue systems.
Another key advantage of RNNs lies in their compact design. This makes them particularly useful in resource-limited settings, such as embedded systems or real-time applications, where low computational overhead and continuous data processing are crucial.
Despite the dominance of Transformers in many sequence-processing tasks—thanks to their scalability and parallelization—RNNs remain relevant in specialized domains. Their sequential processing nature is well suited to applications where data arrives incrementally or where model efficiency is more important than raw performance. Moreover, advancements such as Long Short-Term Memory networks (LSTMs) [46] and Gated Recurrent Units (GRUs) have helped overcome some of the traditional limitations of RNNs, further extending their utility in modern contexts.

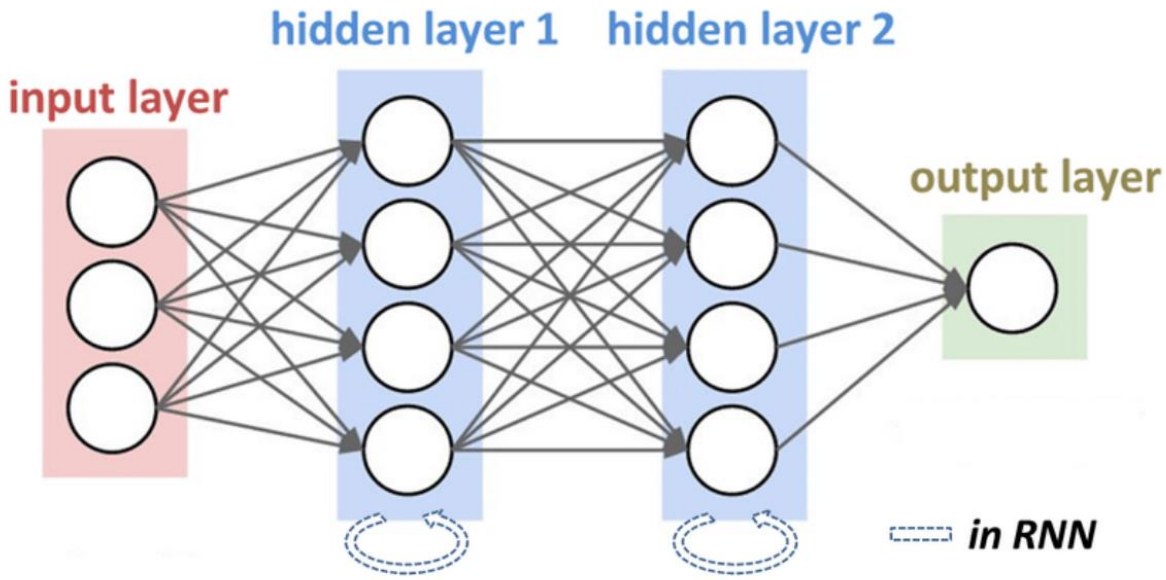

Figure 5 - The architecture of an RNN [47]

3.3.4 The Transformer Architecture

Introduction to the Transformer

In the realm of deep learning for sequence modeling and transduction tasks, such as machine translation and language modeling, traditional approaches have long relied on recurrent neural networks (RNNs) or convolutional neural networks (CNNs) that we mentioned earlier .These models, while effective, suffer from inherent limitations in parallelization and handling long-range dependencies due to their sequential nature. The Transformer [8], introduced in 2017, represents a paradigm shift by avoiding recurrence and convolution entirely, relying solely on attention mechanisms to process sequences.

The core innovation of the Transformer is its use of self-attention, which allows the model to weigh the importance of different parts of the input sequence relative to each other, regardless of their positional distance. This architecture not only enables greater parallelization during training but also achieves superior performance on benchmark tasks. Experiments on machine translation datasets, such as WMT 2014 English-to-German and English-to-French, demonstrated that Transformer models outperform previous state-of-the-art systems, achieving BLEU scores of 28.4 and 41.8, respectively, with significantly reduced training times [8].

This chapter delves into the architecture, components, and empirical results of the Transformer, drawing directly from its foundational design. We explore how attention mechanisms form the backbone of the model, why self-attention offers advantages over traditional layers, and the training systems that enable its efficiency.

Background and Motivation

Sequence transduction models typically involve an encoder that maps an input sequence to a continuous representation and a decoder that generates an output sequence from this representation. Recurrent models, like long short-term memory (LSTM) networks, compute hidden states sequentially, limiting parallelization and making it challenging to capture long-range dependencies. Convolutional alternatives, such as ByteNet and ConvS2S, compute representations in parallel but require stacked layers to connect distant positions, with computational costs growing with sequence length. Attention mechanisms address these issues by modeling dependencies without regard to distance, often used in conjunction with RNNs. Self-attention extends this by relating positions within a single sequence. The Transformer builds on this, using self-attention exclusively to compute input and output representations, reducing sequential operations to a constant number and enabling efficient learning of global dependencies.

Model Architecture

The Transformer follows a standard encoder-decoder structure (which we will explain below) but replaces recurrent or convolutional layers with stacked self-attention and feed-forward networks. Both the encoder and decoder consist of six identical layers, with all sub-layers producing outputs of dimension $d_{\text{model}} = 512$ to support residual connections.

Encoder and Decoder Stacks

The encoder comprises six layers, each with two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. Residual connections and layer normalization are applied around each sub-layer: LayerNorm$(x + \text{Sublayer}(x))$.

The decoder mirrors this but adds a third sub-layer for multi-head attention over the encoder's output. To maintain auto-regressivity, the decoder's self-attention masks future positions, ensuring predictions for position i depend only on prior outputs.

The figure below shows how encoder and decoder are structured within the Transformer architecture.

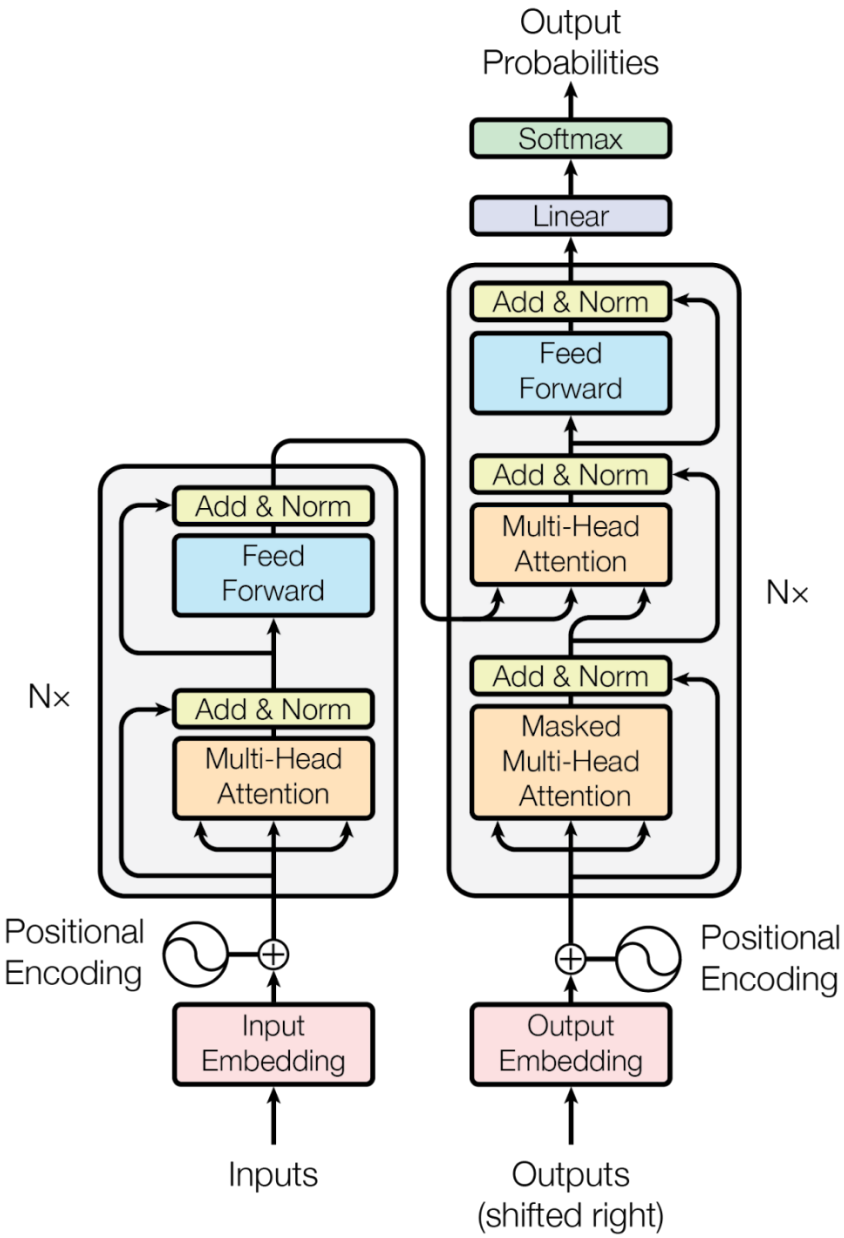

Figure 6 - The Transformer - model architecture.[8]

Attention Mechanisms

Attention maps a query and key-value pairs to an output, computed as a weighted sum of values based on query-key compatibility.

Scaled Dot-Product Attention

The Transformer's attention is "Scaled Dot-Product Attention," (see figure) where queries Q, keys K, and values V are matrices. The output is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaling by $\sqrt{d_k}$ (where d_k is the key dimension) prevents large dot products from saturating the softmax. This is faster and more efficient than additive attention for high dimensions.

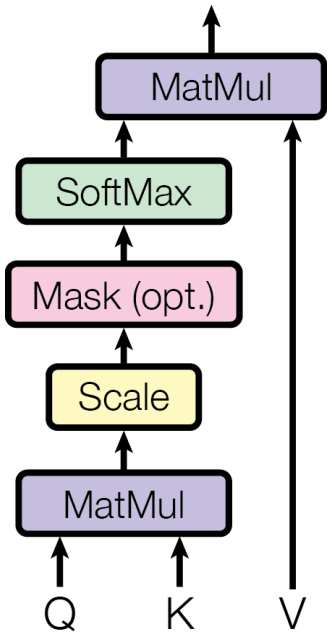

Figure 7 - Scaled Dot-Product Attention. [8]

Multi-Head Attention

Instead of one attention function, multi-head attention projects queries, keys, and values $h = 8$ times into lower dimensions ($d_k = d_v = 64$)(values used in the original "Attention is All you need paper" [8]), applies attention in parallel, and concatenates results before a final projection:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$.

This allows joint attention to different subspaces, maintaining computational cost similar to single-head attention.

Attention is applied in three ways:

- Encoder self-attention: Keys, values, and queries from the previous encoder layer.
- Decoder self-attention: Similar, but masked to prevent attending to future positions.
- Encoder-decoder attention: Queries from the decoder, keys and values from the encoder.

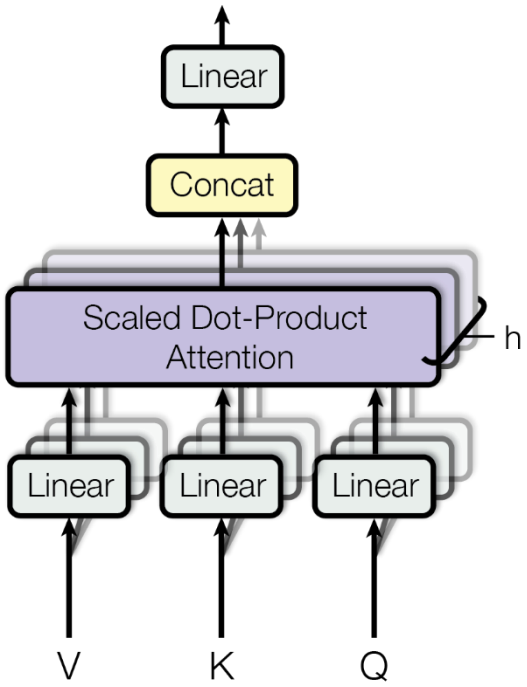

Figure 8 - Multi-Head Attention consists of several attention layers running in parallel. [8]

Position-wise Feed-Forward Networks

Each layer includes a feed-forward network applied identically to each position:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

With inner dimension $d_{ff} = 2048$, this adds non-linearity while preserving parallelism.

Embeddings and Softmax

Input and output tokens are embedded into d_{model}-dimensional vectors. The decoder uses a linear transformation and softmax for next-token probabilities, sharing weights between embedding layers and the pre-softmax linear layer, scaled by $\sqrt{d_{\text{model}}}$.

Positional Encoding

Since the Transformer architecture does not incorporate any notion of sequential order by default, positional information must be introduced separately. To incorporate this sequence order, fixed sinusoidal positional encodings are added to embeddings:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

This allows extrapolation to longer sequences and models relative positions linearly.

Advantages of Self-Attention

Self-attention layers offer key benefits over recurrent and convolutional layers for mapping sequences:

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Figure 9 - Efficiency Metrics of different layer types including self-attention. [8]

Self-attention connects all positions with constant operations and path lengths, aiding long-range dependency learning. It is faster for typical sentence lengths ($n < d$) and yields interpretable attention patterns, often aligning with syntactic structures.

Conclusion and Future Directions

The Transformer demonstrates that attention alone suffices for state-of-the-art sequence transduction, offering parallelization, efficiency, and superior quality. It is used by many state of the art models like GPT(decoder-only), BERT (encoder-only), BART (encoder and decoder) and is the basis for LLMs which we will analyze later in this thesis.

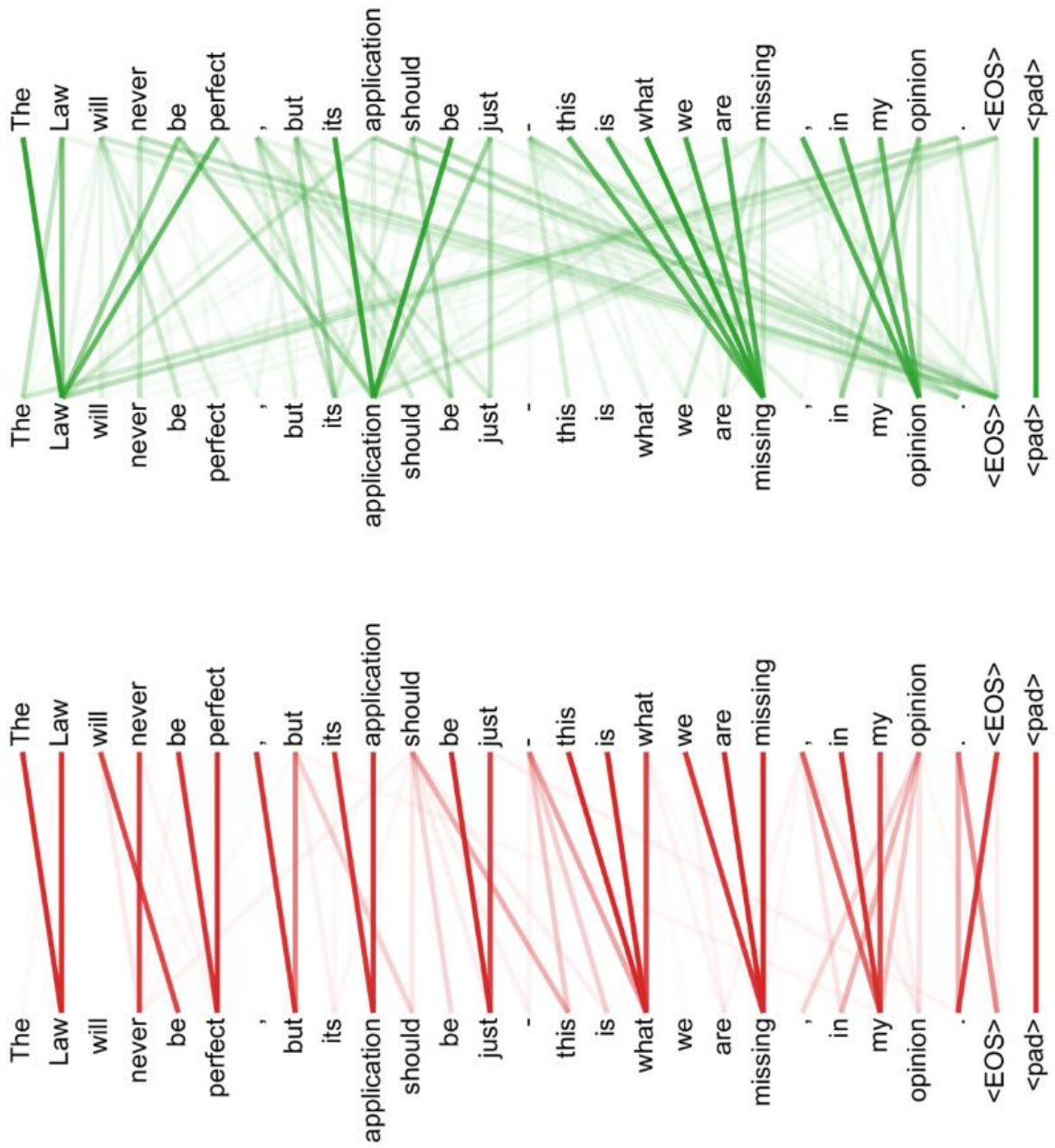

Figure 10 - Example where 2 heads from the encoder self-attention learned to perform different tasks. [8]

4 Natural Language Processing

4.1 Introduction to NLP

The main problem that NLP solves is the understanding of human language. Since computers understand binary code, communication with computers through human language has been made possible thanks to NLP [9]. It is a critical bridge between human communication and computer understanding, empowering machines to interpret, generate, and learn from human language to perform valuable tasks. It is the driving force behind many of the intelligent technologies we interact with daily, making our interactions with machines more natural, efficient, and insightful.

4.2 Definition of Natural Language Processing (NLP)

Natural Language Processing (NLP) is a branch of artificial intelligence (AI) that gives computers the ability to read, understand, interpret, and generate human language. It combines computational linguistics with sophisticated statistical, machine learning, and deep learning models. The primary goal of NLP is to enable computers to process and analyze large volumes of natural language data, deriving meaningful insights and facilitating seamless interaction between humans and machines.
In simpler words, NLP is the technology that allows software applications to understand what we are saying or writing, making sense of its nuances and intent, just as another human would.

4.3 Tokenization

4.3.1 Definition & Importance of Tokenization

Tokenization is one of the most fundamental steps in the Natural Language Processing (NLP) pipeline. It involves splitting raw text into smaller **tokens**—such as words, subwords, characters, or sentences, that serve as the basic units for downstream processing [10], [48].Each token is a semantic unit that can then be processed, analyzed, or transformed into features for

machine learning models. Tokenization not only aids in dissecting unstructured text into manageable parts but also sets the foundation for the functionality of LLMs.

4.3.2 Types of Tokenization

Different strategies are employed depending on the language and task. Here are the most common [10]:

Word Tokenization

Most straightforward: splits text by spaces or punctuation. This tokenization type works well for languages with clear word delimiters like English.

- Example: "Natural Language Processing" → ["Natural", "Language", "Processing"].

Character Tokenization

Breaks text into individual characters.This type of tokenization for misspelling correction, languages without clear boundaries, or highly compressed vocabularies. However, it creates vastly longer sequences and loss of higher-level semantic structure [49].

- Example: "hello" → ["h", "e", "l", "l", "o"].

Subword Tokenization

Splits words into subunits (smaller than words but larger than characters) often using algorithms like Byte-Pair Encoding (BPE) [50], WordPiece, or SentencePiece.

- Example: “unhappiness” → ["un", "happi", "ness"] (depending on training)[51].

Sentence Tokenization

Segments text into sentences.

- Example: Splitting "This is one sentence. This is another." into two elements.

4.3.3 Importance of Tokenization

Tokenization converts raw text into quantifiable units that algorithms can handle, such as vocabulary, counts, or embeddings.
It also helps build the vocabulary (a mapping of unique tokens to IDs) which is essential for both traditional methods like Bag-of-Words and TF-IDF, as well as for modern neural architectures.
In particular, for Large Language Models (LLMs), tokenization influences many aspects of performance, from spelling and encoding issues to handling non-English languages effectively.

4.3.4 Challenges & Considerations of Tokenization

Tokenization is a very useful technique and plays a crucial role in LLMs. However, it faces several challenges. One major issue lies in the ambiguity of languages that lack clear word separators, such as Chinese, Japanese, or Arabic, which makes segmentation far more complex.
Another difficulty is handling contractions, special characters, URLs, and symbols, all of which require careful preprocessing.
Out-of-vocabulary (OOV) words present an additional challenge, often resulting in unknown token placeholders. This reduces both interpretability and flexibility in models.
Finally, while subword methods help reduce vocabulary size, they can sometimes hinder generalization—particularly in morphological or low-resource languages. For instance, SentencePiece has been shown to outperform Byte Pair Encoding (BPE) in some cases because it better preserves morphological structures [52].

4.4 Text Preprocessing Basics

4.4.1 Introduction to Text Preprocessing

Before raw text can be understood by a machine learning model, it must be cleaned and standardized. This process, known as text preprocessing, transforms unstructured text into a structured format. Three fundamental techniques in this phase are the handling of stopwords, stemming, and lemmatization.

4.4.2 Stopwords

In any language, certain words are extremely common but carry little substantive meaning on their own. These include articles (e.g., "a," "an," "the"), conjunctions (e.g., "and," "but," "or"), and prepositions (e.g., "in," "on," "at"). These are called stopwords [11].
The primary goal of removing stopwords is to reduce dimensionality and focus computational resources on the words that truly contribute to the meaning of a text. By filtering them out, we decrease the size of our dataset and help highlight the more meaningful terms.
For example, in the sentence "The quick brown fox jumps over the lazy dog," removing stopwords would yield: ['quick', 'brown', 'fox', 'jumps', 'lazy', 'dog'].
We should note here that stopword removal is not always beneficial. For tasks like sentiment analysis or language translation, where context and nuance are critical, these "small" words can be very important.

4.4.3 Stemming

Stemming [12] is a crude heuristic process that chops off the ends of words to reduce them to a common base or root form, known as a stem. The stem may not be a valid word in itself. The algorithm applies a series of rules (e.g., removing "-ing," "-ed," "-s") without any understanding of the word's context.
The Porter’s Stemmer [53] is a classic and widely used algorithm. Its strength is its simplicity and speed.

- Example:
    - "jumps", "jumping", "jumped" → "jump"
    - "running", "runner", "runs" → "run"
    - "university", "universal" → "univers" (Not a real word)

Stemming is fast and effective for broad term consolidation, but its error-prone nature can lead to strange results and a loss of meaning.

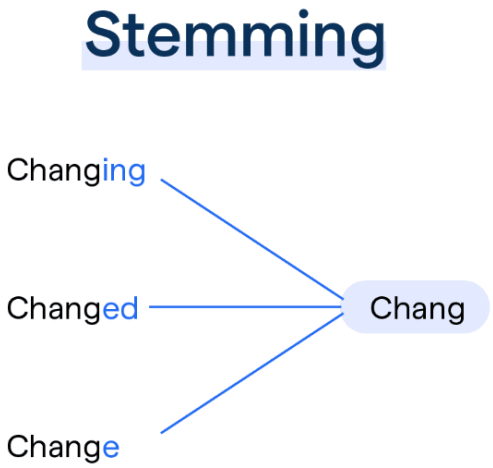

Figure 11 - The words "Changing", "Changed" and "Change" and the stem produced from these words [54]

4.4.4 Lemmatization

Lemmatization [13], in contrast, is a more sophisticated and computationally intensive process. It aims to reduce a word to its canonical form, known as a lemma, which is a valid dictionary word.
Unlike stemming, lemmatization considers the *context* and *part of speech* (e.g., verb, noun) of a word. It uses a detailed vocabulary and morphological analysis to return the base word [55].

- Example:
  - "better" (adjective) → "good" (The lemma)
  - "running" (verb) → "run"
  - "meeting" (as a noun) remains "meeting"; as a verb, it becomes "meet".

While more accurate than stemming, lemmatization is slower and requires more linguistic resources, such as a part-of-speech tagger.

Together, these techniques form a core part of the preprocessing pipeline, enabling models to work with cleaner, more consistent, and more meaningful textual data.

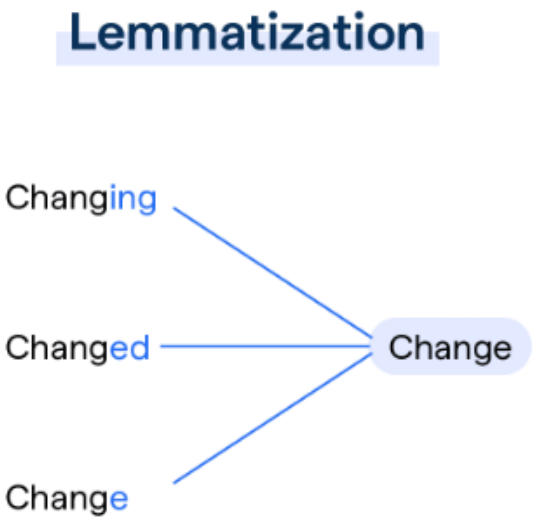

Figure 12 - The words "Changing", "Changed" and "Change" and their lemma [56]

4.5 Parts of Speech Tagging

Part-of-Speech (POS) Tagging is the task of labeling words in a sentence with their grammatical roles (such as noun, verb, adjective, or adverb) based on both meaning and context. For example, in the sentence *"The quick brown fox jumps over the lazy dog,"* the words are tagged as determiner, adjective, noun, verb, and so on. This process is foundational in Natural Language Processing (NLP) because it provides structure to raw text and supports higher-level tasks like parsing, machine translation, and question answering [57].

Early methods for POS tagging were simple but paved the way for modern systems. Default tagging assigns the same tag to every word, often "noun," and serves only as a baseline. Slightly more sophisticated are rule-based systems [58], which rely on handcrafted linguistic rules, such as tagging words ending in *-tion* as nouns. While interpretable, these systems fail with unfamiliar language patterns.
A major breakthrough came with statistical and probabilistic methods like Hidden Markov Models and N-gram taggers . These use training data to estimate the likelihood of tag sequences, enabling more accurate predictions. Accuracy can reach over 90%, and taggers like

CLAWS became widely adopted. To handle unknown words, backoff strategies combine multiple taggers, falling back to simpler ones when needed.
Today, neural models dominate POS tagging. BiLSTMs [59] and transformers use word embeddings and contextual representations to achieve state-of-the-art accuracy, often above 97%. These models learn directly from large annotated corpora, reducing the need for hand-designed rules.

Tag Sets and Standards
POS tagging requires a consistent set of labels. The Penn Treebank tag set is one of the most widely used in English, offering fine-grained categories like singular noun (NN) or past-tense verb (VBD) [60]. Meanwhile, the Universal POS tag set provides broader categories (NOUN, VERB, ADJ, etc.) [61] , designed for multilingual NLP. The choice of tag set depends on the complexity of the language and the level of detail required.

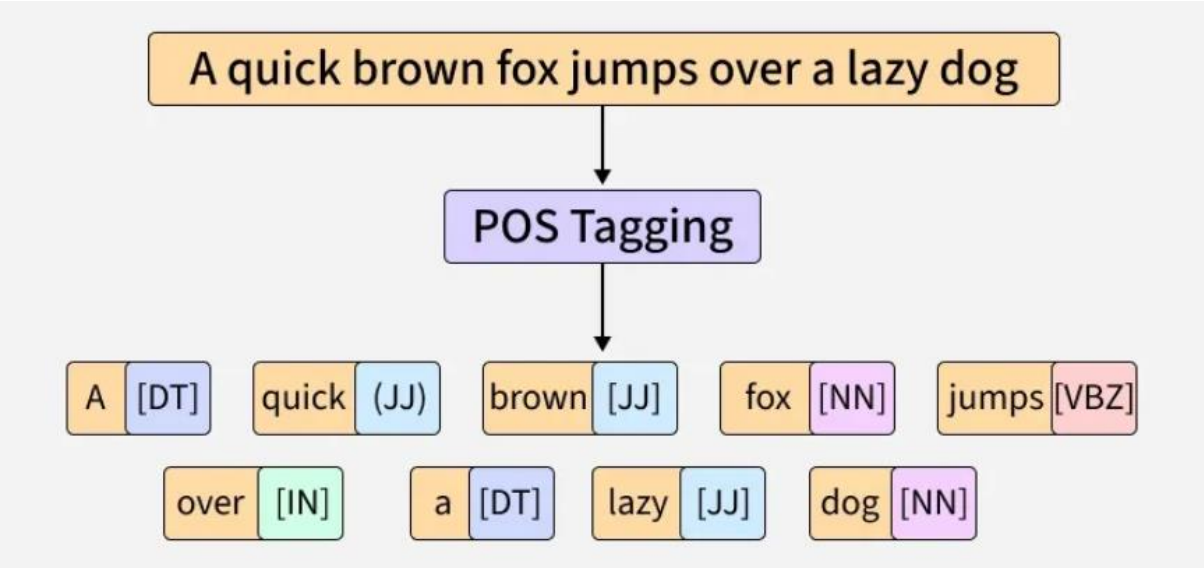

Figure 13 - Example of Part-of-Speech (POS) tagging applied to the sentence "A quick brown fox jumps over a lazy dog". Each word is annotated with its corresponding POS tag [57]

POS tagging is not just an academic exercise, it strengthens downstream NLP applications by clarifying sentence structure and improving feature extraction. Even in the deep learning era, where models can operate on raw embeddings, POS information is still useful for parsing, information extraction, and text analytics. Combining POS features with neural models often leads to more robust performance.

To conclude, POS tagging has evolved from default and rule-based systems to probabilistic models and now to neural networks. Despite these changes, its role remains central: transforming raw text into structured input for deeper language understanding. The balance between interpretability and accuracy continues to shape its development, making it both a historical and modern cornerstone of NLP.

4.6 Named Entity Recognition

Named Entity Recognition (NER) is a key task in Natural Language Processing (NLP) that focuses on identifying and categorizing important elements in text into predefined types such as people, organizations, locations, dates, quantities, and other proper nouns. It allows machines to extract meaningful information from unstructured text by pinpointing the most relevant entities [15], [62]. For instance, in the sentence *"Barack Obama was born in Hawaii in 1961 and later became the President of the United States"*, an NER system would recognize "Barack Obama" as a person, "Hawaii" as a location, "1961" as a date, and "President of the United States" as a title or role. Similarly, in *"Google announced the launch of its new Pixel phone in October 2020"*, "Google" is an organization, "Pixel" is a product, and "October 2020" is a date.

NER plays a crucial role in various applications such as information extraction, question answering, knowledge graph construction, and chatbots. Early NER systems relied on rule-based or dictionary-based approaches, which were limited by their dependence on manually curated lists [63]. Modern methods predominantly use machine learning and deep learning, particularly sequence models like Conditional Random Fields (CRFs) or transformer-based models such as BERT, which consider both the context of words and their relationships within a sentence [64]. By accurately identifying entities, NER enables NLP systems to structure raw text into actionable knowledge, making it easier to analyze, summarize, or interact with large volumes of textual data.

4.7 Feature Representation Methods

One of the most fundamental aspects of Natural Language Processing (NLP) is how textual data is represented in a way that computational models can process effectively. Since raw text cannot be directly interpreted by algorithms, it must be transformed into structured numerical representations known as *features*. The quality and suitability of these representations play a crucial role in determining the performance of downstream NLP tasks such as text classification, machine translation, and sentiment analysis.
Over time, a variety of feature representation methods have been developed, ranging from traditional statistical approaches like bag-of-words and TF-IDF to more sophisticated distributed representations such as word embeddings and contextualized vector models. Each method captures different aspects of language, such as frequency, semantics, or context, and thus offers distinct advantages and limitations.
This chapter provides an overview of the most widely used feature representation methods in NLP, outlining their underlying principles, strengths, and application areas.

4.7.1 One-Hot Encoding

One-Hot Encoding transforms a *categorical feature* into multiple binary (0/1) features—one for each unique category. Precisely one column per category contains a 1 (the “hot” bit), all others are 0 [17], [65].

This method ensures that algorithms interpret categories without implying any ordinal relationship. For example, encoding colors or fruit types avoids mistakenly suggesting that “Apple > Orange” simply because of numeric label assignment [66].

Below is an example of how we would categorize data using One-Hot-encoding.

Fruit	Categorical value of fruit	Price
apple	1	5
mango	2	10
apple	1	15
orange	3	20

Figure 14 - Initial Dataset with Integer Encoded Categorical Feature. [66]

Fruit_apple	Fruit_mango	Fruit_orange	price
1	0	0	5
0	1	0	10
1	0	0	15
0	0	1	20

Figure 15 - Dataset from figure 9 Transformed via One-Hot Encoding. [66]

4.7.2 Bag of Words

A fundamental challenge in Natural Language Processing (NLP) and machine learning is the representation of textual data in a form recognizable to computational algorithms, which typically require fixed-length numerical input vectors. The Bag-of-Words (BoW) model is a simple, yet powerful, feature extraction technique that addresses this challenge by quantifying text based on word frequency.

The model operates on a core premise: it represents a document as a multiset (a "bag") of its words, disregarding all information pertaining to grammar, syntax, and word order . The representation is solely concerned with the occurrence and frequency of words within the

document. This simplification allows for a high-dimensional feature space where each unique word in the corpus vocabulary constitutes a separate dimension [19].

The formal construction of a BoW representation involves a two-step process [20]:

1. **Vocabulary Generation:** Given a corpus of documents $D = \{d_1, d_2, \ldots, d_n\}$, a unified vocabulary V is constructed. This vocabulary is the set of all unique words (or tokens) found across the entire corpus after common preprocessing steps such as lowercasing and removal of punctuation.

   Let $V = \{w_1, w_2, \ldots, w_m\}$ be the vocabulary of size m.

2. **Vectorization:** Each document d_i in the corpus is then encoded as a numerical feature vector v_i of length m. The value of each element in the vector corresponds to the frequency (count) of the corresponding word from V within the document d_i.

   Thus, $v_i = (\text{count}(w_1, d_i), \text{count}(w_2, d_i), \ldots, \text{count}(w_m, d_i))$.

Example

Consider a minimal corpus C comprising two document sentences:

- d_1: "The movie was good good good!"
- d_2: "The movie was bad."

The application of the BoW model proceeds as follows:

4. **Vocabulary Construction (V):** After lowercasing and removing punctuation, the unique words form the vocabulary: $V = \{\text{the,movie,was,good,bad}\}$, with $m = 5$.

5. **Frequency Vectorization:** Each document is mapped onto the 5-dimensional feature space defined by V.
   - For d_1: The word "good" appears three times. The resulting vector is: $v_1 = (1,1,1,3,0)$
   - For d_2: The resulting vector is: $v_2 = (1,1,1,0,1)$

This transformation yields a term-document matrix A, where rows represent documents and columns represent terms (words):

Document	the	movie	was	good	bad
d_1	1	1	1	3	0
d_2	1	1	1	0	1

Figure 16 - Bag of Words implementation results matrix.

Applications and Limitations

The BoW model serves as a cornerstone for numerous NLP tasks, including document classification, topic modeling, and information retrieval. Its efficacy stems from its ability to capture lexical statistics that are often strong indicators of document content and sentiment (e.g., a high frequency of "good" versus "bad").

However, the model's simplicity introduces significant limitations [20]:

- **Loss of Semantic Structure:** By discarding word order and syntactic context, the model fails to capture meaning derived from phrasing and grammar. For instance, the phrases "not bad" and "not good" would be decomposed into identical constituent words, leading to similar vector representations despite their opposing sentiments.
- **High Dimensionality:** The vocabulary size m can be very large, leading to sparse, high-dimensional vectors that pose computational challenges.
- **Vocabulary Sensitivity:** The model's performance is highly dependent on the comprehensiveness and cleanliness of the vocabulary V.

Despite these limitations, the Bag-of-Words model remains a critical baseline and a computationally efficient method for initial feature extraction in text mining workflows. Subsequent techniques, such as TF-IDF weighting and N-grams, are often employed to enhance its basic formulation. We will discuss these techniques in the following sections.

4.7.3 N-gram

A primary limitation of the standard Bag-of-Words (BoW) model is its failure to capture any sequential information or syntactic structure within text, treating a document as an unordered collection of words. The N-gram model extends the BoW paradigm to address this shortcoming by incorporating local word order into the feature representation.

An N-gram is defined as a contiguous sequence of N items (typically words) from a given text sample. The value N defines the scope of the context window the model considers:

- A **Unigram** (1-gram) is a single word, which is equivalent to the standard BoW model.
- A **Bigram** (2-gram) is a sequence of two adjacent words (e.g., "was good").
- A **Trigram** (3-gram) is a sequence of three adjacent words (e.g., "the movie was")[67].

By considering these contiguous sequences as individual tokens, the N-gram model can capture limited local context, phrases, and common expressions, thereby enriching the feature set beyond isolated terms [68].

Model Construction

The construction of an N-gram feature space follows a similar procedure to the standard BoW, with a critical difference in the definition of the vocabulary.

1. **N-gram Vocabulary Generation:** Given a corpus D, the vocabulary V_N is constructed from all unique contiguous N-gram sequences found within the corpus. For a bigram model (N=2), the vocabulary consists of all unique two-word pairs.
    - Let $V_N = \{g_1, g_2, \ldots, g_k\}$ be the N-gram vocabulary of size k.
2. **Vectorization:** Each document d_i is encoded as a numerical feature vector v_i of length k. The value of each element corresponds to the frequency of the corresponding N-gram from V_N within the document d_i.
    - Thus, $v_i = (\text{count}(g_1, d_i), \text{count}(g_2, d_i), \ldots, \text{count}(g_k, d_i))$.

This results in a significantly larger and more sparse term-document matrix compared to the unigram BoW, as the number of possible N-grams grows combinatorially with N.

Example

Consider the same minimal corpus C:

- d_1: "The movie was good good good!"
- d_2: "The movie was bad."

The application of a **Bigram (N=2)** model proceeds as follows:

1. **Bigram Vocabulary Construction (V_2):** The unique bigrams from both documents are extracted to form the vocabulary.
    - From d_1: "the movie", "movie was", "was good", "good good", "good good" (duplicate).
    - From d_2: "the movie", "movie was", "was bad".
    - The unique bigram vocabulary is: $V_2 =$ {the movie,movie was,was good,good good,was bad}, with $k = 5$.
2. **Frequency Vectorization:** Each document is mapped onto this 5-dimensional bigram feature space.
    - For d_1: The bigram "good good" appears twice. The resulting vector is: $v_1 = (1,1,1,2,0)$
    - For d_2: The resulting vector is: $v_2 = (1,1,0,0,1)$

This yields the following bigram term-document matrix:

Document	the movie	movie was	was good	good good	was bad
d_1	1	1	1	2	0

Document	the movie	movie was	was good	good good	was bad
d_2	1	1	0	0	1

Figure 17 - Bigram implementation results matrix.

Applications and Limitations

N-gram models are a powerful enhancement to text representation and are widely used in [69]:

- **Statistical Language Modeling:** For predicting the next word in a sequence, which is fundamental in speech recognition and machine translation.
- **Text Classification:** Capturing phrases (e.g., "not good" vs. "very good") can significantly improve sentiment analysis and topic classification accuracy over unigrams.
- **Information Retrieval:** Improving search relevance by matching multi-word queries to document phrases.

Despite its utility, t he N-gram model introduces significant challenges that must be acknowledged. The number of potential features grows exponentially as *N* increases, leading to an explosion in the dimensionality of the feature space. This exponential growth results in extreme data sparsity, a manifestation of the so-called "curse of dimensionality," and imposes severe computational overhead for both model training and feature storage. A direct consequence of this sparsity is the problem of unseen events, where many linguistically plausible N-grams are absent from the training corpus. This necessitates the use of specialized techniques, such as smoothing, to assign non-zero probabilities to these unseen sequences in language modeling tasks. Furthermore, while the model represents a substantial improvement over a simple Bag-of-Words by capturing local context, its fundamental limitation remains a fixed context window. The context captured is strictly limited to *N* contiguous words, meaning long-range linguistic dependencies cannot be captured without employing prohibitively large and impractical values of *N*.

In conclusion, the N-gram model provides a crucial middle ground between the simplicity of a Bag-of-Words and the complexity of modern deep learning models, allowing for the incorporation of local context at a manageable computational cost for many practical applications [69].

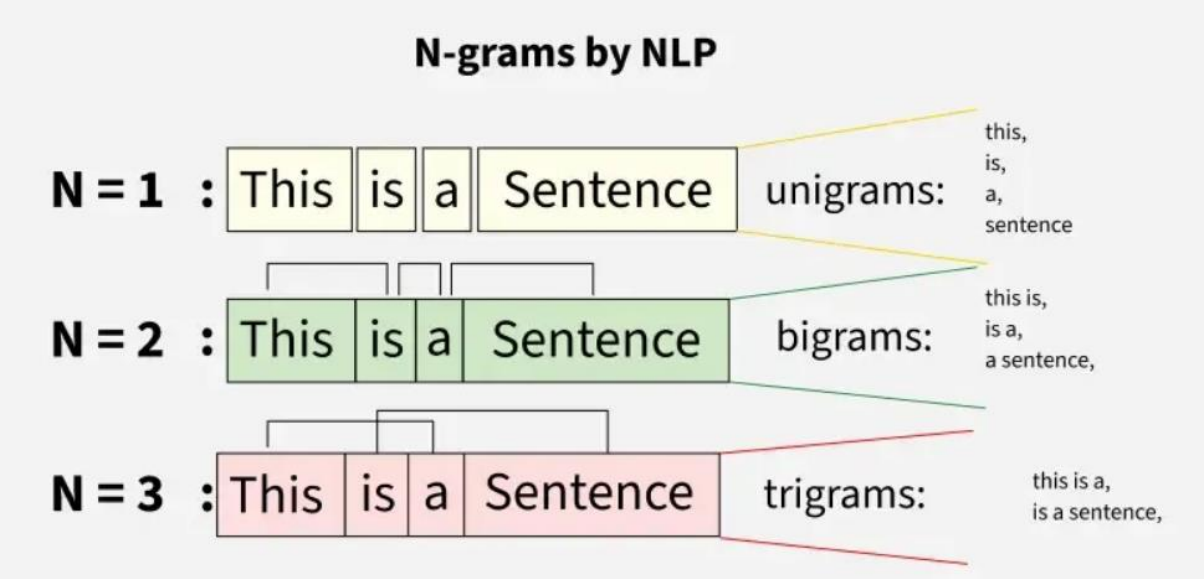

Figure 18 - Implementation of unigram, bigram, and trigram on the same sentence [69]

4.7.4 Term Frequency-Inverse Document Frequency (TF-IDF)

A significant limitation of both the standard Bag-of-Words and N-gram models is their reliance on raw term frequency (TF). These approaches assign higher weight to terms that appear more frequently, under the assumption that frequent terms are more important. However, this assumption is flawed, as the most frequent terms in a language (e.g., "the", "is", "of")—known as stop words—are often the least informative. The Term Frequency-Inverse Document

Frequency (TF-IDF) weighting scheme addresses this issue by refining the concept of term importance.
TF-IDF is a statistical measure that evaluates the importance of a word to a document within a collection (corpus). The core intuition is that a term is important to a specific document if it appears frequently *within that document* (high term frequency) but rarely *across all other documents* in the corpus (high inverse document frequency). This combination effectively discounts the weight of common words that appear in many documents and boosts the weight of terms that are unique and discriminative.

The TF-IDF weight for a term t in a document d, belonging to a corpus D, is the product of two components [23]:

1. **Term Frequency (TF):** This measures how frequently a term occurs in a document. It is often normalized (e.g., by the document length) to prevent a bias towards longer documents. A common normalization is:

$$\text{tf}(t, d) = \frac{\text{count of } t \text{ in } d}{\text{total number of terms in } d}$$

2. **Inverse Document Frequency (IDF):** This measures how much information the word provides, i.e., whether it is common or rare across all documents. It is calculated as the logarithm of the inverse proportion of documents that contain the term:

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

where N is the total number of documents in the corpus D, and $|\{d \in D : t \in d\}|$ is the number of documents containing the term t.

The final TF-IDF score is the product of these two factors:

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

Each document is subsequently represented as a vector of these TF-IDF scores for every term in the vocabulary, rather than raw counts.

Example
Consider a corpus D containing three documents:

- d_1: "the movie was good good good"
- d_2: "the movie was bad"
- d_3: "the book was good"

We will calculate the TF-IDF weight for the word "good" in document d_1.

1. **Term Frequency (TF) for "good" in d_1:**
    - Raw count = 3
    - Total terms in d_1 = 6
    - tf("good", d_1) $= 3/6 = 0.5$
2. **Inverse Document Frequency (IDF) for "good":**
    - Total documents $N = 3$
    - Number of documents containing "good": $|\{d_1, d_3\}| = 2$
    - idf("good", D) $= \log(3/2) = \log(1.5) \approx 0.176$
3. **TF-IDF Calculation:**
    - tf-idf("good", d_1, D) $= 0.5 \times 0.176 \approx 0.088$

For comparison, the IDF for a very common word like "the" (which appears in all 3 documents) would be $\log(3/3) = \log(1) = 0$, resulting in a TF-IDF weight of zero, effectively filtering it out. Conversely, a rare word like "bad" (which appears in only 1 document) would have a much higher IDF value of $\log(3/1) = \log(3) \approx 0.477$, increasing its relative importance in the document where it appears.

Applications and Limitations

TF-IDF [70] is one of the most ubiquitous weighting schemes in information retrieval and text mining. Its primary applications include:

- **Information Retrieval and Search Engines:** It is the foundational algorithm for ranking search engine results. Documents are ranked by the cosine similarity between their TF-IDF vectors and the TF-IDF vector of the search query, effectively returning documents that contain the query's most discriminative terms at high frequency.
- **Text Classification and Clustering:** By representing documents as TF-IDF vectors, machine learning algorithms can more effectively identify patterns based on salient keywords rather than common but meaningless words.
- **Keyword Extraction:** The terms with the highest TF-IDF scores within a document are often strong candidates for being its key keywords or topics.

However, the TF-IDF model is not without its limitations. Firstly, it remains a **bag-of-words** model and, as such, inherently disregards word order, semantic meaning, and context in the same way as the standard BoW model. Phrases and syntactic relationships are lost. Secondly, the IDF component is based on a simple document frequency count and does not truly capture the semantic importance of a term; a rare word is not necessarily meaningful. Furthermore, the standard TF-IDF formulation does not account for the specific positions of terms within a document, potentially missing the significance of words appearing in titles or abstracts. Despite these limitations, its computational efficiency, interpretability, and proven effectiveness ensure its continued relevance as a powerful feature extraction technique and a strong baseline for text representation tasks.

4.7.5 Word Embeddings

4.7.5.1 Introduction to Word Embeddings

In natural language processing (NLP), one of the central challenges lies in representing textual data in a way that can be effectively processed by computational models. Traditional methods which we mentioned earlier, such as the bag-of-words or term frequency–inverse document frequency (TF-IDF), treat words as independent and discrete symbols. While these approaches capture statistical information about word occurrence, they fail to represent semantic relationships, contextual similarity, or syntactic structure. This limitation makes it difficult for algorithms to generalize meaning across different but related terms.

Word embeddings [71] were introduced as a powerful solution to this problem. They are vector representations of words in a continuous, high-dimensional space, where semantic similarity between words is reflected in geometric proximity [24]. For example, the vectors for “king” and “queen” will be closer to each other than to unrelated words such as “table” or “car.” This property allows machine learning models to exploit semantic regularities in language and improves performance across a wide range of tasks, including sentiment analysis, machine translation, and information retrieval.

4.7.5.2 Word2Vec

Word2Vec is a technique in natural language processing (NLP) used to represent words as numerical vectors (embeddings) in such a way that words with similar meanings are placed close to each other in a high-dimensional space. It was introduced by Mikolov et al. (2013) [72] and has become one of the foundational methods for learning word embeddings.

The main idea behind Word2Vec is that *the meaning of a word can be inferred from the company it keeps*. For example, words like *“king”* and *“queen”* often appear in similar contexts, so their vector representations should also be close.

Instead of representing words as one-hot vectors (which are sparse and do not capture meaning), Word2Vec learns **dense, low-dimensional representations** that encode semantic and syntactic relationships.

The work of Mikolov et al. (2013) [72] at Google introduced two highly efficient and influential neural network architectures for learning high-quality word embeddings: the Continuous Bag-of-Words (CBOW) model and the Skip-gram model. While both share the core objective of learning useful vector representations, they approach the learning task from converse perspectives, each with distinct strengths and weaknesses.

4.7.5.2.1 The Continuous Bag-of-Words (CBOW) Model

The way that the Continuous Bag-of-Words model [72] operates is, given a context window surrounding a target word, it aims to predict the target word itself. For a given sequence of words, the model uses the surrounding context words (e.g., the *n* words to the left and right) as input to predict the center word.
Architecturally, the CBOW model consists of an input layer, a single projection layer, and an output layer. The context words are represented as one-hot vectors and fed into the network. Their vectors are then averaged (or summed) in the projection layer to form a single, aggregated context vector. This aggregated vector is subsequently used to compute a probability distribution over the entire vocabulary, attempting to maximize the probability of the actual target word. The training objective is to minimize the loss between the predicted word and the true center word.
The primary advantage of the CBOW model is its efficiency. By averaging the context vectors, it smoothes over the individual contributions of each context word, making it robust and particularly fast to train. This characteristic often makes CBOW perform well on smaller datasets and for more frequent words, as it effectively leverages the combined information from the entire context. However, this averaging process can also be a limitation, as the model may lose the distinct information contributed by less frequent but potentially important context words.

4.7.5.2.2 The Skip-gram Model

In contrast to CBOW, the Skip-gram model [72] inverts the predictive task. Instead of predicting a target from its context, Skip-gram uses a central target word to predict its surrounding context words. For a given center word, the model seeks to predict each of the words within a defined context window.
The architecture of Skip-gram is similar to CBOW but is used in reverse. The input is the one-hot vector of the center word, and the output layer is designed to produce multiple outputs—one for each context word position to be predicted. The model learns a vector representation for the center word that is useful for predicting its likely context.

The key strength of the Skip-gram model lies in its effectiveness with infrequent words and its ability to capture more nuanced representations of word relationships. Because it makes multiple predictions from a single input word, it treats each context word as a separate learning example. This provides more training signals for each word-context pair, allowing it to learn better representations for rare words. It often produces embeddings that perform exceptionally well on semantic tasks, such as word analogy tests (e.g., "man is to king as woman is to queen"). A noted trade-off is that Skip-gram typically requires more training time and a larger corpus to achieve its full potential compared to CBOW, due to its larger number of training examples per data point.

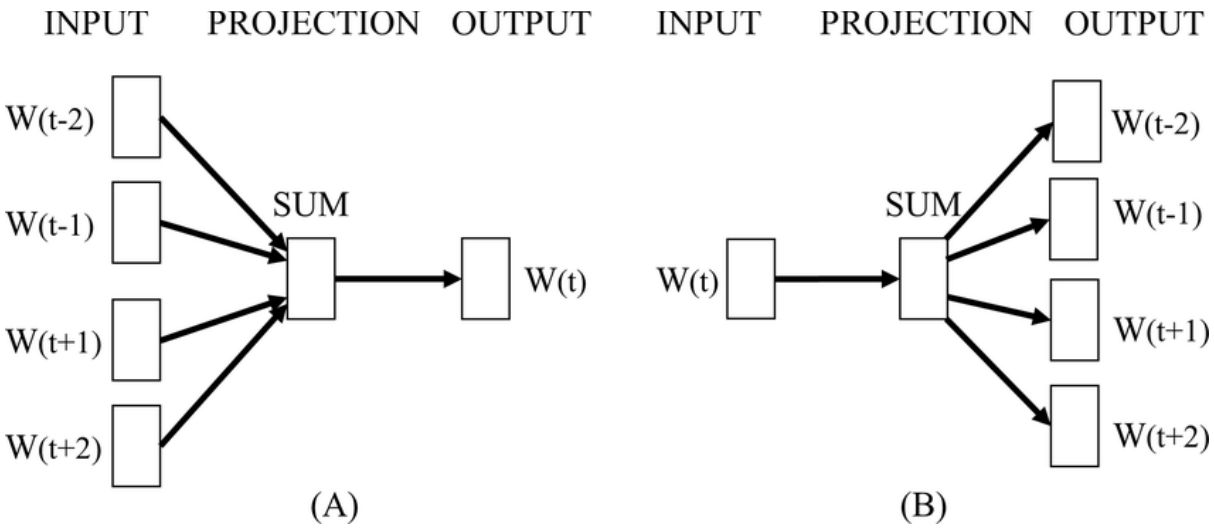

Figure 19 - CBOW architecture (left) and Skip-Gram architecture (right) [73]

Using the above architecture (Word2Vec) we are now able to capture the semantic meaning of the words in the vector instead of just the frequency in which they appear.

4.7.5.3 Word Embedding Similarity Calculation Methods

Word embeddings transform text into continuous vectors, enabling the computation of similarity between words via geometric and algebraic operations in embedding space. Below are mentioned three principal methods for measuring similarity between embedding vectors: **Euclidean Distance**, **Cosine Similarity**, and the **Dot Product.**

Similarity Measures [74]

Let $\mathbf{A} = [a_1, a_2, \ldots, a_n]$ and $\mathbf{B} = [b_1, b_2, \ldots, b_n]$ denote two word embedding vectors.

1. Euclidean Distance
Also referred to as the L2 norm, Euclidean distance measures the straight-line distance between the two vectors:

$$\text{Euclidean distance}(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

A smaller Euclidean distance indicates greater similarity. However, as similarity increases, the value of the distance measure decreases.

2. Cosine Similarity
Cosine similarity assesses the cosine of the angle between two vectors, emphasizing their orientation rather than magnitude:

$$\text{Cosine similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A}^\top \mathbf{B}}{\| \mathbf{A} \| \cdot \| \mathbf{B} \|}$$

Here, similarity increases as the cosine value approaches 1, indicating vectors that point in similar directions.

3. Dot Product
The dot product directly gauges both the alignment and the magnitudes of the vectors:

$$\text{Dot product}(\mathbf{A}, \mathbf{B}) = \mathbf{A}^\top \mathbf{B} = \| \mathbf{A} \| \cdot \| \mathbf{B} \| \cdot \cos(\theta)$$

A higher dot product indicates greater similarity; it increases with both larger magnitudes and greater directional alignment.

5 Generative AI

5.1 Introduction to generative AI

Generative Artificial Intelligence (Generative AI) refers to a class of machine learning models capable of creating new and original content, such as text, images, music, or code, by learning patterns from large datasets. Unlike traditional AI systems, which are primarily designed to analyze data and make predictions, generative models synthesize novel outputs that resemble human-created artifacts. This technology is most prominently represented by deep learning architectures such as Generative Adversarial Networks (GANs) and Transformer-based models, which have enabled significant breakthroughs in natural language processing, computer vision, and creative industries. Generative AI not only demonstrates the rapid advancement of computational intelligence but also raises important questions regarding ethics, authorship, and the future of human–machine collaboration.
In this chapter we will analyze Generative AI and more specifically LLMs which are the main tool we use in the application that we built for this diploma thesis.

5.2 Large Language Models (LLMs)

In previous chapters, we deconstructed the transformer architecture, understanding the mechanics of self-attention in encoders and the autoregressive generation in decoders. We now arrive at the end goal of discussing these concepts: the Large Language Model (LLM). An LLM is not a new fundamental architecture but rather a specific implementation and scaling of the decoder-only or encoder-decoder transformer models. The key differentiator, as the name implies, is scale: scale of parameters, scale of training data, and scale of computational resources. This chapter explores what defines an LLM, how they are built, how they are aligned with human intent, and their inherent capabilities and limitations.

The Core Ingredients of an LLM
While a standard transformer model [8] from the early days might have had tens or hundreds of millions of parameters, an LLM has billions (B) or even trillions (T) of parameters. This scaling is not merely incremental; it leads to emergent abilities unpredictable from smaller models. The creation of an LLM hinges on three pillars:

1. Architecture: Most modern LLMs, such as GPT-4 , LLaMA [75], and PaLM, use a decoder-only transformer architecture. This choice is driven by the success of autoregressive modeling for text generation. The model predicts the next token in a sequence, making it perfectly suited for tasks like conversation, story writing, and code

completion. The encoder-decoder architecture is still prevalent for specific tasks like translation (e.g., Google's T5 frames all tasks as a text-to-text problem).

2. Dataset: The training data is colossal, encompassing trillions of tokens sourced from a diverse corpus of filtered web text, books, articles, code repositories, and more. The quality, diversity, and cleanliness of this data are very important. A key concept here is the "token," which for most LLMs is a sub-word unit (using algorithms like Byte-Pair Encoding - BPE), allowing them to handle a vast vocabulary efficiently.
3. Training: The training process [76] is a monumental undertaking in compute. It involves:
    - Pre-training [77]: This is the initial, incredibly resource-intensive phase where the model learns the fundamental statistics of language. Using a self-supervised objective, typically next-token prediction or a masked language modeling objective, the model builds a rich, internal representation of grammar, facts, reasoning patterns, and even stylistic elements present in the data. This phase establishes the model's "world knowledge" but does not make it directly useful or safe for user interaction.
    - Fine-Tuning (Post-Training): The raw, pre-trained model is a powerful next-token predictor but is not yet an helpful, harmless, and honest assistant. This is where fine-tuning [78], particularly a technique called Instruction Tuning, comes in. The model is further trained on a curated dataset of (prompt, response) pairs that demonstrate desired behaviors—following instructions, answering questions helpfully, refusing harmful requests, and adopting a consistent persona.

A raw pre-trained LLM can complete the prompt "The best way to steal a car is..." with technically accurate but harmful information. The goal of alignment is to ensure the model's outputs are helpful, safe, and aligned with human values. The primary technique for this is Reinforcement Learning from Human Feedback (RLHF), popularized by OpenAI.
RLHF [79]is a multi-stage process:

1. Supervised Fine-Tuning (SFT): A dataset of high-quality human demonstrations (ideal responses to prompts) is used to fine-tune the pre-trained model, teaching it a basic style of interaction.
2. Reward Model Training: Human labelers rank multiple outputs from the SFT model for a given prompt from best to worst. This data is used to train a separate reward model that learns to predict which output a human would prefer.
3. Reinforcement Learning (PPO): The SFT model becomes an "agent" that generates responses. The reward model provides a "reward" score for each generated response. Using a reinforcement learning algorithm (like Proximal Policy Optimization - PPO [80]), the LLM's parameters are updated to maximize this reward, thus optimizing its outputs for human preference without needing continuous human input.

The scale of LLMs unlocks capabilities not seen in smaller models:

- In-Context Learning (ICL): An LLM can perform a task from just a few examples provided in the prompt (few-shot) or even from a single example (one-shot) without

any weight updates. This is a form of meta-learning arising from the model's vast pre-training.

- Instruction Following: The ability to understand and execute complex, multi-step instructions presented in natural language.
- Chain-of-Thought (CoT) Reasoning [32]: When prompted to "think step-by-step," LLMs can break down complex problems (e.g., math word problems) into intermediate steps, significantly improving reasoning performance.
- Code Generation and Comprehension: Trained on vast codebases, LLMs like Codex [81] (powering GitHub Copilot) can generate, explain, and debug code across numerous programming languages.

Despite their power, LLMs have fundamental limitations [82] :

- Hallucination: LLMs are proficient generators of statistically plausible text, not arbiters of truth. They can confidently generate false or nonsensical information, a phenomenon known as "hallucination" [83] .
- Lack of True Understanding: They operate on patterns in data, not on a grounded, embodied understanding of the world. Their knowledge is static, limited to their training cut-off date. We can fight this limitation, as well as the above by enriching the prompt using RAG which is the final part of this diploma thesis. We will analyze this concept thoroughly in later chapters.
- Bias and Toxicity: They can reflect and amplify societal biases present in their training data. Despite extensive efforts in alignment, mitigating all bias remains an open challenge.
- Computational Cost: The inference cost of running a single query for a large model is significant, creating barriers to widespread deployment and raising environmental concerns.

Large Language Models represent a paradigm shift in artificial intelligence. By scaling the transformer architecture to unprecedented sizes and aligning it with sophisticated human feedback, we have created tools that are not just processing language but are, in a functional sense, interacting with it in a way that is profoundly useful. They are the foundational engines powering the current revolution in generative AI. However, they are not sentient nor omniscient; they are complex statistical models whose strengths must be leveraged with a clear-eyed understanding of their weaknesses. In the next chapter, we will explore the practical applications and ecosystem that has blossomed around these powerful models.

5.3 Chatbots History

In this chapter, we provide an overview of the historical progression of chatbots, tracing how they have developed over time. We categorize different types of chatbots, with particular attention to the changing approaches to response generation. We also consider both the strengths and limitations of chatbot systems.

In 1950, Alan Turing introduced the Turing Test [84] as a method for assessing machine intelligence. The test involves a human evaluator communicating via text with both a person and a computer, without knowing which is which. If the evaluator cannot reliably tell them apart, the machine is considered to have demonstrated intelligence.

The first widely recognized chatbot, Eliza, was developed by Joseph Weizenbaum in 1966 [85]. Eliza simulated a psychotherapist by using simple pattern-matching rules and predefined responses. Although limited in capability, it created the illusion of meaningful conversation and quickly gained public attention. In 1971, "Artificial Paranoia" was created by Colby and colleagues [86]. Designed to imitate a patient with schizophrenia, it generated responses based on assumptions and emotional triggers derived from user input. While more sophisticated than Eliza, its language comprehension remained restricted.

In 1995, Alice was introduced by Richard Wallace [87]. Unlike its predecessors, Alice used the Artificial Intelligence Markup Language (AIML) and drew upon a knowledge base of over 41,000 templates. Considered one of the most advanced chatbots of its era, Alice could discuss a wide variety of topics and won the Loebner Prize in 2000, 2001, and 2004.

A major shift came with the appearance of intelligent voice assistants such as Apple's Siri, Microsoft's Cortana, Amazon's Alexa, Google Assistant, and IBM's Watson. These systems were capable of processing spoken commands to complete tasks such as sending text messages, organizing schedules, and managing smart home devices.

In recent years, progress in natural language processing (NLP) has enabled the emergence of more advanced conversational agents. A notable example is ChatGPT, released by OpenAI in 2022. Unlike earlier systems, it leverages the transformer architecture [8] (discussed in previous chapters), allowing it to interpret context and perform a wide array of tasks, including programming, creative writing, and customer service.

The development of chatbots for Greek has been slower due to fewer available resources compared to English. However, researchers from Institute of Language and Speech Processing have recently developed "Meltemi" [88] and "Kri-kri" [89] which are fine-tuned versions of the LLMs "Mistral" [90] and "LLama" [91] respectively.

5.4 End-User Access to Large Language Models

The development of large language models (LLMs) is a highly resource-intensive process, requiring significant computational power and financial investment. Creating these models demands substantial infrastructure, as does the fine-tuning process necessary to enable effective conversational interactions with humans, which relies on high-performance GPUs. Consequently, the creation and refinement of sophisticated LLMs are predominantly undertaken by large organizations with the financial capacity to support such computationally demanding tasks. These organizations play a pivotal role in shaping the accessibility and deployment of LLMs, which can be broadly categorized into open source and closed source models, each with distinct approaches to serving the population.

5.4.1 Open Source LLMs

Open source LLMs, such as Meta AI's LLaMA series [91] (for research purposes), Mistral [90], or Falcon [92], are publicly available, with their architectures, weights, and sometimes training datasets accessible to developers, researchers, and enthusiasts. These models are typically hosted on platforms like GitHub or Hugging Face, allowing users with sufficient hardware, such as consumer-grade GPUs or cloud computing services (e.g., AWS, Google Cloud, or Azure), to download and deploy them. This open approach fosters a collaborative ecosystem, enabling individuals, startups, and academic researchers to build customized solutions, such as niche chatbots or industry-specific applications (like ours). The transparency of open source LLMs promotes scrutiny of their behavior, encouraging community-driven innovation. However, running these models requires technical expertise and computational resources, which can be a barrier for some users. Additionally, the cost of hardware or cloud services for inference and fine-tuning may be prohibitive, and ethical concerns about potential misuse necessitate responsible use guidelines.

5.4.2 Closed-source LLMs

In contrast to open source LLMs, closed source LLMs, such as OpenAI's ChatGPT, Anthropic's Claude, or xAI's Grok, are proprietary, with their architectures and training processes kept confidential. These models are primarily offered as services through APIs (e.g., OpenAI's API or xAI's API for Grok) or user-facing applications, such as web interfaces (e.g., grok.com, chatgpt.com) or mobile apps. Access is often provided through subscription plans, pay-per-use models, or free tiers with limited quotas. Especially the creation of APIs enables developers to integrate LLMs into applications, take advantage of this advanced technology and create useful applications. Closed source LLMs prioritize ease of use, requiring minimal technical expertise, and are hosted on robust infrastructure to ensure scalability and reliability. They often incorporate safety mechanisms to mitigate harmful outputs. However, their proprietary nature limits transparency, making it challenging to address biases or understand their inner workings. Access costs, such as subscriptions for premium features, may exclude

some users, and reliance on a single provider can create dependency and raise data privacy concerns.

5.5 Prompt Engineering

Prompt engineering may be defined as the methodological process of formulating precise instructions that enable the effective adaptation and operation of a large language model (LLM) [93]. In order to contextualize this process, it is necessary to examine the typical structure of inputs provided to an LLM in conversational applications. Such inputs are generally organized into a sequence of messages, each of which is assigned a distinct role [94] [95]. The most fundamental message types are the following:

- **System Message:** This message establishes the directions and constraints that the model must adhere to throughout the entire interaction.
- **User Message:** This message represents the contribution or query introduced by the human participant engaging with the model.
- **Assistant Message:** This message corresponds to the response produced by the model in accordance with the preceding instructions and user input.

Prompt Engineering Techniques

1. Zero-Shot Prompting

Zero-shot prompting [29] represents the most fundamental interaction with a large language model (LLM), wherein a task is presented without any prior examples or demonstrations of desired behavior. The model is expected to rely solely on its pre-trained knowledge and inherent reasoning capabilities to interpret the instruction and generate an appropriate response. This technique is predicated on the model's ability to perform internal task recognition and generalization. Its primary advantage lies in its simplicity and efficiency, as it requires minimal effort from the user. However, its efficacy is limited to well-defined and common tasks where the model's training data provides a strong foundation. For complex, multi-step, or highly nuanced tasks, zero-shot prompting is prone to failure, as the model may make incorrect assumptions about the format, depth, or context of the required output, leading to inaccuracies or oversimplifications.

2. One-Shot Prompting

One-shot prompting [30] extends the zero-shot approach by including a single illustrative example within the prompt. This example consists of an input and its corresponding desired output, serving as a concrete demonstration of the task for the model to emulate. The mechanism functions by providing a contextual anchor that guides the model's response generation towards a specific format, style, or structural pattern. This technique is particularly advantageous for tasks that require a defined schema but are not overly complex, effectively bridging the gap between the ambiguity of zero-shot and the resource intensity of few-shot prompting. A significant limitation is that a solitary example may be insufficient to communicate all necessary task constraints or to cover potential edge cases, which can result in inconsistent performance when novel inputs deviate from the provided example.

3. Few-Shot Prompting

Few-shot prompting is a powerful and widely adopted technique that involves providing the model with multiple examples of a task within the prompt context. By presenting several input-output pairs, the model is conditioned to identify and replicate the underlying pattern, reasoning strategy, or stylistic conventions demonstrated. This method was comprehensively demonstrated by Brown et al. [96] in their seminal work, which showed that scaling up model parameters dramatically improved their ability to perform in-context learning from these examples. It is highly effective for complex tasks such as classification with numerous categories, sophisticated text transformation, or applications requiring a specific tonal register. The principal strength of few-shot prompting is its demonstrable ability to significantly enhance model performance and output reliability across a diverse range of challenges. The primary drawback is its increased consumption of context window tokens, which raises computational costs and can become a limiting factor in extended interactions. Furthermore, the selection of optimal examples is non-trivial and often requires an iterative, empirical process.

4. Negative Prompting

Negative prompting [31] is a technique wherein the user explicitly specifies elements, styles, or types of information that must be excluded from the model's output. While prominently featured in text-to-image generation models to prevent unwanted visual artifacts, it is equally critical in text-based LLMs for content control and refinement. The technique operates by instructing the model to avoid certain pathways or associations during its generation process. It is exceptionally well-suited for mitigating biases, preventing the inclusion of sensitive or irrelevant information, and steering the model away from common clichés or generic responses. The effectiveness of negative prompting can be compromised if the instructions conflict with the core objective stated in the primary prompt or if the model's prior biases are exceptionally strong.

5. Iterative Prompting

Iterative prompting is best understood not as a discrete technique but as a meta-strategy or development process. It entails a cyclical methodology of crafting an initial prompt, evaluating the generated output, diagnosing its shortcomings, and refining the prompt accordingly. This process is repeated until the output meets the desired standard of quality and precision. This approach is indispensable for complex, real-world applications where the ideal output is difficult to specify axiomatically in a single attempt. It embodies the pragmatic principle that effective prompt engineering is a collaborative dialogue between the human and the model. The main limitation of this approach is its inherent time consumption and its dependency on the user's analytical skill to correctly identify the reasons for suboptimal outputs.

6. Prompt Chaining

Prompt chaining [97] is an advanced engineering strategy that deconstructs a single, intricate task into a sequence of smaller, more manageable subtasks. The output from each prompt in the sequence is utilized as a foundational input for the subsequent prompt, creating a coherent pipeline. This technique is designed to manage complexity by reducing the cognitive load on the model at any single step, thereby minimizing compound errors and enhancing the overall accuracy and coherence of the final output. It is ideally employed for sophisticated reasoning tasks, multi-document analysis, and the generation of long-form, structured content. The significant disadvantage of prompt chaining is the requirement for external orchestration, typically through programming code, to manage the flow of information between steps, which increases the technical barrier to implementation.

7. Chain-of-Thought (CoT) Prompting

Chain-of-Thought (CoT) prompting is a specialized variant of few-shot prompting specifically designed to elicit explicit, step-by-step reasoning from the model. The provided examples within the prompt include a detailed rationale that leads to the final answer, instructing the model to emulate this explanatory process. This technique is profoundly effective for arithmetic problems, logical deductions, and other complex reasoning tasks where the final answer is less valuable without an understanding of its derivation. This method was formalized by Wei et al. (2022) [98] and has been shown to significantly improve performance on complex reasoning benchmarks. CoT prompting works by unlocking the model's latent ability to perform structured reasoning, a capability often obscured by standard prompting techniques. A notable extension, "Zero-Shot CoT," where the simple instruction "Let's think step by step" is appended to a prompt, has also proven remarkably effective. There are also extensions of this method like Tree of Thoughts prompting [99] and Graph of Thoughts [100]. The main limitation is the

necessity for carefully crafted reasoning examples and the increased consumption of computational resources to generate the extended reasoning text.

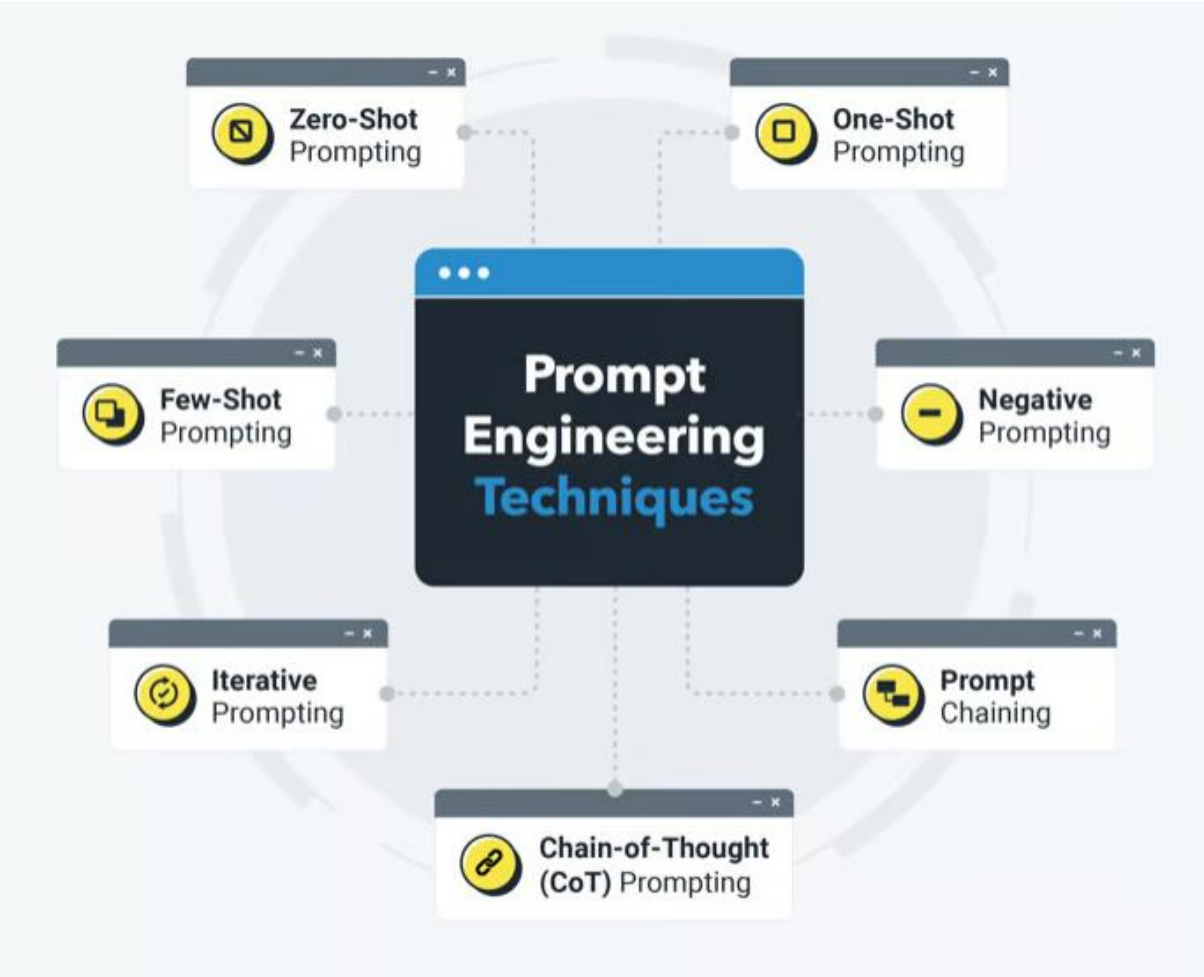

Figure 20 - Prompt Engineering Techniques [101]

5.6. Use Cases of Large Language Models (LLMs)

Introduction

Large Language Models (LLMs) represent a significant advancement in artificial intelligence, leveraging vast datasets and sophisticated neural architectures to process, understand, and generate human-like text. Their applications span multiple domains, transforming traditional workflows by automating complex tasks, enhancing decision-making, and personalizing user experiences. This chapter explores prominent use cases of LLMs across key sectors, drawing on recent research and industry implementations to illustrate their practical impact. While LLMs offer substantial benefits, their deployment also introduces considerations and limitations which will be addressed in the next section.

Use Cases in Education

In the education sector, LLMs are reshaping teaching and learning by providing personalized, scalable support. One primary application is in adaptive learning systems, where LLMs generate customized explanations, examples, and interactive exercises tailored to individual student needs. For instance, they can offer alternative explanations or visual aids to accommodate diverse learning styles, thereby improving accessibility and engagement.
LLMs also facilitate content creation for educators, such as drafting lesson plans, generating assessment questions, or summarizing extensive documents, which saves time and allows teachers to focus on higher-level instruction. In knowledge tracing, LLMs enhance student records by producing auxiliary information, enabling more accurate tracking of learning progress. Additionally, they support scientific writing skills among students by empowering innovation and fostering independent thinking when used appropriately.

Use Cases in Healthcare

Healthcare stands out as a domain where LLMs demonstrate profound transformative potential, particularly in diagnostics, patient engagement, and research. LLMs enhance clinical support by analyzing patient records and electronic health records to automate audits and assist in decision-making. For example, they can simplify documentation tasks, improve patient communication through interactive chatbots, and aid in diagnostics by processing medical literature and symptoms.
In medical education, LLMs serve as tools for generating drafts, summarizing articles, and checking language, thereby supporting scientific writing and learning. They also contribute to research by facilitating clinical trial recruitment and hypothesis generation. Specific use cases include personalized patient care, where LLMs summarize data for better clinical decisions, and drug discovery assistance through NLP tasks.

Despite these advancements, evaluations indicate that while LLMs excel in controlled settings, their clinical readiness requires further validation to ensure reliability and safety. LLMs thus promise to elevate healthcare efficiency while necessitating careful integration.

Use Cases in Business and Finance

In business and finance, LLMs drive efficiency through automation and insightful analytics. Key applications include fraud detection, where models analyze transaction patterns to identify anomalies, and risk assessment by processing regulatory updates in real-time. They also enable personalized financial advisory services, interpreting customer complaints and generating tailored recommendations.

Compliance and investment strategies benefit from LLMs' ability to navigate complex regulations and interpret market information for better decision-making. In broader industry contexts, LLMs support workflow automation, such as generating reports or enhancing customer support, leading to significant productivity gains. For smaller institutions, however, resource intensiveness poses a barrier, highlighting the need for scalable solutions.

Use Cases in Scientific Research

LLMs are increasingly integral to scientific research, aiding in data extraction, hypothesis verification, and literature synthesis. They automate code generation for testing hypotheses and summarize vast datasets, reducing time and errors in manual processes. In literature reviews, LLMs streamline the process by extracting key insights from publications, supporting tasks like sentiment analysis and named entity recognition.

They also generate synthetic data or assist in writing scientific content, such as letters of recommendation or article drafts, while maintaining ethical standards. In specialized domains, LLMs perform NLP tasks with high potential, though their use in generating research outputs raises questions about originality. Despite this, LLMs are not a panacea; their limitations in accuracy necessitate human oversight.

Use Cases in Entertainment and Creative Industries

In entertainment and creative sectors, LLMs augment human creativity by generating content and personalizing experiences. They assist in scriptwriting, music composition, and personalized news feeds, enabling rapid prototyping and innovation. For content creation, LLMs produce marketing materials or blog posts, shifting workflows toward dynamic, AI-enhanced production.

Personalization in streaming platforms uses LLMs to recommend content based on user preferences, improving monetization and engagement. In pre- and post-production, they handle tasks like generating visuals or editing, though copyright concerns with training data persist. Studies show LLMs can enhance divergent thinking but may homogenize ideas if over-relied

upon. LLMs thus enhance rather than replace creativity, with implications for design and media.

Use Cases in Customer Service and Marketing

Customer service and marketing leverage LLMs for efficient, personalized interactions. In support, LLMs power chatbots that handle queries, extract context from conversations, and provide instant responses, achieving up to 10x productivity gains. They analyze sentiment in customer feedback for nuanced insights.
In marketing, LLMs generate tailored content like emails or website copy, segment customers based on data, and translate materials for global reach. Use cases include automating ticketing and FAQ handling in B2B contexts. Adapting LLMs with interaction data further boosts engagement and satisfaction.

5.7 Critical Limitations of Large Language Models

The remarkable capabilities of Large Language Models (LLMs), as detailed in the previous chapter, have caused a paradigm shift in artificial intelligence and natural language processing. Their proficiency in generating coherent, contextually relevant, and often insightful text has led to their deployment across a vast array of domains, from creative writing to technical support. However, it is a critical fallacy to equate this proficiency with genuine understanding, reasoning, or intelligence. The performance of LLMs is fundamentally constrained by a set of inherent limitations rooted in their architecture, training data, and operational mechanics.
This chapter provides a critical examination of these limitations. A thorough understanding of these constraints is not merely an academic exercise but a prerequisite for the responsible development, deployment, and interpretation of LLM outputs in real-world applications. In the below paragraphs we will explore issues of factual accuracy, reasoning capabilities, inherent biases, computational demands, and the elusive nature of true understanding. Some of the limitations highlighted in this chapter provide the motivation for Retrieval-Augmented Generation, discussed in the following chapter. A critical examination of these limitations facilitates a deeper understanding of the practical significance of this technique.

The Illusion of Knowledge: Hallucinations and Factual Inaccuracy

Perhaps the most significant and perilous limitation of LLMs is their tendency to generate plausible yet entirely fabricated or incorrect information, a phenomenon commonly termed "hallucination" [102]. Unlike a database or a search engine that retrieves stored records, an

LLM generates text by predicting the most probable next token based on its training data and the provided prompt.
This statistical nature means LLMs lack a certainly correct model of the world. They do not "know" facts but instead, they have learned statistical correlations between words and concepts. Consequently, they can:

- Invent citations and sources: Generating references to non-existent academic papers or books.
- Provide incorrect data: Stating inaccurate statistics, historical dates, or scientific facts with high confidence.
- Create fictitious narratives: Generating detailed but false descriptions of events or biographies.

This limitation poses a severe risk in high-stakes domains like healthcare, law, and journalism, where factual accuracy is paramount. Mitigation strategies like Retrieval-Augmented Generation (RAG) [33], discussed in the next chapter, attempt to ground LLMs in external, verifiable knowledge sources, but the core tendency to hallucinate remains an unsolved fundamental problem.

The Absence of Robust Reasoning and Common Sense

While LLMs can solve certain logical puzzles and perform reasoning tasks within their training distribution, their reasoning is often unstable and superficial. They excel at mimicking reasoning patterns found in their data but struggle with novel problems requiring genuine abstraction, multi-step planning, or the application of commonsense knowledge not explicitly stated in text.

Key failures include:

- Logical Inconsistencies: Inability to maintain consistency over long passages of text, often contradicting themselves within a single response.
- Difficulty with Counterfactuals: Struggling to reason about scenarios that contradict established facts or their training data.
- Poor Arithmetic and Symbolic Reasoning: Despite being trained on vast amounts of data, their performance on basic arithmetic or symbolic manipulation is unreliable without specific tool augmentation (e.g., a calculator).

This suggests that LLMs are not built upon a foundation of logical rules or causal models but are instead engaging in a form of "stochastic parroting" [103] recombining seen patterns without deep comprehension.

Inherent Biases and the Reflection of Training Data Imperfections

LLMs are not objective or neutral entities. They are mirrors reflecting the entirety of their training data: the good, the bad, and the ugly. The internet-based corpora used for training contain pervasive societal biases, stereotypes, and offensive content. Consequently, LLMs inevitably learn, amplify, and can perpetuate these biases [104].
These biases manifest as:

- Sociodemographic Bias: Generating text that associates certain professions, traits, or behaviors with specific genders, ethnicities, or nationalities.
- Representational Bias: Over- or under-representing certain viewpoints, cultures, or languages based on their prevalence in the training data (typically a Western, English-language bias).
- Toxic and Harmful Outputs: Generating abusive, hateful, or otherwise harmful language, even when prompted with seemingly neutral inputs.

Debiasing LLMs is an active area of research but remains profoundly challenging. Techniques like reinforcement learning from human feedback (RLHF) can suppress the most overtly harmful outputs but often fail to address more subtle, ingrained biases. The model's behavior is ultimately a direct function of its data, and perfect data does not exist.

Static Knowledge Limitation

The knowledge of an LLM is frozen in time at the moment of its pre-training. A model trained on data up to late 2023 has no innate knowledge of events, discoveries, or cultural shifts occurring thereafter. This presents a major limitation for applications requiring up-to-date information (e.g., news analysis, scientific research, current events).
Strategies to overcome this include:

1. Periodic Retraining: Costly, computationally intensive, and environmentally unsustainable.
2. Fine-tuning on new data: A more efficient alternative but can introduce catastrophic forgetting, where the model loses performance on previously learned tasks. Also, this technique is very resource-intensive which is a key factor to considerate.
3. Retrieval-Augmented Generation (RAG) (see next chapter): The most promising approach, where the LLM is paired with an external, updatable knowledge base, allowing it to access current information without altering its core parameters.

This limitation underscores that an LLM is not a living, learning system but a static snapshot of a point in time.

Computational and Environmental Costs
The development and operation of state-of-the-art LLMs incur staggering computational, financial, and environmental costs. Training a single model like GPT-3 was estimated to consume several hundred megawatt-hours of electricity and generate a carbon footprint

equivalent to hundreds of tons of CO_2 [105]. Furthermore, inference, generating responses for millions of users, requires continuous, massive GPU power.
This raises serious concerns regarding:

- Environmental Sustainability: The carbon footprint of widespread LLM use.
- Economic Barrier to Entry: Concentrating the power to develop frontier models in the hands of a few well-funded tech corporations, potentially stifling innovation and academic research.
- Resource Allocation: Ethical questions about the societal value of allocating vast energy resources to these systems.

The Opacity of Black-Box Models and the Explainability Problem

LLMs are considered "black boxes". With hundreds of billions of parameters and complex, non-linear interactions, it is virtually impossible to trace a specific model output back to a specific piece of training data or to understand the internal "reasoning" process that led to it. This lack of explainability and interpretability is a critical barrier for deployment in regulated industries like finance (e.g., for loan denials) or medicine (e.g., for diagnoses), where decisions must be justified and auditable.
The field of Explainable AI is working on techniques to shed light on model behavior, but providing clear, causal explanations for LLM outputs remains a fundamental challenge [106].

The Fundamental Lack of Understanding and Consciousness

At their core, LLMs are sophisticated autocomplete systems. They manipulate symbols without any grounding in real-world experience, sensory input, or embodied cognition. They have no desires, beliefs, intentions, or consciousness. The text they generate is a statistical construct, devoid of genuine understanding or semantic meaning.
This philosophical limitation, often discussed in the context of the "Chinese Room" argument [107], is crucial for tempering expectations. LLMs simulate understanding but do not instantiate it. They are powerful tools for processing and generating human language, but they are not sentient beings and should not be anthropomorphized.

Conclusion

This chapter has delineated the critical limitations that restrict the capabilities of Large Language Models. From their tendency to hallucinate and lack of robust reasoning to their inherent biases, static knowledge, immense costs, and opaque nature, these constraints are not mere bugs to be fixed but inherent properties of their current architecture and training paradigm.
Acknowledging these limitations is not to diminish the transformative potential of LLMs but to contextualize it. It serves as a vital corrective to the hype that often surrounds this technology and provides a framework for responsible development. In the following chapter we will

analyze how we can avoid these limitations so we can create applications that are very useful in the real world.

6 Retrieval-Augmented Generation (RAG)

This chapter will provide a comprehensive examination of Retrieval-Augmented Generation. The subsequent sections will detail the core architectural components, including the retriever and generator models. We will then explore various methodologies, from naive RAG implementations to advanced techniques. Following this, the chapter will discuss key metrics and frameworks for evaluating the performance of RAG systems. To connect these concepts to a practical application, we will simultaneously analyze their implementation within the chatbot developed for this thesis. We will begin by making an introduction to RAG and restating the chatbot's requirements to provide a clear reasoning for the methodologies chosen in this chapter.

6.1 Introduction to RAG and Chapter Outline

Retrieval-Augmented Generation (RAG) represents a paradigm shift in how large-scale language models interact with information, effectively bridging the gap between their static, internalized knowledge and the dynamic, vast amount of external data repositories. This chapter introduces the fundamental concepts of RAG, positioning it as a pivotal architectural pattern designed to enhance the factuality, timeliness, and verifiability of generative AI systems.

The Limitations of Traditional Large Language Models

As established in the preceding chapter, while Large Language Models (LLMs) exhibit profound capabilities in natural language understanding and generation, they are constrained by several inherent limitations. These challenges, which stem directly from their design as self-contained, parametric models, create a clear need for architectures like RAG. The primary limitations can be summarized as follows:

- **Knowledge Cutoff:** An LLM's knowledge is frozen at the point its training concludes. It has no intrinsic mechanism to access information or events that have occurred post-training, rendering its outputs potentially outdated.
- **Hallucinations:** LLMs can generate text that is plausible and grammatically correct but factually inaccurate or entirely nonsensical. This phenomenon, often termed 'hallucination', arises when the model lacks the necessary information in its parametric weights and instead generates a statistically likely but unsubstantiated response.

- **Lack of Domain-Specificity:** While trained on vast general corpora, standard LLMs lack deep expertise in specialized, proprietary, or niche domains. Fine-tuning is a possible remedy, but it is computationally expensive, time-consuming, and must be repeated to incorporate new information.
- **Opacity and Lack of Citations:** The "black box" nature of LLMs makes it exceedingly difficult to trace a generated statement back to its source material. This opacity is a significant barrier to trust and verification, particularly in academic, medical, and enterprise contexts where source attribution is critical.

Introducing RAG as a Solution

Retrieval-Augmented Generation (RAG) [108] is a hybrid AI framework that enhances the capabilities of a generative LLM by dynamically retrieving relevant information from an external knowledge base before generating a response. At its core, RAG synergizes two distinct types of knowledge:

1. **Parametric Knowledge:** This is the knowledge implicitly encoded within the LLM's neural network parameters during its pre-training phase. It encompasses broad language patterns, general facts, and reasoning abilities.
2. **Non-Parametric Knowledge:** This refers to explicit information stored in an external data source, such as a vector database, a document corpus, or a knowledge graph.

Conceptually, the RAG process functions as an "open-book exam" for the LLM. When presented with a prompt or query, the system does not immediately rely on its internal knowledge alone. Instead, it first employs a retriever component to search the external knowledge base for documents or text snippets that are semantically relevant to the input query. These retrieved passages are then concatenated with the original prompt and fed as augmented context to the **generator** (the LLM). Armed with this timely and specific information, the LLM can formulate a response that is not only contextually appropriate but also grounded in verifiable, external facts.

This architecture directly mitigates the core limitations of traditional LLMs. It provides a mechanism for incorporating up-to-date information, reduces the likelihood of hallucinations by grounding responses in retrieved evidence, allows for deep domain specialization without costly retraining, and enables source citation by pointing to the specific documents used for generation.

Below is an image describing the whole process of Retrieval Augmented Generation. It depicts all the steps that will be analyzed in this chapter and provides a good visual understanding of the RAG pipeline.

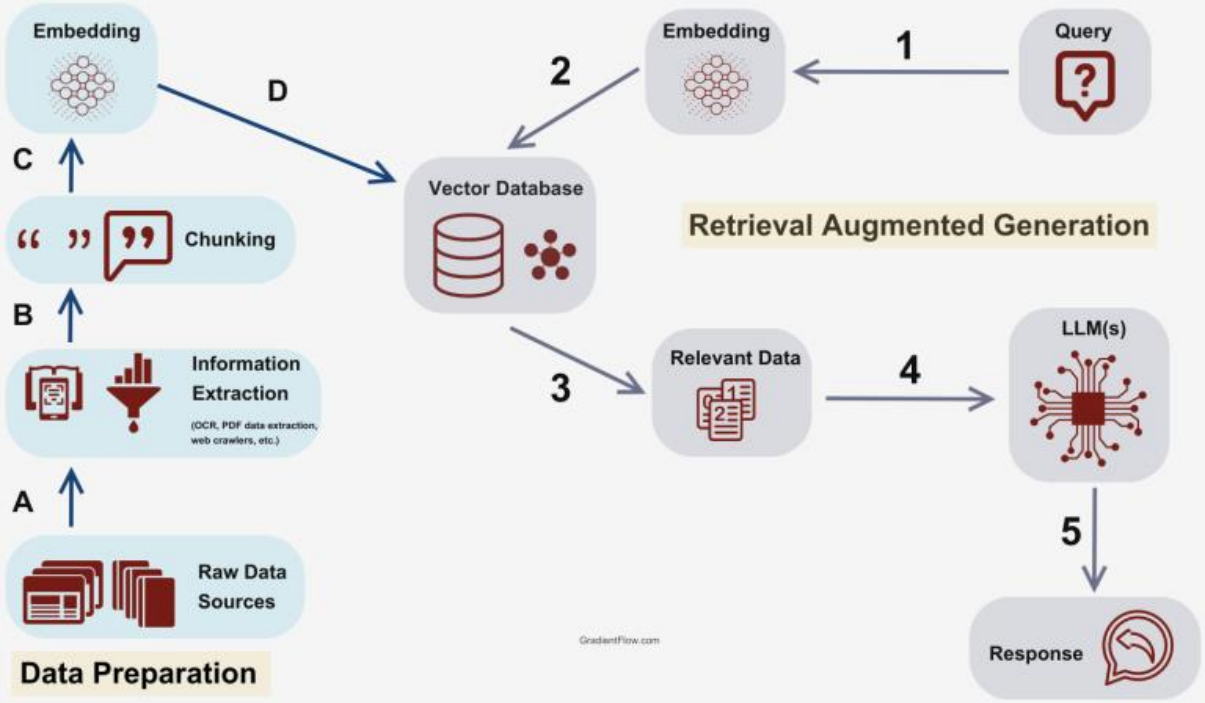

Figure 21 - Process of Retrieval Augmented Generation. [109]

6.2 Chatbot Application requirements-goals

Before proceeding to the actual techniques described in this chapter let's shortly restate the application that we have to build for this thesis. The project details the design, implementation, and evaluation of a Retrieval-Augmented Generation (RAG) system built to assist users of the "ATHENA" application. "ATHENA" is a vital tool used by bank employees to create diagrams for financial data representation. A common pain point for these employees is the need to find answers to specific questions about the application's functionality. This often requires them to stop their work and go through a lengthy user guide, a process that is both inefficient and frustrating. To solve this problem, we are developing a RAG chatbot that will serve as an instant knowledge base. By allowing employees to ask questions in natural language and get immediate responses, the chatbot is expected to streamline their workflow, reduce downtime, and significantly increase their day-to-day productivity.

The user guide that will be used for retrieval is not plain text but has some quite complex structures like tables and images. Therefore, advanced techniques need to be used in order to successfully save and retrieve this information. We will discuss all of those techniques in the following sections.

6.3 Overview of the LangChain Ecosystem

The LangChain ecosystem [34] has quickly grown into one of the most important toolkits for building applications powered by large language models (LLMs). At its core, LangChain provides a framework that helps developers connect LLMs with external data sources, APIs, and reasoning capabilities, making these models more practical and versatile in real-world scenarios.

The ecosystem can be thought of as a collection of components and integrations that work together to unlock the potential of LLMs. The central library, LangChain, serves as the foundation. It offers abstractions for prompts, memory, chains, and agents—concepts that help developers manage how an LLM processes input, maintains context, executes tasks, and interacts with external tools.

Beyond the core library, LangChain is supported by a growing community of integrations. These allow applications to seamlessly connect with databases, vector stores, APIs, and cloud services. For example, LangChain makes it possible to pair LLMs with vector databases for semantic search (as seen in the following sections), or to link them with APIs so that the model can fetch up-to-date information or execute actions on behalf of the user. This modular design ensures flexibility: developers can swap components in and out depending on the needs of their project.

Another key aspect of the ecosystem is its developer infrastructure. LangSmith, for instance, enables developers to test, monitor, and debug their LangChain applications, ensuring reliability and scalability. Similarly, LangServe provides deployment tools so that chains and agents can be exposed as APIs and integrated into larger systems without friction. Together, these tools make it easier not just to prototype, but also to move applications into production.

The ecosystem is also characterized by its strong community and open-source spirit. Contributions from developers worldwide continue to expand the number of supported integrations and enrich the documentation and examples available. This collaborative energy has helped LangChain become more than just a library—it has grown into a full-fledged ecosystem that lowers the barriers for anyone who wants to build with LLMs.

In short, the LangChain ecosystem is about providing the structure, tools, and integrations needed to transform raw LLM capabilities into useful, scalable, and reliable applications. It sits at the intersection of language models, external data, and real-world workflows, making it a cornerstone for modern AI development. We used Langchain's framework for the majority of our application as it can be observed in the following chapters and the code repository.

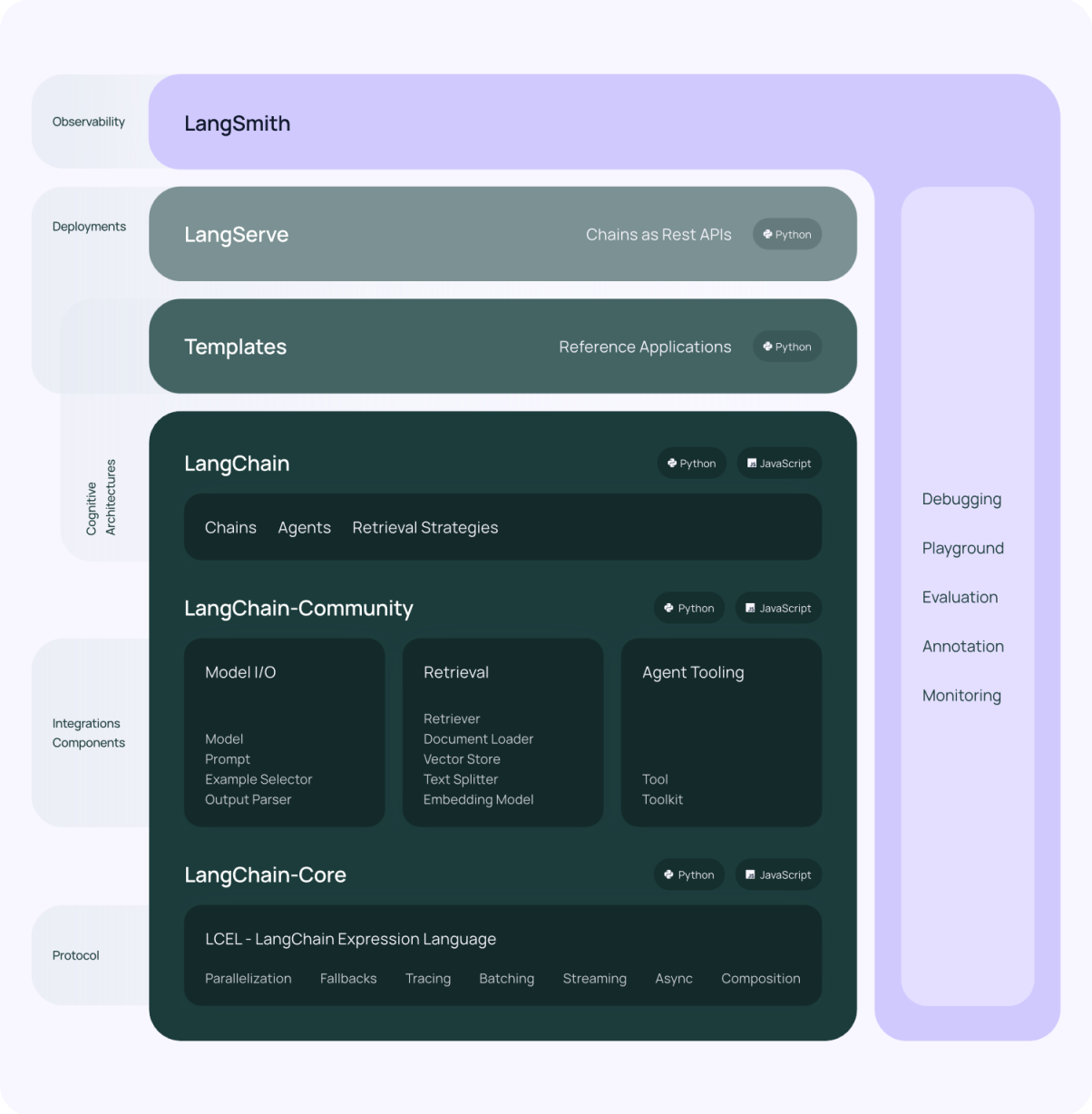

Figure 22 - Visual Representation of the LangChain Ecosystem's Components [110]

6.4 The Indexing Pipeline (Data Preparation)

The indexing pipeline is the foundational offline process in a Retrieval-Augmented Generation (RAG) system, responsible for transforming raw data into a structured, searchable knowledge base. This pipeline ensures that external information is readily accessible for retrieval during the generation phase. It involves several critical steps: loading and preprocessing raw documents, chunking them into manageable pieces, generating embeddings, and storing these embeddings in a vector database for efficient similarity searches. Each step is designed to optimize the quality and accessibility of the knowledge base, enabling the RAG system to deliver accurate and contextually relevant responses.

6.4.1 Data Loading & Preprocessing

The indexing pipeline begins with sourcing and preparing raw documents to create a robust knowledge base for the Retrieval-Augmented Generation (RAG) system. Data loading involves collecting documents from diverse sources, such as PDFs, HTML pages, text files, or databases, depending on the application's requirements. Common sources include web pages, internal knowledge bases, academic papers, or user-uploaded files. In our case, these documents were two PDFs containing information about how to use the "ATHENA" application.

Once collected, the documents undergo preprocessing to ensure they are clean and usable. This step typically includes:

- **Text Extraction**: Extracting raw text from various formats. For PDFs, tools like PyPDF, PDFMiner or PyMuPDF4LLM can be used to parse text, while HTML documents may require libraries like BeautifulSoup to strip tags and extract content. For structured data (e.g., JSON or CSV), relevant fields are extracted and normalized.

  For our implementation we used PyMuPDF4LLM through the Langchain ecosystem. The reason was that this framework has built-in functionality to parse the tables and images. More specifically, we used the "table_strategy" argument and the "loader" argument which provides several ways to parse images. We implemented two ways of parsing images as shown in the images below. The first way was the most efficient, because it used a multimodal LLM ('gpt-4o-mini') and, through a carefully structured prompt, it output on the final document the thorough description of each image, in the positions that the respective images were originally placed. As a result, the LLM that used this text later in the RAG pipeline had highly accurate information in regards to the content of the image. The second image parsing method was through OCR and it was not as efficient as there was loss of information in the

produced text. We used this method in the second PDF for testing purposes and the results of the image extraction were significantly worse than the first method.

```python
# loading PDF, DOCX and TXT files as LangChain Documents
def load_document_OPENAI(file):
    import os
    name, extension = os.path.splitext(file)

    if extension == '.pdf':
        from langchain_pymupdf4llm import PyMuPDF4LLMLoader
        from langchain_community.document_loaders.parsers import LLMImageBlobParser
        from langchain_openai import ChatOpenAI
        print(f'Loading {file}')
        loader = PyMuPDF4LLMLoader(
            file,
            mode="page",
            extract_images=True,
            images_parser=LLMImageBlobParser(
                prompt= '''Είσαι ένας βοηθός με αποστολή να συνοψίζεις εικόνες για σκοπούς ανάκτησης. Απαντάς στην ελληνική γλώσσα.
                Σύνοψη: Κατάγραψε με ακρίβεια και συντομία τα δεδομένα της εικόνας, ώστε να είναι εύκολη η ανάκτησή τους.
                Περιεχόμενο: Συμπερίλαβε όλα τα εμφανιζόμενα στοιχεία χωρίς σχόλια, αξιολογήσεις ή ερμηνείες.
                Διάταξη: Διατήρησε τη δομή του πίνακα ή των δεδομένων όσο γίνεται.
                Μορφή: Η απάντηση να είναι σε markdown χωρίς να περιέχει επεξηγηματικό κείμενο ή περιγράμματα κώδικα όπως ```.''',
                model = ChatOpenAI(model='gpt-4o-mini', temperature=1)
            ),
            table_strategy="lines",
        )
    elif extension == '.docx':
        from langchain.document_loaders import Docx2txtLoader
        print(f'Loading {file}')
        loader = Docx2txtLoader(file)
    elif extension == '.txt':
        from langchain.document_loaders import TextLoader
        loader = TextLoader(file)
    else:
        print('Document format is not supported!')
        return None

    data = loader.load()
    return data
```

Figure 23 - Implementation of the first method for document parsing where the images were parsed through an LLM.

```python
def load_document_OCR(file):
    import os
    name, extension = os.path.splitext(file)

    if extension == '.pdf':
        from langchain_pymupdf4llm import PyMuPDF4LLMLoader
        from langchain_community.document_loaders.parsers import TesseractBlobParser
        print(f'Loading {file}')
        loader = PyMuPDF4LLMLoader(
            file,
            mode="page",
            extract_images=True,
            images_parser=TesseractBlobParser(),
            table_strategy="lines",
        )
    elif extension == '.docx':
        from langchain.document_loaders import Docx2txtLoader
        print(f'Loading {file}')
        loader = Docx2txtLoader(file)
    elif extension == '.txt':
        from langchain.document_loaders import TextLoader
        loader = TextLoader(file)
    else:
        print('Document format is not supported!')
        return None

    data = loader.load()
    return data
```

Figure 24 - Implementation of the first method for document parsing where the images were passed through OCR.

- **Cleaning**: Removing noise such as special characters, extra whitespace, or formatting artifacts (e.g., page numbers, headers/footers in PDFs). This ensures the text is coherent and free of irrelevant elements. In this part, PyMuPDF4LLM was helpful because it automatically parsed all the pdf in markdown format, ideal form for LLM input.
- **Normalization**: Ensuring consistency across documents, such as standardizing date formats, expanding abbreviations, or translating languages if needed.
- **Validation**: Checking for data quality issues, like incomplete documents or encoding errors, and filtering out low-quality or irrelevant content.

The goal of preprocessing is to produce clean, standardized text that can be effectively chunked and embedded in subsequent steps. The quality of preprocessing directly impacts the system's ability to retrieve relevant information, as noisy or poorly formatted data can lead to suboptimal embeddings and retrieval performance.

At the end of this stage, we obtained a single LangChain document that combined the contents of the first PDF with those of the second. The images and tables were parsed using the methods described earlier.

6.4.2 Document Chunking

Since language models have a limited context window (the amount of text they can process at once), feeding them entire long documents, like the one we described in the previous section is inefficient and often impossible. Chunking is the strategy of breaking down these large documents into smaller, semantically coherent pieces. The choice of chunking strategy is a critical design decision that directly impacts retrieval quality.

- **Fixed-Size Chunking**: This is the simplest method. The text is split into chunks of a fixed number of characters or tokens (e.g., 500 tokens) with a potential overlap between consecutive chunks.
  This method is fast and easy to implement, but it can unnaturally split sentences or separate related ideas, leading to a loss of semantic context.

- **Recursive Chunking**: A more sophisticated approach that attempts to preserve semantic boundaries. It splits the text recursively using a predefined list of separators, such as paragraphs (\n\n), sentences (.), and then spaces ( ). It prioritizes keeping related content together by first trying to split at the largest logical unit (a paragraph) before moving to smaller ones.

  This method is generally better at maintaining context than fixed-size chunking, though it can be slightly more complex to implement.

  This was the method we used (as shown in the image below) and we tested different chunk sizes and chunk overlap. Afterwards, we evaluated these different combinations to decide which one is the best. The evaluation process will be described in detail later in this diploma thesis.

```python
def chunk_data(data, chunk_size=100, chunk_overlap=0):
    from langchain.text_splitter import RecursiveCharacterTextSplitter
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
    chunks = text_splitter.split_documents(data)
    return chunks
```

Figure 25 - Function to implement recursive chunking using Langchain's RecursiveCharacterTextSplitter.

- **Semantic Chunking**: This is the most advanced strategy. Instead of relying on character counts or separators, it uses the semantic meaning of the text to create chunks. This can be done by looking for changes in topics or by using a language model to determine the most logical break points between ideas.
  This method produces the most contextually coherent chunks, leading to superior retrieval performance. However, it is computationally more expensive and complex, often requiring an embedding model even during the chunking process itself.

6.4.3 Embeddings Generation and Vector Database storage

Once documents are preprocessed and chunked into manageable units, the next step in the indexing pipeline is embedding generation. An embedding, as discussed earlier in this thesis, is a numerical vector representation of text, designed so that semantically similar pieces of text are mapped close to each other in a high-dimensional space. This transformation allows a retrieval-augmented generation (RAG) system to search and reason over knowledge not by simple keyword matching, but by semantic similarity.
Embeddings serve as the backbone of modern information retrieval systems, particularly in RAG architectures. By converting text into dense vector representations, embeddings enable the system to capture the semantic essence of the content. For instance, words like "car" and "automobile" or phrases like "machine learning" and "artificial intelligence" may be positioned closer together in the embedding space due to their contextual similarity, even if they share no common keywords. This capability is critical for RAG systems, which rely on retrieving relevant documents or text chunks to generate accurate and contextually appropriate responses.
The process begins with selecting an appropriate embedding model. Common choices include models such as text-embedding-ada-002, BAAI/bge-m3, text-embedding-3-large or more specialized models like Sentence-BERT, which are fine-tuned to produce high-quality embeddings for sentences or paragraphs. These models are trained on large corpora to understand linguistic nuances, ensuring that the resulting vectors reflect not just syntactic structure but also deeper semantic relationship. For our application, we used text-embedding-3-large, as it was observed to have high accuracy for the Greek language.
To generate embeddings, each chunk of preprocessed text is passed through the chosen embedding model. The model processes the text and outputs a fixed-length vector. These vectors are then stored in a vector database, such as FAISS, ChromaDB and Pinecone, which is optimized for fast similarity searches using techniques like cosine similarity or Euclidean distance, discussed earlier in this paper. In our implementation, we used Pinecone for quick API retrieval and Chromadb for local testing.
The quality of embeddings directly impacts the performance of the RAG system. Poorly generated embeddings may fail to capture nuanced relationships between text chunks, leading to irrelevant retrievals. This is the reason the correct selection of the model is vital for a well-functioning RAG application.

```python
def insert_or_fetch_embeddings(index_name, chunks):
    from langchain.embeddings import AzureOpenAIEmbeddings
    from pinecone import Pinecone
    from pinecone import ServerlessSpec
    from langchain_pinecone import PineconeVectorStore


    # pc = pinecone.Pinecone()
    pc = Pinecone()


    embeddings = AzureOpenAIEmbeddings(
    openai_api_version="2024-02-01",
    azure_deployment="text-embedding-3-large",
    model="text-embedding-3-large",
    chunk_size=2800
    )




    # loading from existing index
    existing_index_names = [index["name"] for index in pc.list_indexes()]
    if index_name in existing_index_names:
        print(f'Index {index_name} already exists. Loading embeddings ... ', end='')
        vector_store = PineconeVectorStore.from_existing_index(index_name=index_name, embedding=embeddings)
        print('Ok')
    else:
        # creating the index and embedding the chunks into the index
        print(f'Creating index {index_name} and embeddings ...', end='')

        # creating a new index
        pc.create_index(
            name=index_name,
            dimension=3072,
            metric='cosine',
            spec=ServerlessSpec(
                cloud="aws",
                region="us-east-1"
        )
        )
        vector_store = PineconeVectorStore(index_name=index_name, embedding=embeddings)
        vector_store.add_documents(documents=chunks)
        print('Ok')

    return vector_store
```

Figure 26 - Function used for creating embeddings with Pinecone.

```python
def create_embeddings_chroma(chunks, persist_directory='./chroma_db'):
    from langchain.vectorstores import Chroma
    from langchain.embeddings import AzureOpenAIEmbeddings
    embeddings = AzureOpenAIEmbeddings(
    openai_api_version="2024-02-01",
    azure_deployment="text-embedding-3-large",
    model="text-embedding-3-large",
    chunk_size=2000
    )

    vector_store = Chroma.from_documents(chunks, embeddings, persist_directory=persist_directory)

    return vector_store
```

Figure 27 - Function used for creating embeddings with ChromaDB.

6.5 The Retrieval and Generation Pipeline (Real-time Querying)

The retrieval and generation pipeline represents the online, real-time process that is initiated when a user submits a query to the system. This phase leverages the indexed data prepared during the offline ingestion process to generate a factually grounded and contextually relevant response. The pipeline consists of four sequential stages: user query embedding, semantic search and retrieval, context augmentation, and augmented generation. This sequence ensures that the Large Language Model's (LLM) response is synthesized from verified information retrieved from the knowledge base, rather than relying solely on its internal, pre-trained parameters [111].

6.5.1 User Query Embedding

The first step in the real-time pipeline is the transformation of the user's natural language query into a high-dimensional numerical vector. This process, as described in the previous section, is critical for enabling semantic comprehension by the system.

To achieve this, the user's input string is processed by the same pre-trained embedding model that was used during the offline data ingestion phase (The embedding model in our case was text-embedding-3-large, as mentioned in the previous section). Maintaining consistency in the embedding model is critical. Using the same model ensures that the query and the document chunks reside within the same vector space. This shared semantic space allows for meaningful comparisons, where vectors that are closer together represent concepts that are more

semantically similar. The output of this stage is a query vector, which encapsulates the semantic intent of the user's question, ready for the subsequent retrieval phase.

6.5.2 Semantic Search & Retrieval

With the query transformed into a vector, the system then performs a semantic search against the indexed vector database. The objective is to identify and retrieve the document chunks that are most relevant to the user's query. This is not a keyword-based search but a search for semantic similarity.

The search is executed by comparing the user's query vector against all the document chunk vectors stored in the database. This comparison is quantified using a similarity metric. Two of the most common metrics employed for this purpose are Cosine Similarity and Dot Product, as discussed previously on this paper.

The system calculates the similarity score between the query vector and every document chunk vector in the database. It then ranks the chunks by their scores in descending order and retrieves the top-k most relevant results, where 'k' is a pre-configured parameter. The choice of 'k' represents a trade-off between providing sufficient context and overloading the LLM with too much, potentially irrelevant, information. In order to decide on which ‘k’ value was the best, we used evaluation metrics discussed later. Similar to the Vector Storage, we used ChromaDB and Pinecone for similarity search. Since we used Pinecone through an API, the retrieval of the relevant chunks was much faster with this method.

6.5.3 Context Augmentation

Once the top-k document chunks are retrieved, they form the context that will be used to ground the LLM's response. These seperate pieces of text are not sent to the LLM in their raw form. Instead, they are systematically formatted and integrated into a prompt template in a process known as context augmentation.

The prompt template is a pre-defined structure that combines the retrieved context with the original user query. It typically includes explicit instructions for the LLM, guiding it on how to behave and how to use the provided information. An example of a simplified prompt template that we actually used in the testing of our application is this:

'''
Χρησιμοποίησε τα παρακάτω αποσπάσματα περιεχομένου για να απαντήσεις στην ερώτηση του χρήστη.
Αν δεν βρεις την απάντηση στο παρεχόμενο περιεχόμενο, απλώς απάντησε "Δεν γνωρίζω."
Περιεχόμενο: {context}
'''

This structured prompt is crucial. It clearly outlines the scope of information the LLM is permitted to use, explicitly instructing it to synthesize an answer based only on the retrieved factual data. This step is the cornerstone of the Retrieval-Augmented Generation (RAG) architecture, directly linking the final output to the source knowledge base.

6.5.4. Augmented Generation

The final stage of the pipeline is augmented generation. The complete, augmented prompt, containing the instructions, the retrieved context, and the user's query, is sent as a single input to the generator LLM. In our case, this LLM was gpt-4 from OpenAI’s API.

Upon receiving this prompt, the LLM's task is not to recall information from its vast pre-trained knowledge but to perform a task of synthesis and summarization based exclusively on the provided context. The model reads and comprehends the supplied text chunks and formulates a coherent, natural language answer that directly addresses the user's question.

This method of grounding the LLM's response in retrieved, factual data is the primary mechanism for mitigating the risk of "hallucinations", the generation of plausible but incorrect or fabricated information as discussed previously. By constraining the LLM to a specific set of trusted documents, the system ensures that the generated answer is verifiable, accurate, and directly traceable to the source knowledge base. The final output presented to the user is this synthesized, context-aware, and factually grounded response.

Thanks to the versatility and compact development of LangChain frameworks, in our final solution, we implemented the steps of retrieval, context augmentation and augmented generation with one simple function as shown in the figure below.

```python
def ask_and_get_answer(vector_store, q, k=2):
    from langchain.chains import RetrievalQA
    from langchain_openai import AzureChatOpenAI

    llm = AzureChatOpenAI(
        azure_deployment="gpt-4",
        api_version="2025-01-01-preview",
    )

    retriever = vector_store.as_retriever(search_type='similarity', search_kwargs={'k': k})

    # Retrieve the documents separately
    docs = retriever.get_relevant_documents(q)

    # Optional: print or inspect the retrieved chunks
    print("Retrieved Chunks:")
    for i, doc in enumerate(docs):
        print(f"\nChunk {i + 1}:\n{doc.page_content}")

    # Then build the chain
    chain = RetrievalQA.from_chain_type(llm=llm, chain_type="stuff", retriever=retriever)

    answer = chain.invoke(q)
    return answer, docs  # return both the answer and the chunks
```

Figure 28 - Function for retrieval, context augmentation and augmented generation with LangChain.

6.6 Colpali

Introduction

In the field of information retrieval, traditional systems for processing visually rich documents (such as PDFs containing tables, figures, and layouts) rely on complex pipelines involving PDF parsing, optical character recognition (OCR), layout detection, and text chunking. These methods were analyzed in the previous sections of this thesis. These steps are often error-prone and fail to fully utilize visual elements, leading to suboptimal performance in tasks like Retrieval Augmented Generation. The ColPali method addresses this by leveraging Vision Language Models (VLMs) to directly embed document images, bypassing preprocessing and enabling end-to-end trainable retrieval that incorporates both textual and visual cues. This approach is motivated by evidence that improving document ingestion yields greater gains than refining text embeddings alone. ColPali introduces a benchmark called ViDoRe for evaluating such systems and demonstrates superior results on visually intensive tasks.

Architecture

ColPali is built upon the PaliGemma-3B VLM, which combines a SigLIP vision encoder with a Gemma-2B language model through a multimodal projection layer. The process begins by rendering document pages as images, which are divided into patches (typically 1024 for

PaliGemma). These patches are encoded by the vision model and treated as tokens, prepended to a textual prompt before being fed into the language model.
A key innovation is the addition of a projection layer that maps the language model's output embeddings (for both text tokens and image patches) to a lower-dimensional space (D=128). This creates multi-vector representations per page, akin to the ColBERT model, where each vector corresponds to a token or patch. This design allows fine-grained matching between queries and documents in a shared embedding space, exploiting the VLM's pre-trained alignment between vision and language. Variants explored include reducing patches (e.g., to 512) or adapting other VLMs like Idefics2-8B or Qwen2-VL-2B, but the core PaliGemma-based setup balances efficiency and performance.
Optional token pooling, such as hierarchical mean pooling, can reduce redundant embeddings (e.g., from blank areas) by up to 66.7% while retaining nearly full effectiveness, further optimizing storage and computation.

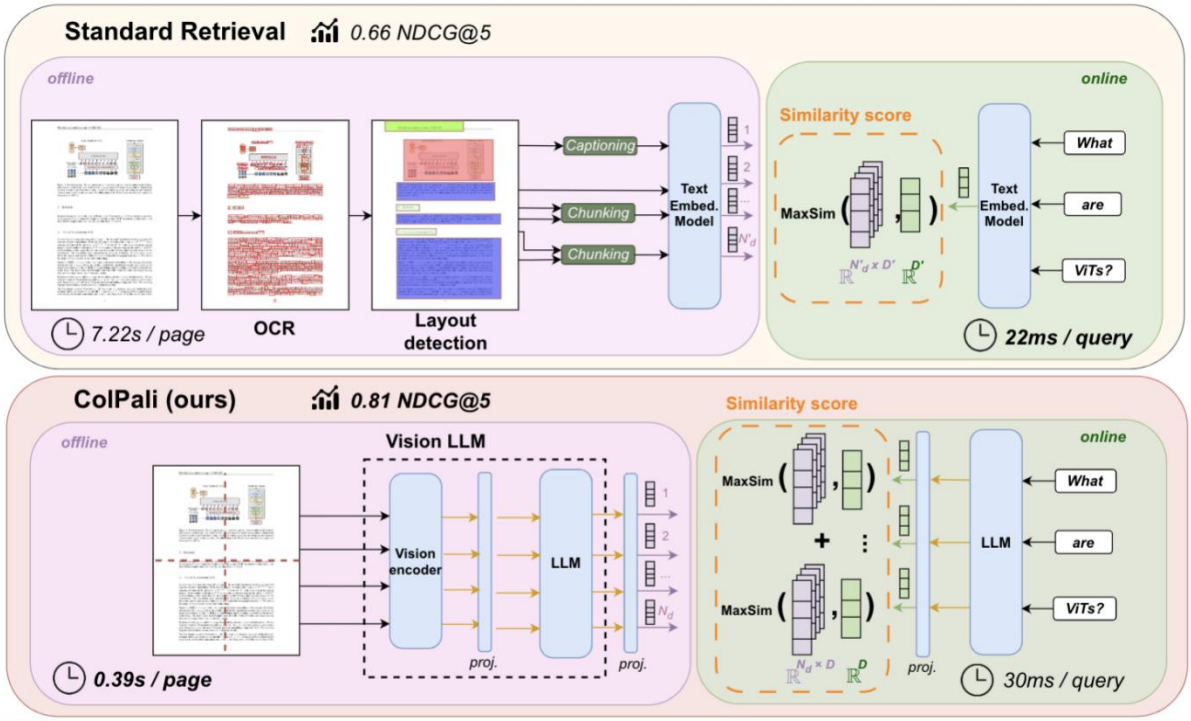

Figure 29 - Colpali Architecture

Training Process

ColPali is fine-tuned on a dataset of approximately 119,000 query-page pairs, sourced from academic benchmarks (e.g., DocVQA, InfoVQA) and synthetic data generated from web-crawled PDFs using advanced language models like Claude-3 Sonnet. Training employs Low-Rank Adaptation (LoRA) on the transformer layers and projection module, with a batch size of 32, bfloat16 precision, and a paged AdamW optimizer over one epoch.
The objective uses a contrastive loss based on late interaction scores: for a query-document pair, the positive score is maximized against the hardest in-batch negative. Queries are augmented with placeholder tokens for flexibility. This end-to-end training optimizes the entire

retrieval pipeline, including visual processing. Removals show that freezing the vision encoder and using multi-vector interaction are crucial for performance, while task-specific fine-tuning (e.g., for non-English languages) enhances adaptability without harming generalization.

How ColPali Works for Document Retrieval

ColPali operates in two main phases: offline indexing and online querying.

Initially, there is the indexing pipeline where document pages are rendered as images and passed through the model to generate multi-vector embeddings (one per patch/token). These are stored efficiently (about 257 KB per page in compressed form), requiring no OCR or captioning.

The Vision LLM (Large Language Model) layer is the core component of the indexing pipeline, responsible for processing document images and generating meaningful embeddings. In this layer, a document page is first rendered as an image and divided into patches. These patches are fed into a vision encoder, which extracts visual features from the image content (such as text, tables, or diagrams). The encoded features are then combined with a textual prompt and passed to the LLM part of the model. The LLM, built on a pre-trained language model like Gemma-2B, interprets the visual data in context with the prompt, producing a rich representation of the page.

A key feature of this layer is the projection (proj.) mechanism, which maps the output embeddings from both the vision encoder and the LLM into a lower-dimensional space (D=128). This creates multi-vector representations for each patch or token on the page, enabling fine-grained encoding of both visual and textual elements. The process is optimized for efficiency, taking approximately 0.39 seconds per page, and leverages pre-trained y and document embeddings, respectialignment between vision and language to capture the document's structure and content holistically. This end-to-end approach eliminates the need for separate OCR or layout analysis, making it a streamlined solution for document understanding.

After the pages have been indexed the user can query the system to get the relevant context. This happens thanks to the querying pipeline where the text query is embedded into multi-dimension vectors using the same model used in the indexing process. Relevance is computed via a late interaction mechanism: for each query vector, find the maximum similarity (dot product) to any document vector, then sum across all query vectors. This score enables precise matching that highlights prominent visual and textual elements.

In this pipeline, the core component is the Similarity Score layer that handles the retrieval process by matching a user query against the indexed document embeddings. During the online phase, a text query is processed by the same LLM to generate query embeddings in the same multi-vector format. The similarity between the query and document is computed using a late interaction mechanism.

In this mechanism, for each query vector, the system calculates the maximum similarity score (MaxSim) with any document vector across all patches or tokens. This is done by taking the dot product between corresponding vectors, denoted as $\langle E_q(i) \mid E_d(j) \rangle$, where E_q and E_d represent query and document embeddings, respectively. The total similarity score,

$$\mathrm{LI}(q,d) = \sum_{i \in [|1,N_q|]} \max_{j \in [|1,N_d|]} \langle \mathbf{E_q}^{(i)} | \mathbf{E_d}^{(j)} \rangle$$

is the sum of these maximum similarities across all query vectors. This approach allows the model to focus on the most relevant parts of the document, such as specific visual elements or text snippets that align with the query. The process is fast, taking about 30 milliseconds per query, and supports precise retrieval with an nDCG@5 score of 0.81, indicating high relevance in the top five results.

Together, these layers enable ColPali to efficiently index and retrieve information from visually rich documents, leveraging the strengths of vision-language integration and advanced similarity matching.

The method supports fast inference (e.g., 0.39 seconds per page for indexing, 30 ms for query encoding) and scalability with tools like Flash Attention. Additionally, it offers interpretability through heatmaps, visualizing which image patches align with query terms—revealing the model's implicit OCR and focus on features like charts or layouts.

Results and Conclusion

On the ViDoRe benchmark, ColPali achieves state-of-the-art nDCG@5 scores (81.3% average), outperforming text-based baselines by 14–30 points, especially on visual tasks like table extraction (+32.4%) and infographics (+29.5%). It also excels in text-heavy domains and generalizes zero-shot to languages like French. Compared to pipelines with OCR and captioning, ColPali is faster and more robust, with lower latency and storage needs post-compression.

In summary, ColPali represents a shift toward vision-centric document retrieval, simplifying workflows while enhancing accuracy through multimodal embeddings. Its end-to-end nature makes it adaptable for domain-specific applications, paving the way for more efficient RAG systems in research and industry.

6.7 RAG Pipeline Evaluation

The efficacy of a Retrieval-Augmented Generation (RAG) system cannot be taken for granted. While RAG architectures offer a powerful solution to the limitations of traditional Large Language Models (LLMs), their performance depends on the successful interplay of their distinct components: the retriever and the generator. A flaw in either component can compromise the quality of the final output, leading to irrelevant, inaccurate, or incomplete answers. Therefore, a systematic and rigorous evaluation methodology is not merely a final step but a critical aspect of the development lifecycle. This chapter outlines the primary approaches to evaluating RAG systems, covering both component-wise metrics and holistic, end-to-end metrics. After this, we will explain how we practically implemented these metrics to come to conclusions about which RAG pipeline configuration is the optimal.

6.7.1 RAG Pipeline Evaluation Metrics

In order to evaluate an LLM application, there are a lot of different metrics for specific aspects of our application. In our case, which is a RAG application, metrics specific to the RAG pipeline have been invented, so we can evaluate how effective the several parts of our system perform. Below we will discuss the most important of these metrics. The information about these metrics were derived from the documentation of a RAG end-to-end evaluation framework called RAGAS (RAG Assessment) [35]. This framework is the one we used to implement the evaluation of our system.

6.7.1.1 Component-Wise Evaluation Metrics

Component-wise metrics focus on dissecting the RAG pipeline into its core elements: the query, retrieved context, and generated answer. These metrics help identify bottlenecks, such as irrelevant retrievals or unfaithful generations, without requiring a full end-to-end assessment.

Faithfulness

Faithfulness quantifies the factual consistency between the AI-generated answer and the retrieved context. It ensures that the answer does not introduce hallucinations or unsubstantiated claims. The process involves decomposing the answer into simpler sentences and verifying each against the context using an LLM.

The faithfulness score is calculated as:

$$\text{Faithfulness score} = \frac{|\text{Number of claims in the generated answer that can be inferred from given context}|}{|\text{Total number of claims in the generated answer}|}$$

This metric requires two API calls: one LLM call to generate simpler sentences from the answer, and another to judge their faithfulness to the context.

The required data is:

- The original question/query
- The retrieved context
- The AI-generated answer

In practice, faithfulness scores can be visualized alongside supporting data, such as in tabular form, to facilitate analysis.

Answer Relevancy

Answer relevancy assesses how pertinent the AI-generated answer is to the original query. This metric is particularly complex, involving question generation and semantic similarity computations to detect non-committal or off-topic responses.

The process unfolds as follows:

1. An LLM generates a new question based on the answer and assigns a non-committal score (1 for non-committal, e.g., "I don't know", 0 otherwise).
2. Both the generated and original questions are embedded.
3. Cosine similarity is computed between the embeddings.
4. The final score is the product of the non-committal score and the cosine similarity.

The formula is:

$$\text{Answer relevancy} = \frac{1}{N}\sum_{i=1}^{N} \cos\left(E_{g_i}, E_o\right) = \frac{1}{N}\sum_{i=1}^{N} \frac{E_{g_i} \cdot E_o}{\| E_{g_i} \| \| E_o \|}$$

Where E_{g_i} is the embedding of the i-th generated question, E_o is the embedding of the original question, and N is the number of generated questions (default: 3).

This metric requires three API calls: one LLM call for question generation and non-committal scoring, and two embedding calls.

Data Required:

- Retrieved context
- AI-generated answer

An answer is deemed relevant when it directly and appropriately addresses the original question. Relevancy scores are useful for debugging responses that stray from the query's intent.

Context Recall

Context recall measures how comprehensively the retrieved context covers the ground truth answer, similar to traditional recall in information retrieval. It evaluates each statement in the ground truth to determine if it can be attributed to the context.

The score is:

$$\text{Context recall} = \frac{|\text{GT sentences that can be attributed to context}|}{|\text{Number of sentences in GT}|}$$

This requires one LLM call to attribute ground truth sentences to the context.

Data Required:

- Original question/query
- Retrieved context
- Ground truth (post-generation reference answer)

High context recall indicates effective retrieval mechanisms.

Context Precision

Context precision evaluates the utility of the retrieved context in deriving the AI answer, similar to precision in statistics. It assesses whether each context chunk contributes meaningfully to the response.

The calculation involves:

1. An LLM categorizes each context chunk as helpful (1) or not (0).
2. Precision is computed iteratively, weighted by relevance.

The formula is:

$$\text{Context Precision@K} = \frac{\sum_{k=1}^{K}(\text{Precision@k} \times v_k)}{\text{Total number of relevant items in the top } K \text{ results}}$$

$$\text{Precision@k} = \frac{\text{true positives@k}}{\text{true positives@k} + \text{false positives@k}}$$

Where K is the number of context chunks, and $v_k \in \{0,1\}$ is the relevance at rank k.

This metric requires one LLM call for verdicts.

Data Required:

- Original question/query
- Retrieved context
- AI-generated answer

Context precision is essential for optimizing retrieval to avoid noise.

Context Entity Recall

Context entity recall focuses on entities (e.g., names, concepts) rather than full text. It computes the intersection of entities between the context and ground truth.

The score is:

$$\text{Context entity recall} = \frac{|CE \cap GE|}{|GE|}$$

Where CE are context entities and GE are ground truth entities.

This requires two LLM calls: one for extracting entities from the context and one from the ground truth.

Data Required:

- Retrieved context
- Ground truth

This metric is valuable for entity-centric domains like knowledge graphs.

6.7.1.2 End-to-End Evaluation Metrics

Answer Semantic Similarity

Answer semantic similarity evaluates the closeness between the AI-generated answer and the ground truth. It is computed by embedding both texts and measuring cosine similarity:

$$\text{Semantic similarity} = \frac{E_a \cdot E_{gt}}{\| E_a \| \| E_{gt} \|}$$

Where E_a is the embedding of the AI answer and E_{gt} is the embedding of the ground truth. This metric provides a semantic rather than lexical comparison, making it robust to paraphrasing.

Answer Correctness

Answer correctness is a hybrid metric that combines statement-level validation with semantic similarity. It works as follows:

3. Decompose both the AI answer and ground truth into simpler statements.
4. Classify statements as True Positive (TP), False Positive (FP), or False Negative (FN).
5. Compute an F1 score:

$$F1 = \frac{2 \times TP}{2 \times TP + FP + FN}$$

4. Compute semantic similarity between the AI answer and the ground truth.
5. Aggregate both measures using a weighted average (default weighting: 0.75 for F1, 0.25 for similarity).

This provides a balanced measure of both factual correctness and semantic alignment.

6.7.1.3 Aspect Critiques

Aspect critiques represent a qualitative evaluation approach, where the LLM judges answers along multiple dimensions of quality. Each aspect yields a binary score (0 or 1). Supported dimensions include:

- **Harmfulness**: Potential for causing harm.
- **Maliciousness**: Intention to deceive or exploit.
- **Coherence**: Logical organization of content.
- **Correctness**: Factual accuracy.
- **Conciseness**: Brevity and avoidance of redundancy.

Although aspect critiques can provide useful diagnostic insights, their implementation currently suffers from limitations in strictness and consistency.

6.7.1.4 Conclusion of Evaluation Metrics

The metrics outlined in this chapter offer a robust toolkit for evaluating RAG systems. Component-wise metrics enable granular debugging, end-to-end metrics assess overall efficacy, and aspect critiques address qualitative concerns. In the following section we will analyze the RAG pipeline evaluation in practice by describing how we implemented some of the above metrics to achieve the best results in our system.

6.7.2 RAG Pipeline Evaluation in Practice

Following the discussion of the various evaluation metrics for Retrieval-Augmented Generation (RAG), this section focuses on the practical assessment of our own application with respect to the most relevant of these metrics. In designing the evaluation process, we experimented with different parameter configurations, specifically varying the chunk size, the degree of chunk overlap, and the number of retrieved chunks.

It is important to emphasize that a wide range of alternative configurations and evaluation criteria could have been considered. The RAG pipeline offers numerous tunable aspects, and the evaluation can be conducted across multiple dimensions, depending on the underlying evaluation data and the chosen metrics. However, for the purposes of this study, we deliberately

selected the aforementioned parameters, as we identified them to be the most critical factors influencing the performance of our system.

The first step was deciding the configurations. We chose 4 different configurations and 3 different metrics which we deemed most important. For the first configuration, the initial PDF was loaded using the default load_document function, which applies the LLMImageBlobParser class to replace images with their textual descriptions. In contrast, the second PDF was processed using the load_document_OCR function, where images were replaced with OCR-extracted text. In this setup, the chunk size was set to 2800, with no chunk overlap, and the retrieval parameter specified two chunks to be retrieved. Three additional configurations were then tested, all using the default load_document function but with different chunking parameters. The second configuration employed a chunk size of 2000 with no overlap, retrieving two chunks. The third configuration used a chunk size of 3500 with an overlap of 100, also retrieving two chunks. Finally, the fourth configuration applied a chunk size of 4200 with an overlap of 100, retrieving only one chunk.
After applying the configurations, we generated a CSV file for each configuration with the columns: "user_input," "retrieved_contexts," "response," "reference," "faithfulness," "context_precision," and "answer_correctness." The first four columns formed the test dataset, while the last three represented the evaluation metrics. We calculated the average for each metric across all configurations and created bar plots to compare the average values for each metric, as shown in the three figures below.

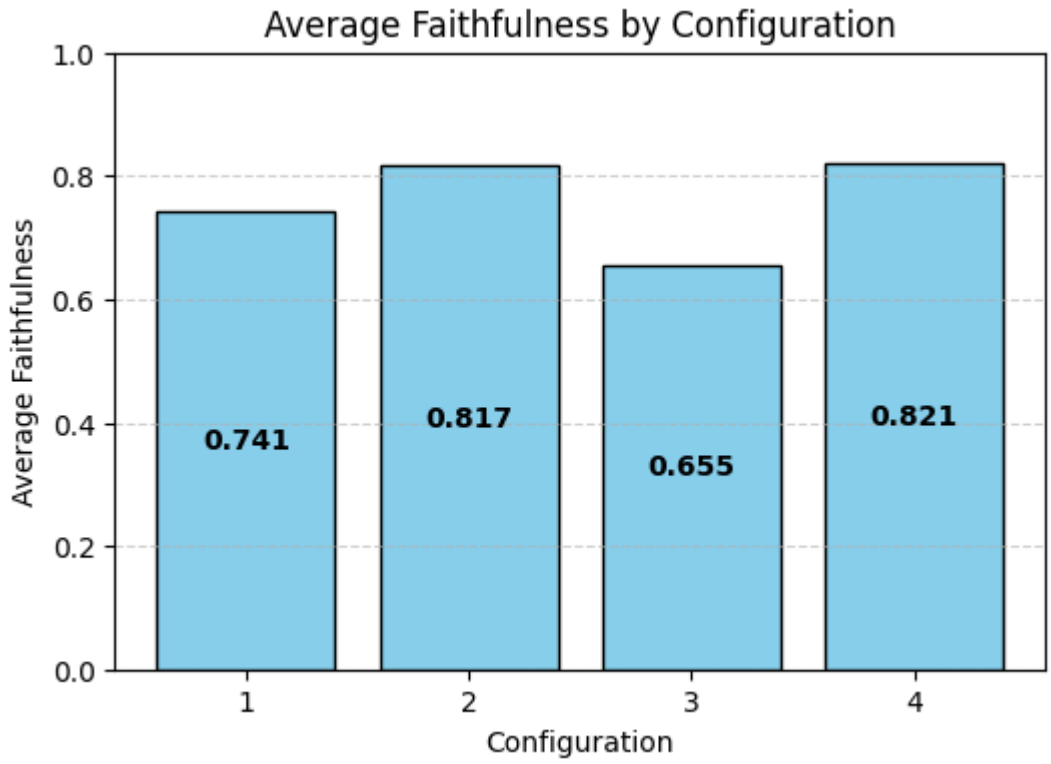

Figure 30 - Bar Plot of Average Faithfulness by Configuration.

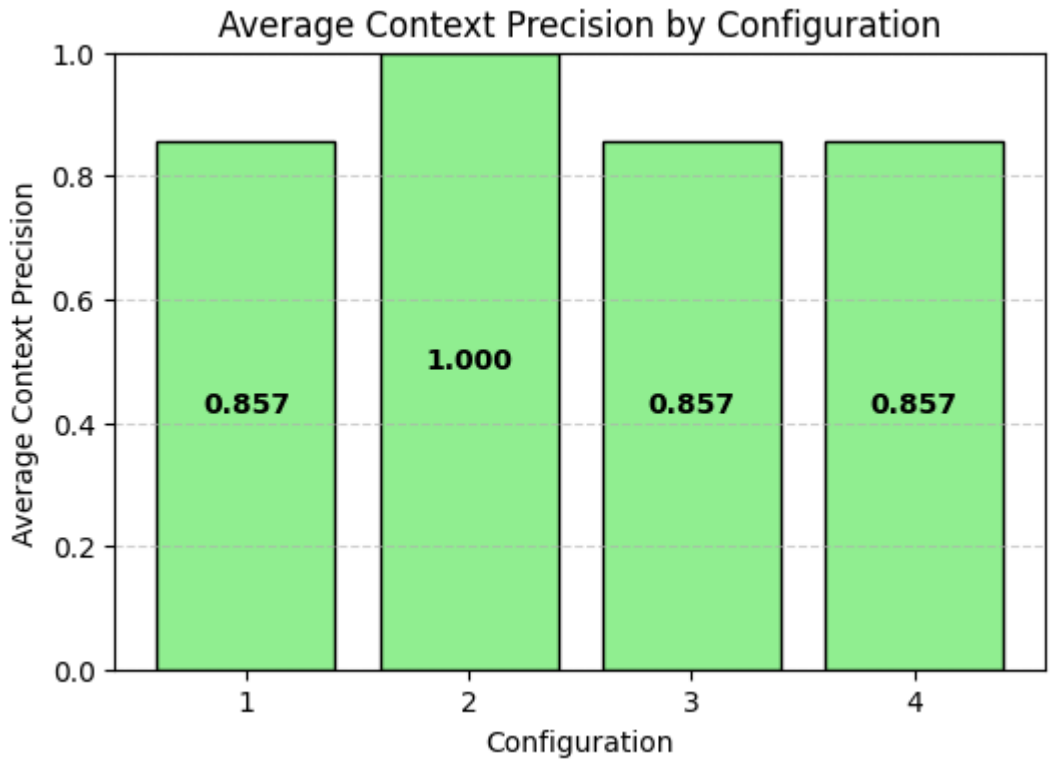

Figure 31 - Bar Plot of Average Context Precision by Configuration.

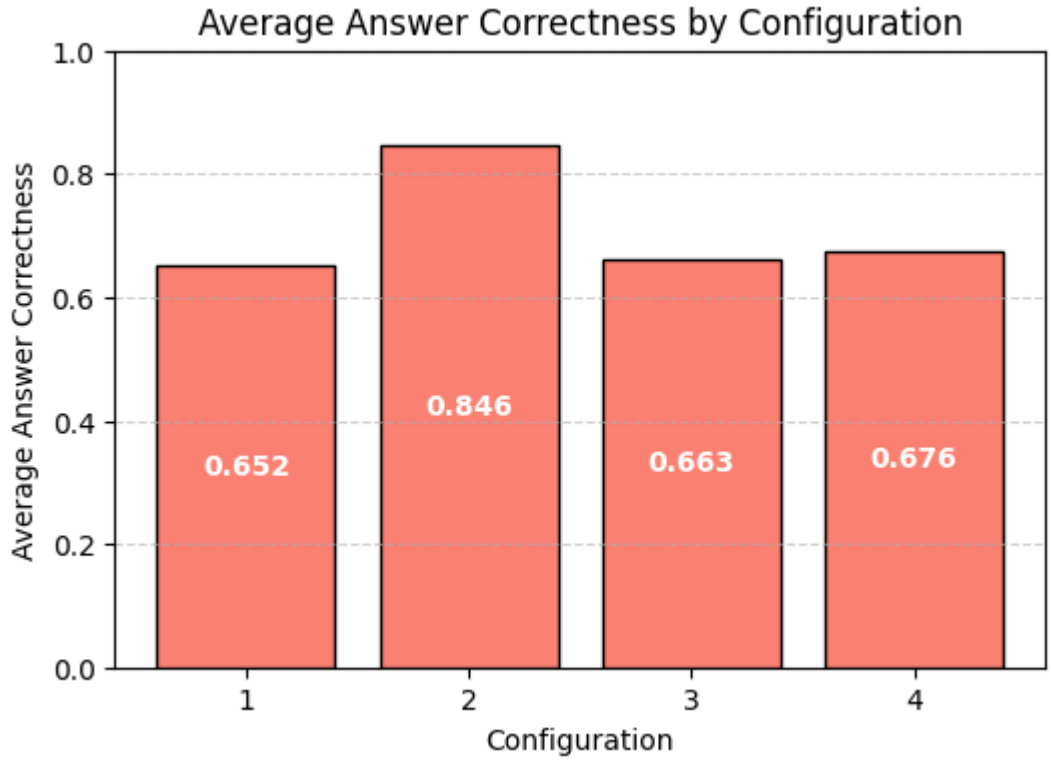

Figure 32 - Bar Plot of Average Answer Correctness by Configuration.

The above evaluation results reveal distinct performance profiles across the four configurations in our retrieval-augmented generation (RAG) pipeline.

Faithfulness scores, which assess the absence of hallucinations by measuring alignment between generated answers and retrieved context, peak at 0.821 for Configuration 4 and 0.817 for Configuration 2, while Configuration 3 lags notably at 0.655, indicating potential overgeneration or context misalignment.

Context Precision, evaluating the relevance and ranking quality of retrieved documents, achieves 1.000 in Configuration 2, surpassing the consistent but lower 0.857 across the others, underscoring superior retrieval efficacy in this setup.

Answer Correctness, measuring semantic accuracy against ground truth, favors Configuration 2 at 0.846, markedly ahead of Configurations 1 (0.652), 3 (0.663), and 4 (0.676), suggesting more reliable end-to-end response quality.

Collectively, Configuration 2 demonstrates the optimal balance of high faithfulness, flawless context retrieval, and superior correctness, making it the recommended choice for deployment in this RAG system to minimize errors and enhance reliability. Of course, as previously noted, numerous additional configurations, along with a broader range of input questions and corresponding answers, could be tested to achieve fully precise evaluation outcomes. However, implementing such an extensive approach would entail significantly more time and resource expenditures. For this reason, we continued our application development with the second configuration which performed better based in this specific evaluation implementation.

In the next chapter we will continue by explaining how we deployed our RAG application and created the end-user interface.

6.8 End-to-End Application Deployment

The present chapter outlines the procedural framework adopted for the deployment of the chatbot system to its intended end-users. The discussion commences with an examination of the Streamlit framework, which served as the foundation for the application's front-end interface. Following this, the chapter provides a comprehensive description of the development and deployment process, ending in a demonstration of the fully functional application.

6.8.1 Streamlit Overview

Before continuing with the actual deployment process of our RAG application we will provide an overview of the Streamlit framework.

Streamlit [112] is an open-source framework designed to simplify the development of interactive web applications for data science and machine learning. Its primary appeal lies in its simplicity and ease of use, allowing data scientists, researchers, and engineers to build and deploy interactive applications with minimal effort, all while focusing on the core functionality rather than complex frontend design. The framework is Python-based, meaning developers can leverage their existing knowledge of Python libraries like Pandas, NumPy, and Matplotlib to quickly prototype and visualize data.

One of the defining characteristics of Streamlit is its emphasis on rapid prototyping. Unlike traditional web development frameworks that require knowledge of HTML, CSS, and JavaScript, Streamlit abstracts away much of the frontend complexity. The syntax is intuitive and highly readable, allowing users to focus on the logic and interactivity of the application. With just a few lines of code, developers can create widgets such as sliders, buttons, and dropdown menus, which can be dynamically linked to the underlying data or model outputs. This capability is particularly useful for creating data-driven applications, dashboards, or machine learning tools that require user interaction.

The framework's integration with machine learning models is another key strength. Streamlit supports the seamless integration of pre-trained models, enabling data scientists to deploy their models as web applications without the need for complex backend infrastructure. This is especially valuable for rapid testing, model validation, or sharing insights with stakeholders who may not have a deep technical background. Additionally, Streamlit provides the ability to visualize data in real-time, making it ideal for exploring datasets, visualizing trends, and tracking changes in model performance.

In the context of a Retrieval-Augmented Generation (RAG) application, Streamlit can play a pivotal role in presenting and interacting with the outputs generated by the model. For example, an application could allow users to input queries, retrieve relevant information from a knowledge base, and display the model’s generated responses in an easy-to-read format. By providing an intuitive user interface (UI), Streamlit ensures that non-expert users can interact with advanced machine learning systems with ease. The framework's real-time rendering capabilities allow users to observe changes in the model’s output, helping them gain insights into the underlying process and improve decision-making.

Overall, Streamlit's simplicity, ease of deployment, and focus on interactivity make it a powerful tool for developing web-based applications, particularly in the fields of data science, machine learning, and natural language processing. Its relevance in the context of a RAG application is evident, as it allows for the creation of user-friendly interfaces that can effectively showcase and interact with complex machine learning models.

6.8.2 Streamlit Application Deployment

Having provided a comprehensive overview of the Streamlit framework, we now proceed to outline the process we followed for deploying our application.

First, after determining the most suitable RAG pipeline configuration (as discussed in the previous chapter), we installed Streamlit in our project directory and created a dedicated Python file to host the application. This file defined several key components of the system: a function for loading the embeddings generated in earlier stages of the process, a function responsible for generating responses, as well as the user interface elements. Specifically, the interface included a title, a sidebar that enabled users to input their credentials and submit their queries, and various components for rendering the chat interface. The whole code is available in the project directory. After testing that the application functioned in the desired way, we continued with the deployment of the application.

For the deployment of our application, we used Streamlit's built-in deployment functionality. The first step was to upload the source code to github and then, after following Streamlit's docs instructions we deployed the app and made it available to the URL: "https://diplomaappdeployment.streamlit.app/".

Below we can see the final interface of our application, along with an example of questions based on the "ATHENA" user guide and their answers.

Figure 33 - End-User interface of our RAG application.

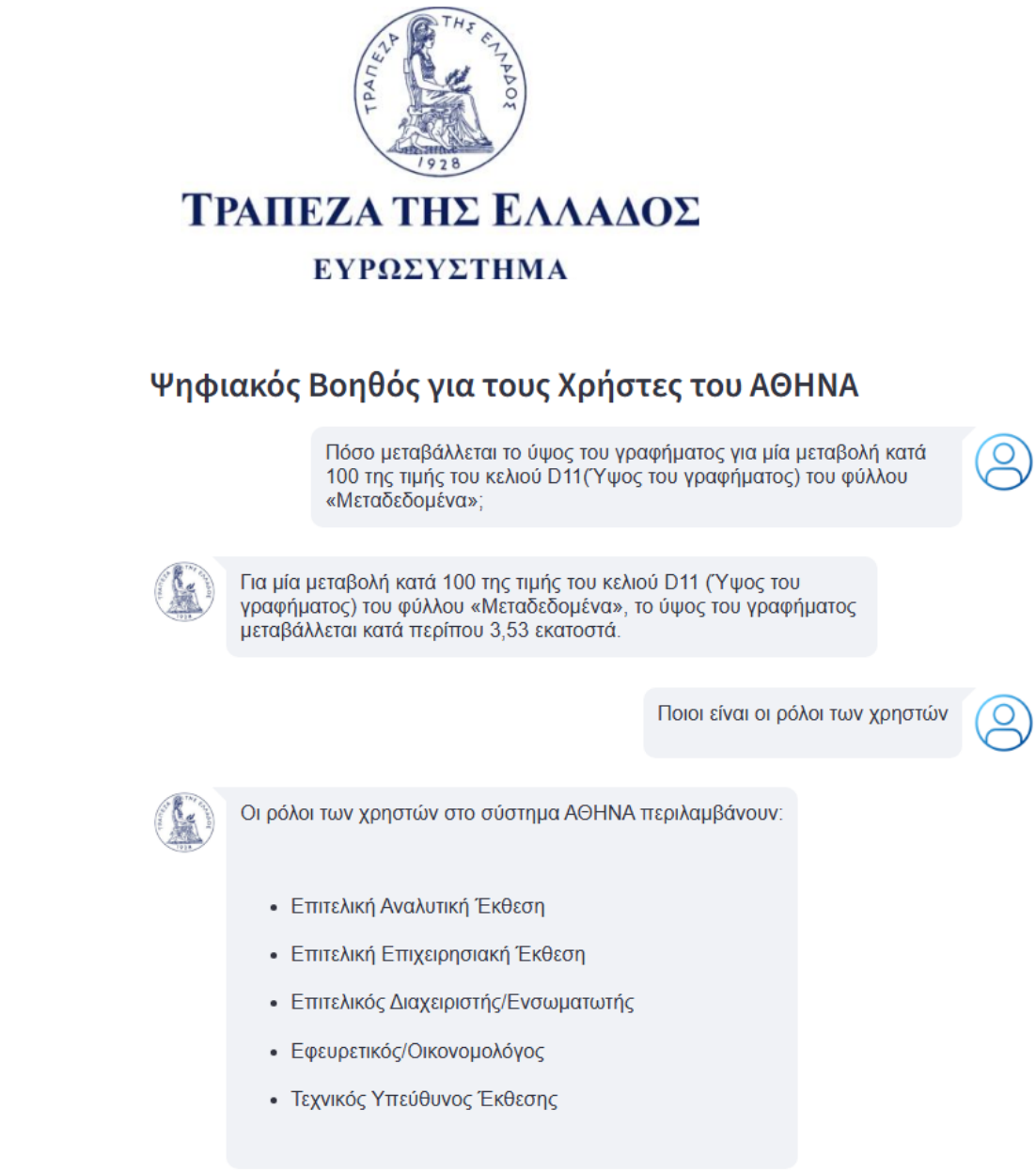

Figure 34 - Example of questions and their answers in our final RAG application.

7 Conclusion

This diploma thesis detailed the design, implementation, and evaluation of a Retrieval-Augmented Generation (RAG) chatbot system, engineered to serve as an interactive guide for the "Athena" application. The primary objective was to create a tool that could effectively understand user queries and retrieve accurate, contextually relevant information directly from the user guide documentation. The implemented system was proven successful in this endeavor, demonstrating a robust capacity to satisfactorily answer a wide range of questions. By providing immediate and targeted access to information, the chatbot significantly streamlines the user's experience, mitigating the need for manual navigation through extensive documents and thereby enhancing overall usability.

However, it is imperative to maintain a critical perspective on the system's limitations. While the architectural choices and techniques employed (specifically within the RAG pipeline) were explicitly selected to constrain hallucinations and improve factual accuracy, the inherent probabilistic nature of generative AI models means that the possibility of producing an inaccurate or erroneous response cannot be completely eradicated. This acknowledgment is not a shortcoming but a fundamental characteristic of current technology, underscoring the need for continuous refinement.

Looking forward, the performance and reliability of the chatbot can be advanced through several strategic initiatives of further development. A primary area for investigation involves the core models that power the system. The exploration and integration of more advanced language models, particularly those with superior capabilities in comprehending the syntactic and semantic nuances of the Greek language, could yield substantial improvements in both the retrieval of relevant text passages and the generation of fluent, precise answers.

Equally critical to the system's success is the quality and structure of its underlying knowledge base. The process of data extraction, chunking, and vector storage was a major factor influencing response quality. Future work should, therefore, investigate more sophisticated data processing techniques. For instance, the handling of non-textual elements like complex tables, diagrams and images presents a particular challenge. Developing or integrating more advanced methods for parsing and representing this structured information would ensure a more comprehensive and accurate knowledge base for the model to draw upon.

Finally, the intrinsic quality of the source material itself is a determining factor. The clarity, simplicity, and comprehensiveness of the "Athena" User Guide directly correlate with the quality of the chatbot's outputs. Enriching the dataset with an expanded collection of Frequently Asked Questions (FAQs), detailed step-by-step tutorials, and concrete examples covering a broader spectrum of application use cases would provide a stronger foundation for

the language model. In essence, a well-structured and exhaustive source document naturally leads to more precise and helpful generated responses.

To summarize, this thesis has established a functional and effective RAG-based chatbot system that fulfills its primary objective. The foundation laid here is solid, yet it also opens up numerous possibilities for enhancement. By pursuing improvements in model sophistication, data processing pipelines, and source material quality, the system's accuracy, depth, and utility can be progressively elevated to meet even more demanding user needs.

Bibliography

[1] Κ. Διαμαντάρας and Δ. Μπότσης, *Μηχανική Μάθηση*. Εκδόσεις Κλειδάριθμος, 2019.

[2] "Supervised learning - Wikipedia." Accessed: Aug. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Supervised_learning

[3] "Reinforcement learning - Wikipedia." Accessed: Aug. 30, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Reinforcement_learning

[4] "Reinforcement Learning - GeeksforGeeks." Accessed: Aug. 30, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/what-is-reinforcement-learning/

[5] "Feedforward Neural Network - GeeksforGeeks." Accessed: Aug. 30, 2025. [Online]. Available: https://www.geeksforgeeks.org/nlp/feedforward-neural-network/

[6] "Convolutional Neural Network | Deep Learning | Developers Breach." Accessed: Aug. 30, 2025. [Online]. Available: https://developersbreach.com/convolution-neural-network-deep-learning/

[7] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network," *Physica D*, vol. 404, Mar. 2020, doi: 10.1016/j.physd.2019.132306.

[8] A. Vaswani *et al.*, "Attention Is All You Need," 2023.

[9] "What Is NLP (Natural Language Processing)? | IBM." Accessed: Sep. 06, 2025. [Online]. Available: https://www.ibm.com/think/topics/natural-language-processing

[10] "Tokenization in NLP - GeeksforGeeks." Accessed: Sep. 06, 2025. [Online]. Available: https://www.geeksforgeeks.org/nlp/nlp-how-tokenizing-text-sentence-words-works

[11] "To Use or Lose: Stop Words in NLP | by Moirangthem Gelson Singh | Medium." Accessed: Sep. 06, 2025. [Online]. Available: https://medium.com/@gelsonm/to-use-or-lose-stop-words-in-nlp-de946edaa468

[12] "What Is Stemming? | IBM." Accessed: Sep. 06, 2025. [Online]. Available: https://www.ibm.com/think/topics/stemming

[13] "Lemmatization - Wikipedia." Accessed: Sep. 06, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Lemmatization

[14] "Stemming and lemmatization." Accessed: Sep. 06, 2025. [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[15] "Named-entity recognition - Wikipedia." Accessed: Sep. 07, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Named-entity_recognition

[16] "Named Entity Recognition - GeeksforGeeks." Accessed: Sep. 07, 2025. [Online]. Available: https://www.geeksforgeeks.org/nlp/named-entity-recognition/

[17] "Categorical data: Vocabulary and one-hot encoding | Machine Learning | Google for Developers." Accessed: Sep. 07, 2025. [Online]. Available: https://developers.google.com/machine-learning/crash-course/categorical-data/one-hot-encoding

[18] “One Hot Encoding in Machine Learning - GeeksforGeeks.” Accessed: Aug. 25, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/ml-one-hot-encoding/
[19] “Bag-of-words model - Wikipedia.” Accessed: Sep. 08, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Bag-of-words_model
[20] “A Gentle Introduction to the Bag-of-Words Model - MachineLearningMastery.com.” Accessed: Sep. 08, 2025. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-bag-words-model/
[21] “N-grams in NLP. N-grams, a fundamental concept in NLP… | by Abhishek Jain | Medium.” Accessed: Aug. 25, 2025. [Online]. Available: https://medium.com/@abhishekjainindore24/n-grams-in-nlp-a7c05c1aff12
[22] “N-gram in NLP - GeeksforGeeks.” Accessed: Aug. 25, 2025. [Online]. Available: https://www.geeksforgeeks.org/nlp/n-gram-in-nlp/
[23] “Understanding TF-IDF (Term Frequency-Inverse Document Frequency) - GeeksforGeeks.” Accessed: Sep. 09, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/understanding-tf-idf-term-frequency-inverse-document-frequency/
[24] “Word embedding - Wikipedia.” Accessed: Sep. 09, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Word_embedding
[25] “Word Embeddings in NLP - GeeksforGeeks.” Accessed: Aug. 25, 2025. [Online]. Available: https://www.geeksforgeeks.org/nlp/word-embeddings-in-nlp/
[26] “Word2vec - Wikipedia.” Accessed: Sep. 09, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Word2vec
[27] “Word Embedding using Word2Vec - GeeksforGeeks.” Accessed: Aug. 25, 2025. [Online]. Available: https://www.geeksforgeeks.org/python/python-word-embedding-using-word2vec/
[28] “Large language model - Wikipedia.” Accessed: Aug. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Large_language_model
[29] “Zero-Shot Prompting: Examples, Theory, Use Cases | DataCamp.” Accessed: Sep. 21, 2025. [Online]. Available: https://www.datacamp.com/tutorial/zero-shot-prompting
[30] “What is One Shot Prompting? | IBM.” Accessed: Sep. 21, 2025. [Online]. Available: https://www.ibm.com/think/topics/one-shot-prompting
[31] Y. Ban, R. Wang, T. Zhou, M. Cheng, B. Gong Google, and C.-J. Hsieh, “Understanding the Impact of Negative Prompts: When and How Do They Take Effect?,” Jun. 2024, Accessed: Sep. 21, 2025. [Online]. Available: https://arxiv.org/pdf/2406.02965v1
[32] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models Chain-of-Thought Prompting”.
[33] P. Lewis *et al.*, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”, Accessed: Sep. 21, 2025. [Online]. Available: https://github.com/huggingface/transformers/blob/master/
[34] “Introduction | 🦜🔗 LangChain.” Accessed: Sep. 24, 2025. [Online]. Available: https://python.langchain.com/docs/introduction/

[35] “Introduction | Ragas.” Accessed: Sep. 26, 2025. [Online]. Available: https://docs.ragas.io/en/v0.1.21/index.html
[36] “What Is a Chatbot? | Oracle.” Accessed: Aug. 11, 2025. [Online]. Available: https://www.oracle.com/chatbots/what-is-a-chatbot/
[37] “Machine learning - Wikipedia.” Accessed: Aug. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning
[38] “Linear regression - Wikipedia.” Accessed: Aug. 12, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Linear_regression
[39] “Logistic Regression in Machine Learning - GeeksforGeeks.” Accessed: Aug. 17, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/understanding-logistic-regression
[40] “Logistic regression - Wikipedia.” Accessed: Aug. 17, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Logistic_regression
[41] “Logistic regression | Definition & Facts | Britannica.” Accessed: Aug. 27, 2025. [Online]. Available: https://www.britannica.com/science/logistic-regression
[42] “What Is Logistic Regression? | Master’s in Data Science.” Accessed: Aug. 27, 2025. [Online]. Available: https://www.mastersindatascience.org/learning/machine-learning-algorithms/logistic-regression
[43] “scikit-learn: machine learning in Python — scikit-learn 1.7.1 documentation.” Accessed: Aug. 27, 2025. [Online]. Available: https://scikit-learn.org/stable/
[44] “Structure of Perceptron | Download Scientific Diagram.” Accessed: Aug. 30, 2025. [Online]. Available: https://www.researchgate.net/figure/Structure-of-Perceptron_fig2_330742498
[45] A. Sherstinsky, “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network”, Accessed: Sep. 06, 2025. [Online]. Available: https://www.linkedin.com/in/alexsherstinsky
[46] R. C. Staudemeyer and E. R. Morris, “Understanding LSTM-a tutorial into Long Short-Term Memory Recurrent Neural Networks,” 2019.
[47] “A neural network with 2 hidden layers | Download Scientific Diagram.” Accessed: Sep. 06, 2025. [Online]. Available: https://www.researchgate.net/figure/A-neural-network-with-2-hidden-layers_fig3_351347266
[48] “Tokenization in NLP : Definition ,Types and Techniques.” Accessed: Sep. 06, 2025. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp
[49] “Tokenization in NLP : All you need to know | by Abdallah Ashraf | Medium.” Accessed: Sep. 06, 2025. [Online]. Available: https://medium.com/%40abdallahashraf90x/tokenization-in-nlp-all-you-need-to-know-45c00cfa2df7
[50] “Byte-pair encoding - Wikipedia.” Accessed: Sep. 06, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Byte-pair_encoding
[51] “The Technical User’s Introduction to LLM Tokenization.” Accessed: Sep. 06, 2025. [Online]. Available: https://christophergs.com/blog/understanding-llm-tokenization

[52] “Linguistic Laws Meet Protein Sequences: A Comparative Analysis of Subword Tokenization Methods.” Accessed: Sep. 06, 2025. [Online]. Available: https://arxiv.org/html/2411.17669v1
[53] “Introduction to Stemming - GeeksforGeeks.” Accessed: Sep. 06, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/introduction-to-stemming/
[54] “Stemming in NLP: Key Concepts and Fundamentals Explained.” Accessed: Sep. 06, 2025. [Online]. Available: https://botpenguin.com/glossary/stemming
[55] “Lemmatization in NLP. Lemmatization is a more advanced and… | by Kevinnjagi | Medium.” Accessed: Sep. 06, 2025. [Online]. Available: https://medium.com/@kevinnjagi83/lemmatization-in-nlp-2a61012c5d66
[56] “Lemmatization: Key Componenets, Benefits & Types| BotPenguin.” Accessed: Sep. 06, 2025. [Online]. Available: https://botpenguin.com/glossary/lemmatization
[57] “POS(Parts-Of-Speech) Tagging in NLP - GeeksforGeeks.” Accessed: Sep. 07, 2025. [Online]. Available: https://www.geeksforgeeks.org/nlp/nlp-part-of-speech-default-tagging/
[58] “Part-of-speech tagging - Wikipedia.” Accessed: Sep. 07, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Part-of-speech_tagging
[59] P. Wang, Y. Qian, F. K. Soong, L. He, and H. Zhao, “Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network,” Oct. 2015, Accessed: Sep. 07, 2025. [Online]. Available: https://arxiv.org/pdf/1510.06168
[60] “Penn Treebank P.O.S. Tags.” Accessed: Sep. 07, 2025. [Online]. Available: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
[61] “Universal POS tags.” Accessed: Sep. 07, 2025. [Online]. Available: https://universaldependencies.org/u/pos/
[62] “Named Entity Recognition - GeeksforGeeks.” Accessed: Sep. 07, 2025. [Online]. Available: https://www.geeksforgeeks.org/nlp/named-entity-recognition
[63] “A Brief History of Named Entity Recognition.” Accessed: Sep. 07, 2025. [Online]. Available: https://arxiv.org/html/2411.05057v1
[64] V. Yadav and S. Bethard, “A Survey on Recent Advances in Named Entity Recognition from Deep Learning models,” 2019, Accessed: Sep. 07, 2025. [Online]. Available: http://2016.bionlp-st.org/tasks/bb2
[65] “One-hot - Wikipedia.” Accessed: Sep. 07, 2025. [Online]. Available: https://en.wikipedia.org/wiki/One-hot
[66] “One Hot Encoding in Machine Learning - GeeksforGeeks.” Accessed: Sep. 07, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/ml-one-hot-encoding
[67] “N-grams in NLP. N-grams, a fundamental concept in NLP… | by Abhishek Jain | Medium.” Accessed: Sep. 08, 2025. [Online]. Available: https://medium.com/@abhishekjainindore24/n-grams-in-nlp-a7c05c1aff12
[68] J. Daniel and J. H. Martin, “Speech and Language Processing,” 2025.
[69] “N-gram in NLP - GeeksforGeeks.” Accessed: Sep. 08, 2025. [Online]. Available: https://www.geeksforgeeks.org/nlp/n-gram-in-nlp/

[70] “TF-IDF: Weighing Importance in Text - Let’s Data Science.” Accessed: Sep. 09, 2025. [Online]. Available: https://letsdatascience.com/tf-idf/
[71] R. Stuart and P. Norvig, *Τεχνητή Νοημοσύνη: Μια σύγχρονη προσέγγιση*, 4th ed. Εκδόσεις Κλειδάριθμος, 2021.
[72] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space”, Accessed: Sep. 09, 2025. [Online]. Available: http://ronan.collobert.com/senna/
[73] “Model architecture of (A) CBOW and (B) Skip-gram. | Download Scientific Diagram.” Accessed: Sep. 09, 2025. [Online]. Available: https://www.researchgate.net/figure/Model-architecture-of-A-CBOW-and-B-Skip-gram_fig1_335355568
[74] “Measuring similarity from embeddings | Machine Learning | Google for Developers.” Accessed: Sep. 09, 2025. [Online]. Available: https://developers.google.com/machine-learning/clustering/dnn-clustering/supervised-similarity
[75] H. Touvron *et al.*, “LLaMA: Open and Efficient Foundation Language Models”, Accessed: Sep. 17, 2025. [Online]. Available: https://github.com/facebookresearch/xformers
[76] A. R. Openai, K. N. Openai, T. S. Openai, and I. S. Openai, “Improving Language Understanding by Generative Pre-Training”, Accessed: Sep. 17, 2025. [Online]. Available: https://gluebenchmark.com/leaderboard
[77] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language Models are Unsupervised Multitask Learners”, Accessed: Sep. 17, 2025. [Online]. Available: https://github.com/codelucas/newspaper
[78] T. B. Brown *et al.*, “Language Models are Few-Shot Learners,” 2020.
[79] L. Ouyang *et al.*, “Training language models to follow instructions with human feedback”.
[80] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. K. Openai, “Proximal Policy Optimization Algorithms”.
[81] M. Chen *et al.*, “Evaluating Large Language Models Trained on Code”, Accessed: Sep. 17, 2025. [Online]. Available: https://www.github.com/openai/human-eval.
[82] OpenAI, “GPT-4 Technical Report”.
[83] S. Kadavath *et al.*, “Language Models (Mostly) Know What They Know,” 2022.
[84] A. M. Turing, “COMPUTING MACHINERY AND INTELLIGENCE,” *Computing Machinery and Intelligence. Mind*, vol. 49, pp. 433–460, 1950.
[85] Joseph Weizenbaum, “Computational Linguistics.” Accessed: Sep. 21, 2025. [Online]. Available: https://web.stanford.edu/class/cs124/p36-weizenabaum.pdf
[86] K. M. Colby, S. Weber, and F. D. Hilf, “Artificial Paranoia’,” *Artif Intell*, vol. 2, pp. 1–25, 1971.
[87] “The Elements of AIML Style,” 2003.
[88] L. Voukoutis *et al.*, “Meltemi: The first open Large Language Model for Greek,” 2024, Accessed: Sep. 21, 2025. [Online]. Available: https://huggingface.co/ilsp/

[89] D. Roussis *et al.*, “Krikri: Advancing Open Large Language Models for Greek,” 2025, Accessed: Sep. 21, 2025. [Online]. Available: https://huggingface.co/datasets/wikimedia/wikisource
[90] A. Q. Jiang *et al.*, “Mistral 7B”.
[91] H. Touvron *et al.*, “LLaMA: Open and Efficient Foundation Language Models”, Accessed: Sep. 21, 2025. [Online]. Available: https://github.com/facebookresearch/xformers
[92] E. Almazrouei *et al.*, “The Falcon Series of Open Language Models The Falcon LLM Team *,” 2023, Accessed: Sep. 21, 2025. [Online]. Available: https://huggingface.co/tiiuae/
[93] J. White *et al.*, “A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT,” 2023.
[94] “Llama 3.1 | Model Cards and Prompt formats.” Accessed: Sep. 21, 2025. [Online]. Available: https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_1/
[95] “Overview - OpenAI API.” Accessed: Sep. 21, 2025. [Online]. Available: https://platform.openai.com/docs/overview
[96] T. B. Brown *et al.*, “Language Models are Few-Shot Learners,” 2020.
[97] “What is prompt chaining? | IBM.” Accessed: Sep. 21, 2025. [Online]. Available: https://www.ibm.com/think/topics/prompt-chaining
[98] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models Chain-of-Thought Prompting”.
[99] S. Yao *et al.*, “Tree of Thoughts: Deliberate Problem Solving with Large Language Models”, Accessed: Sep. 21, 2025. [Online]. Available: https://github.com/princeton-nlp/tree-of-thought-llm.
[100] M. Besta *et al.*, “Graph of Thoughts: Solving Elaborate Problems with Large Language Models”, Accessed: Sep. 21, 2025. [Online]. Available: https://github.com/spcl/graph-of-thoughts
[101] “Prompt Engineering Guide: Unlocking the Potential of AI Models.” Accessed: Sep. 21, 2025. [Online]. Available: https://www.eweek.com/artificial-intelligence/guide-to-prompt-engineering/
[102] Z. Ji *et al.*, “LLM Internal States Reveal Hallucination Risk Faced With a Query,” *BlackboxNLP 2024 - 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP - Proceedings of the Workshop*, pp. 88–104, Jul. 2024, doi: 10.18653/v1/2024.blackboxnlp-1.6.
[103] M. Binz *et al.*, “How should the advent of large language models affect the practice of science?,” *Proc Natl Acad Sci U S A*, vol. 122, no. 5, Dec. 2023, doi: 10.1073/pnas.2401227121.
[104] Y. Guo *et al.*, “Bias in Large Language Models: Origin, Evaluation, and Mitigation,” Nov. 2024, Accessed: Sep. 21, 2025. [Online]. Available: https://arxiv.org/pdf/2411.10915v1
[105] J. Morrison, C. Na, J. Fernandez, T. Dettmers, E. Strubell, and J. Dodge, “Holistically Evaluating the Environmental Impact of Creating Language Models,” Mar. 2025, Accessed: Sep. 22, 2025. [Online]. Available: https://arxiv.org/pdf/2503.05804v1

[106] W. Hsieh *et al.*, “A Comprehensive Guide to Explainable AI: From Classical Models to LLMs,” 2024.
[107] “Chinese room - Wikipedia.” Accessed: Sep. 22, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Chinese_room
[108] P. Lewis *et al.*, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”, Accessed: Sep. 24, 2025. [Online]. Available: https://github.com/huggingface/transformers/blob/master/
[109] “Techniques, Challenges, and Future of Augmented Language Models - Gradient Flow.” Accessed: Sep. 24, 2025. [Online]. Available: https://gradientflow.com/techniques-challenges-and-future-of-augmented-language-models/
[110] “Block diagram of the LangChain ecosystem. [17] | Download Scientific Diagram.” Accessed: Sep. 24, 2025. [Online]. Available: https://www.researchgate.net/figure/Block-diagram-of-the-LangChain-ecosystem-17_fig2_379507857
[111] “What is Retrieval Augmented Generation, and How Can You Use It? | by Niall McNulty | Medium.” Accessed: Sep. 24, 2025. [Online]. Available: https://medium.com/@niall.mcnulty/what-is-retrieval-augmented-generation-and-how-can-you-use-it-d5db3169dc3a
[112] “Streamlit documentation.” Accessed: Sep. 27, 2025. [Online]. Available: https://docs.streamlit.io/