



Σχολή Θετικών Επιστημών και Τεχνολογίας
Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά Συστήματα
(MSc)

Μεταπτυχιακή Διπλωματική Εργασία
Εφαρμογή Μεγάλων Γλωσσικών Μοντέλων και Μηχανικής
Μάθησης στη Βελτιστοποίηση και Αυτοματοποίηση
Επιχειρησιακών Διαδικασιών.

Ηλίας Γκόρος

Επιβλέπων καθηγητής: Ανδρέας Καναβός

Πάτρα, Ιούνιος 2026

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή («συγγραφέας/δημιουργός») που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.



Εφαρμογή Μεγάλων Γλωσσικών Μοντέλων και Μηχανικής
Μάθησης στη Βελτιστοποίηση και Αυτοματοποίηση
Επιχειρησιακών Διαδικασιών.

Ηλίας Γκόρος

Επιτροπή Επίβλεψης Μεταπτυχιακής Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:

Ανδρέας Καναβός

Συν-Επιβλέπων Καθηγητής:

Γεώργιος Μαυρομμάτης

Πάτρα, Ιούνιος 2026

Δόξα τω Θεώ πάντων ένεκεν.

Περίληψη

Η εν λόγω διπλωματική εργασία εστιάζει στην εφαρμογή Μεγάλων Γλωσσικών Μοντέλων (Large Language Models – LLMs) και τεχνικών Μηχανικής Μάθησης (Machine Learning – ML) με στόχο τη βελτιστοποίηση και την αυτοματοποίηση επιχειρησιακών διαδικασιών.

Η αυξανόμενη πολυπλοκότητα των επιχειρησιακών δεδομένων, σε συνδυασμό με την ανάγκη για ταχύτερη και ακριβέστερη λήψη αποφάσεων, καθιστούν αναγκαία την αξιοποίηση σύγχρονων τεχνολογιών μέσα σε αυτές και της τεχνητής νοημοσύνης.

Στην παρούσα εργασία, σχεδιάζεται και υλοποιείται ένα υβριδικό σύστημα που συνδυάζει embedding-based μεθόδους, αλγορίθμους μηχανικής μάθησης και LLMs, με εφαρμογή στην αντιστοίχιση περιγραφών προϊόντων, τη σύγκριση δεδομένων προμηθευτών και την υποστήριξη επιχειρησιακών αποφάσεων, όπως η κοστολόγηση και οι παραγγελίες. Η Python χρησιμοποιείται ως κύρια γλώσσα προγραμματισμού, ενώ ο Microsoft SQL Server λειτουργεί ως κεντρικό σύστημα διαχείρισης δεδομένων. Παράλληλα, αξιοποιούνται εργαλεία όπως το Excel για οπτικοποίηση και έλεγχο αποτελεσμάτων, καθώς και θα εξεταστούν και χρησιμοποιηθούν βιβλιοθήκες μηχανικής μάθησης όπως οι scikit-learn, pandas, NumPy, XGBoost και τα νευρωνικά δίκτυα.

Η προτεινόμενη προσέγγιση βασίζεται σε end-to-end ροή δεδομένων, η οποία περιλαμβάνει τη συλλογή δεδομένων μέσω bots και APIs, τον καθαρισμό και την προετοιμασία τους, τον υπολογισμό embeddings, τη χρήση semantic matching με cosine similarity και την ενσωμάτωση κανόνων (rules) και thresholds για τον περιορισμό αβεβαιοτήτων και hallucinations. Επιπλέον, εφαρμόζεται μηχανισμός επαλήθευσης αποτελεσμάτων μέσω LLMs, ενισχύοντας την αξιοπιστία της τελικής απόφασης. Πλέον θα χρησιμοποιηθούν τα LLMs και για έτερες διαδικασίες όπως η κατασκευή ενός Chatbot.

Τα αποτελέσματα δείχνουν ότι η υβριδική προσέγγιση υπερτερεί έναντι των καθαρά ML ή LLM συστημάτων, προσφέροντας μεγαλύτερη ακρίβεια, ανθεκτικότητα στον θόρυβο και βελτιωμένη επιχειρησιακή αποδοτικότητα. Η εργασία καταδεικνύει τη σημαντική συνεισφορά των LLMs της Μηχανικής Μάθησης αλλά και της αυτοματοποίησης διαδικασιών στο σύγχρονο ανταγωνιστικό επιχειρησιακό περιβάλλον.

Λέξεις – Κλειδιά

Μεγάλα Γλωσσικά Μοντέλα, Μηχανική Μάθηση, Semantic Matching, Embeddings, Υβριδικά Συστήματα, Επιχειρησιακή Αυτοματοποίηση.

Application of LLMs and Machine Learning in Optimizing and Automating Business Processes.

Ilias Gkoros

Abstract

This thesis focuses on the application of Large Language Models (LLMs) and Machine Learning (ML) techniques with the aim of optimizing and automating business processes. The increasing complexity of business data, combined with the need for faster and more accurate decision-making, makes the adoption of modern technologies and artificial intelligence within organizations essential.

In this work, a hybrid system is designed and implemented that combines embedding-based methods, machine learning algorithms, and LLMs, applied to product description matching, supplier data comparison, and the support of business decisions such as costing and ordering. Python is used as the main programming language, while Microsoft SQL Server serves as the central data management system. In parallel, tools such as Excel are utilized for data visualization and result validation, and machine learning libraries such as scikit-learn, pandas, NumPy, XGBoost, and neural networks are examined and applied.

The proposed approach is based on an end-to-end data pipeline that includes data collection via bots and APIs, data cleaning and preprocessing, embedding computation, semantic matching using cosine similarity, and the integration of rules and thresholds to reduce uncertainty and hallucinations. Additionally, a verification mechanism using LLMs is implemented to enhance the reliability of the final decision. Furthermore, LLMs are also employed for additional processes, such as the development of a chatbot.

The results indicate that the hybrid approach outperforms purely ML-based or LLM-based systems, offering higher accuracy, increased robustness to noise, and improved operational efficiency. The thesis demonstrates the significant contribution of LLMs, Machine Learning, and process automation within the modern competitive business environment.

Keywords

Large Language Models, Machine Learning, Semantic Matching, Embeddings, Hybrid Systems, Business Process Automation.

Περιεχόμενα

Περίληψη	v
Abstract	vi
Συντομογραφίες & Ακρωνύμια	ix
1. Εισαγωγή	1
1.1 Αντικείμενο της διπλωματικής εργασίας.	2
1.2 Σύγχρονα επιχειρησιακά προβλήματα και ανάγκη αυτοματοποίησης.	3
1.3 Ρόλος των LLMs και της Μηχανικής Μάθησης στη λήψη αποφάσεων.	5
1.3.1 Η Μηχανική Μάθηση ως Πλαίσιο Ποσοτικής Αξιολόγησης και Υπολογισμού Σημασιολογικής Ομοιότητας.	6
1.3.2 Ο Ρόλος των LLMs και ο Κίνδυνος των Ψευδαισθήσεων (Hallucinations).	7
1.3.3 Ο Υβριδικός Χαρακτήρας και η Αρχιτεκτονική RAG.	9
1.3.4 Σφάλματα Τύπου I & II και το Κατώφλι Αποδοχής (Thresholding).	11
1.4 Στόχοι και ερευνητικά ερωτήματα.	13
1.5 Μεθοδολογική προσέγγιση.	14
1.6 Δομή της διπλωματικής εργασίας.	15
2. Θεωρία LLMs, MLs και τεχνολογίες εφαρμογής των μοντέλων:	17
2.1 Μεγάλα Γλωσσικά Μοντέλα (LLMs).	17
2.1.1 Ορισμός και βασικές αρχές λειτουργίας.	19
2.1.2 Ντετερμινιστικός χαρακτήρας και περιορισμοί.	22
2.1.3 Οι Ψευδαισθήσεις (Hallucinations) και Στρατηγικές Ελέγχου.	23
2.1.4 Μηχανική Προτροπών (Prompt Engineering) και Στρατηγικές.	26
2.2 Μηχανική Μάθηση (Machine Learning).	27
2.2.1 Επιβλεπόμενη (Supervised) και Μη Επιβλεπόμενη (Unsupervised) Μάθηση.	29
2.2.2 Ensemble-Hybrid Αλγόριθμοι: Ανθεκτικότητα και Αντιπαράβολή XGBoost με Random Forest.	30
2.2.3 Ο Ρόλος της Μηχανικής Μάθησης σε Επιχειρησιακά Δεδομένα.	32
2.3 Διανυσματικές Αναπαραστάσεις Embeddings και Semantic Matching	32
2.3.1 Αναπαράσταση κειμένου σε διανύσματα (embeddings).	32
2.3.2 Ομοιότητα Συνημιτόνου (Cosine Similarity) και Προκλήσεις.	33
2.3.3 Semantic Matching σε Περιγραφές Προϊόντων και Βάσεις Διανυσμάτων.	33
2.4 Υβριδικά Συστήματα Hybrid Systems: Γέφυρα μεταξύ LLM και ML	35
2.4.1 Συνδυασμός στατιστικής-πιθανοτήτων και ντετερμινιστικών μεθόδων.	35
2.4.2 Πλεονεκτήματα στην Επιχειρησιακή Εφαρμογή.	35
2.4.3 Ρόλος κανόνων (rules) και thresholds.	35
2.5 Τεχνολογίες και βιβλιοθήκες υλοποίησης.	36
2.5.1 Python, ML Frameworks και Πράκτορες (AI Agents).	36
2.5.2 Βάσεις Δεδομένων: Ο Μετασχηματισμός του SQL Server	37
2.5.3 Χρήση Excel ως ενδιάμεσο εργαλείο επιχειρησιακής γνώσης.	38
2.5.4 APIs και διασύνδεση εξωτερικών συστημάτων.	38
3. Υλοποίηση & Εφαρμογή του Προτεινόμενου Συστήματος Αντιστοίχισης Προϊόντων.	40
3.1 Συνολική αρχιτεκτονική συστήματος.	41
3.1.1 Περιγραφή end-to-end ροής δεδομένων. Agents.	43
3.1.2 Αρχιτεκτονική Βασισμένη σε Πράκτορες (Agent-Based Architecture).	45
3.1.3 Αρχιτεκτονική Τριών Σταδίων (Three-Stage Pipeline).	46
3.2 Εφαρμογή Αντιστοίχισης Προϊόντων LLMs.	46
3.2.1 Δημιουργία Εικονικής ΒΔ και Εικονικού Αρχείου Προμηθευτή.	46

3.2.2 Εφαρμογή αντιστοίχισης προϊόντων χωρίς σύνδεση με τη ΒΔ.	51
3.2.3 Εφαρμογή αντιστοίχισης προϊόντων συνδέοντας τη ΒΔ.	61
3.3 Υλοποίηση μέσω Πρακτόρων Agents.	70
3.4 Εφαρμογή Αντιστοίχισης Προϊόντων με Machine Learning.	72
3.4.1 Εισαγωγικό Πρακτικό Παράδειγμα Εκτέλεσης XGBoost. Supervised M.L.	73
3.4.2 Στάδιο Προεκπαίδευσης Μοντέλου Μηχανικής Μάθησης.	74
3.4.3 Αντιστοίχιση προϊόντων με το Μοντέλο Μηχανικής Μάθησης.	82
3.4.4 Επεξήγηση PKL αρχείων.	93
3.5 Εφαρμογή Αντιστοίχισης Προϊόντων με τοπικό LLM.	97
3.5.1 Ollama ως Πλατφόρμα Τοπικής εκτέλεσης LLM.	97
3.5.2 Εφαρμογή και Παραδείγματα Τοπικού LLM.	99
3.6 Ανατροφοδότηση και Συνεχής Βελτίωση.	105
4. Συμπεράσματα Επέκταση και αξιολόγηση συστημάτων:	105
4.1 Αξιολόγηση των Προτεινόμενων Συστημάτων.	105
4.2 Συμπεράσματα Επέκταση και Επιχειρησιακή Αξιοποίηση.	106
4.2.1 Παράδειγμα Επέκτασης και Αξιοποίησης του Συστήματος.	106
4.2.2 Δειγματοληπτικός Έλεγχος και Ποιοτική Αξιολόγηση των Αποτελεσμάτων.	112
4.2.3 Μελλοντική Δυνατότητα αξιοποίησης Frameworks Ενορχήστρωσης LLM. .	116
4.2.4 Μελλοντική Επέκταση με μηχανισμό συνεχούς μάθησης.	117
4.2.5 Πλέον Μελλοντικές Επεκτάσεις – Υλοποιήσεις.	121
4.2.6 Βελτιστοποίηση με χρήση Qdrant Vector Database για Τεράστια Δεδομένα.	122
4.3 Αξιολόγηση των Προτεινόμενων Συστημάτων.	127
5. Βιβλιογραφία – Παραρτήματα:	128

Συντομογραφίες & Ακρωνύμια

ΔΕ	Διπλωματική Εργασία
ΕΑΠ	Ελληνικό Ανοικτό Πανεπιστήμιο
ΠΣ	Πρόγραμμα Σπουδών
ΣΥΝ	Συντονιστής

1. Εισαγωγή.

Η ταχεία εξέλιξη της Τεχνητής Νοημοσύνης (Artificial Intelligence – AI), και ιδιαίτερα η ανάπτυξη των Μεγάλων Γλωσσικών Μοντέλων (Large Language Models – LLMs), έχει επηρεάσει ουσιαστικά τον τρόπο με τον οποίο οι σύγχρονες επιχειρήσεις διαχειρίζονται και αξιοποιούν τα δεδομένα τους. Η δυνατότητα των LLMs να κατανοούν και να επεξεργάζονται φυσική γλώσσα, σε συνδυασμό με τις τεχνικές της Μηχανικής Μάθησης (Machine Learning – ML), δημιουργεί νέες προοπτικές για την αυτοματοποίηση σύνθετων επιχειρησιακών διαδικασιών. Παράλληλα, η σύγχρονη διεθνής βιβλιογραφία επισημαίνει ότι η παραδοσιακή Ρομποτική Αυτοματοποίηση Διαδικασιών (Robotic Process Automation – RPA), η οποία βασίζεται κυρίως σε προκαθορισμένους κανόνες και γραμμικές συσχετισμούς, εμφανίζει σημαντικούς περιορισμούς όταν καλείται να διαχειριστεί μη δομημένα δεδομένα, ασαφείς πληροφορίες και μεταβολές στη ροή εργασιών, γεγονός που μειώνει την αποδοτικότητά μιας επιχείρησης στο σύγχρονο επιχειρησιακό περιβάλλον.

Συνεπώς, η μετάβαση προς την Ευφυή Αυτοματοποίηση (Intelligent Automation – IA) αναδεικνύεται ως αναγκαία εξέλιξη για τις επιχειρήσεις που λειτουργούν σε ένα διαρκώς μεταβαλλόμενο και ανταγωνιστικό περιβάλλον. Η ταχύτητα λήψης αποφάσεων, η ακρίβεια της πληροφορίας και η δυνατότητα άμεσης αξιοποίησης της διαθέσιμης γνώσης αποτελούν κρίσιμους παράγοντες επιχειρησιακής αποτελεσματικότητας. Ο συνεχώς αυξανόμενος όγκος και η σημασιολογική πολυπλοκότητα των δεδομένων καθιστούν ανεπαρκείς τόσο τις χειροκίνητες διαδικασίες όσο και τις αμιγώς αυτοματοποιημένες που βασίζονται σε συγκεκριμένους κανόνες. Για τον λόγο αυτό, θεωρείται απαραίτητη η ανάπτυξη υβριδικών συστημάτων, τα οποία συνδυάζουν στατιστικές τεχνικές μάθησης με την ευελιξία και την ικανότητα κατανόησης που προσφέρουν τα σύγχρονα γλωσσικά μοντέλα LLMs, επιτρέποντας αποτελεσματικότερη επεξεργασία σύνθετων πληροφοριών, αυτοματοποίηση διαδικασιών και εξαγωγή συμπερασμάτων για τη μετέπειτα λήψη αποφάσεων.

<https://arxiv.org/abs/2412.18063>

(Large Language Model-Driven Efficient Robotic Process Automation for OCR)

1.1 Αντικείμενο της διπλωματικής εργασίας.

Το αντικείμενο της παρούσας διπλωματικής εργασίας είναι η μελέτη, ο σχεδιασμός και η πρακτική υλοποίηση ενός υβριδικού «ευφυούς» συστήματος, το οποίο συνδυάζει Μεγάλα Γλωσσικά Μοντέλα (LLMs) με τεχνικές Μηχανικής Μάθησης (ML), με στόχο την αποτελεσματική αυτοματοποίηση επιχειρησιακών διαδικασιών. Η αποκλειστική χρήση LLMs σε επιχειρησιακά περιβάλλοντα ενδέχεται να παρουσιάσει προβλήματα αξιοπιστίας, λόγω της φύσης των παραγόμενων αποκρίσεων που στηρίζονται σε στοχαστικά μοντέλα πιθανοτήτων. Για τον λόγο αυτό, αυξάνεται το ερευνητικό ενδιαφέρον προς υβριδικές αρχιτεκτονικές, στις οποίες τα γλωσσικά μοντέλα συνεργάζονται με ντετερμινιστικούς μηχανισμούς ελέγχου και σαφώς ορισμένα λειτουργικά όρια, προκειμένου να επιτυγχάνεται μεγαλύτερη συνέπεια.

Η παρούσα ερευνητική προσέγγιση εστιάζει ιδιαίτερα στην αξιοποίηση μεθόδων που βασίζονται σε διανυσματικές αναπαραστάσεις κειμένου (embeddings) και τεχνικές σημασιολογικής αντιστοίχισης (semantic matching). Τα σύγχρονα embedding models, έχουν αποδειχθεί ιδιαίτερα αποτελεσματικά στην ανίχνευση σημασιολογικών ομοιοτήτων μεταξύ ετερογενών και μη δομημένων text δεδομένων, ακόμη και όταν υπάρχουν διαφοροποιήσεις στη σύνταξη, την ορολογία ή την πληρότητα των περιγραφών. Στο πλαίσιο της παρούσας εργασίας, οι τεχνικές αυτές αξιοποιούνται για την αντιστοίχιση περιγραφών προϊόντων, τη συγκριτική επεξεργασία δεδομένων και γενικότερα την υποστήριξη επιχειρησιακών διαδικασιών.

Παράλληλα, εξετάζεται ο ρόλος των Μεγάλων Γλωσσικών Μοντέλων ως μηχανισμών κατανόησης φυσικής γλώσσας, ερμηνείας των αποτελεσμάτων και τελικής σημασιολογικής επαλήθευσης. Εφαρμόζεται ενορχηστρωμένο πλαίσιο συνεργαζόμενων υποσυστημάτων, στο οποίο οι λειτουργίες κατανόησης, πρόβλεψης και ελέγχου κατανέμονται σε διακριτά επίπεδα επεξεργασίας. Η συνεργασία μεταξύ της σημασιολογικής ικανότητας των LLMs και της υπολογιστικής ακρίβειας των αλγορίθμων Μηχανικής Μάθησης δημιουργεί ένα υβριδικό πλαίσιο που μειώνει το επιχειρησιακό ρίσκο και αυξάνει την αξιοπιστία των αποτελεσμάτων.

https://www.researchgate.net/publication/400558545_Agent-based_Hybrid_AI_Models_and_Technologies_A_systematic_Literature_Review

(Agent-based Hybrid AI Models and Technologies: A Systematic Literature Review).

1.2 Σύγχρονα επιχειρησιακά προβλήματα και ανάγκη αυτοματοποίησης.

Στο σύγχρονο ψηφιοποιημένο επιχειρησιακό περιβάλλον, οι οργανισμοί καλούνται να συλλέγουν, να επεξεργάζονται και να διαχειρίζονται καθημερινά μεγάλους όγκους δεδομένων που προέρχονται από πολλαπλές και ετερογενείς πηγές. Οι πηγές αυτές μπορεί να προέρχονται από σχεσιακές ή μη βάσεις δεδομένων, αρχεία xlsx, csv, xml, κ.α., εξωτερικά συστήματα διασύνδεσης μέσω APIs, καθώς και πληροφοριακά συστήματα ERP. Σημαντικό μέρος της πληροφορίας αυτής δεν βρίσκεται σε πλήρως δομημένη μορφή. Η χειροκίνητη ερμηνεία και μεταφορά των δεδομένων αυτών σε δομημένες βάσεις αποτελεί μια χρονοβόρα διαδικασία με αυξημένη πιθανότητα σφαλμάτων και περιορισμένη επιχειρησιακή αποδοτικότητα.

(Large Language Model-Driven Efficient Robotic Process Automation for OCR).

Ένα ιδιαίτερα χαρακτηριστικό πρόβλημα στο πλαίσιο αυτό είναι η αυτοματοποιημένη αντιστοίχιση περιγραφών προϊόντων (product matching ή record linkage) μεταξύ διαφορετικών εμπορικών καταλόγων. Στην πράξη, το ίδιο φυσικό προϊόν είναι δυνατόν να εμφανίζεται με διαφορετικές περιγραφές, εναλλακτικές συντομογραφίες, παραλλαγές κωδικών παραγωγού, διαφορετικές μονάδες μέτρησης ή ακόμη και διαφορετική γλώσσα καταγραφής. Οι παραδοσιακές μέθοδοι λεξιλογικής αναζήτησης, οι οποίες βασίζονται κυρίως σε άμεση ή μερική ταύτιση λέξεων-κλειδιών, αδυνατούν να εντοπίσουν με συνέπεια τις υποκείμενες σημασιολογικές ομοιότητες μεταξύ τέτοιων εγγραφών. Αυτό έχει ως αποτέλεσμα τη δημιουργία διπλοεγγραφών, αστοχίες στην τιμολογιακή σύγκριση και μειωμένη δυνατότητα ενιαίας αξιολόγησης δεδομένων από πολλαπλούς προμηθευτές.

<https://www.techscience.com/cmc/v71n1/45385>

(Entity Matching using Sentence-BERT and Transformer-based Similarity Encoding).

Η μετάβαση από την επιφανειακή ομοιότητα λέξεων στη βαθιά σημασιολογική συγγένεια επιτυγχάνεται μέσω διανυσματικών αναπαραστάσεων, όπου η μαθηματική απόσταση των δεδομένων αντικατοπτρίζει την εννοιολογική τους συνάφεια. Με τον τρόπο αυτό καθίσταται εφικτός ο υπολογισμός της πραγματικής σημασιολογικής συνάφειας μεταξύ δύο φαινομενικά διαφορετικών περιγραφών-κειμένων. Μοντέλα όπως το Sentence-BERT και η οικογένεια MiniLM έχουν αποδειχθεί ιδιαίτερα αποτελεσματικά σε εργασίες semantic similarity και entity matching, καθώς επιτρέπουν ακριβέστερη αναγνώριση σχετικών

εγγραφών σε περιβάλλοντα ετερογενούς πληροφορίας (Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks). Σε συνδυασμό με επιβλεπόμενους αλγορίθμους μηχανικής μάθησης, όπως οι ensemble classifiers, καθίσταται δυνατή η ανάπτυξη ενός αυτοματοποιημένου πλαισίου ανίχνευσης ταυτόσημων ή ισχυρά συσχετισμένων προϊόντων με αυξημένο επίπεδο αξιοπιστίας.

(Agent-based Hybrid AI Models and Technologies: A Systematic Literature Review).

Πίνακας 1: Σύγκριση δυνατοτήτων και περιορισμών των συστημάτων Λεξιλογικής και Σημασιολογικής Ανάκτησης Πληροφορίας.

<i>Τεχνικό Χαρακτηριστικό</i>	<i>Παραδοσιακή Λεξιλογική Ανάκτηση (Lexical Retrieval)</i>	<i>Σημασιολογική Ανάκτηση μέσω Embeddings (Semantic Retrieval)</i>
Βασικός Μηχανισμός λειτουργίας	Ακριβής ή στατιστική μερική ταύτιση όρων (π.χ. αλγόριθμοι TF-IDF, BM25)	Υπολογισμός ομοιότητας (π.χ. Cosine Similarity) σε πολυδιάστατο διανυσματικό χώρο
Διαχείριση Συνωνύμων και Εννοιών	Σχεδόν αδύνατη χωρίς την εκ των προτέρων κατασκευή και συντήρηση ογκωδών λεξικών	Αυτόματη αναγνώριση εννοιών μέσω της προεκπαίδευσης του μοντέλου σε τεράστια σώματα κειμένων
Διαχείριση Πολυγλωσσικότητας (Multilingualism)	Απαιτεί τη χρήση ξεχωριστών, εξωτερικών μοντέλων μετάφρασης πριν την αναζήτηση	Εφικτή εγγενώς μέσω μοντέλων Multilingual Embeddings (π.χ. LaBSE, XLM-R)
Ανοχή σε Σφάλματα Πληκτρολόγησης (Typos)	Εξαιρετικά χαμηλή (απαιτεί την προσθήκη δαπανηρών αλγορίθμων απόστασης τύπου Levenshtein)	Πολύ υψηλή (τα σύγχρονα λεκτικά μοντέλα tokenization κατανοούν το ευρύτερο πλαίσιο της πρότασης)
Υποστήριξη Πολυτροπικότητας (Multimodality)	Περιορίζεται αυστηρά σε αναζήτηση κειμένου έναντι κειμένου	Προσφέρει δυναμική ικανότητα ταυτόχρονου συνδυασμού κειμένου, μεταδεδομένων και εικόνας (Multimodal representations)

Η ενσωμάτωση τεχνικών σημασιολογικής ανάκτησης σε επιχειρησιακές διαδικασίες δεν συμβάλλει μόνο στη μείωση του λειτουργικού φόρτου που σχετίζεται με τη χειροκίνητη επεξεργασία δεδομένων, αλλά επιτρέπει και την αποτελεσματικότερη αξιοποίηση της διαθέσιμης πληροφορίας. Μέσω της αυτοματοποιημένης ενοποίησης ετερογενών δεδομένων, οι επιχειρήσεις αποκτούν ταχύτερη πρόσβαση σε συγκρίσιμα και δομημένα

αποτελέσματα, γεγονός που βελτιώνει την ταχύτητα απόκρισης σε μεταβολές της αγοράς και ενισχύει τη διαδικασία λήψης αποφάσεων. Παράλληλα, μειώνεται η ανάγκη απασχόλησης ανθρώπινων πόρων σε επαναλαμβανόμενες εργασίες χαμηλής αναλυτικής αξίας δίνοντας την ευκαιρία σε ουσιαστικότερες εργασίες.

(Large Language Model-Driven Efficient Robotic Process Automation for OCR; Agent-based Hybrid AI Models and Technologies: A Systematic Literature Review).

1.3 Ρόλος των LLMs και της Μηχανικής Μάθησης στη λήψη αποφάσεων.

Η διαδικασία λήψης αποφάσεων στις σύγχρονες επιχειρήσεις είναι ιδιαίτερα απαιτητική, καθώς χαρακτηρίζεται από υψηλή πολυπλοκότητα, αυξημένο όγκο δεδομένων και ανάγκη για άμεση απόκριση. Οι αποφάσεις δεν βασίζονται πλέον μόνο σε εμπειρικά κριτήρια ή στην ανθρώπινη διαίσθηση, αλλά στη συστηματική αξιοποίηση δεδομένων και στην εφαρμογή υπολογιστικών μοντέλων ανάλυσης.

Στο πλαίσιο αυτό, η Μηχανική Μάθηση (Machine Learning – ML) και τα Μεγάλα Γλωσσικά Μοντέλα (Large Language Models – LLMs) αποτελούν δύο από τις σημαντικότερες τεχνολογικές συνιστώσες των σύγχρονων ευφυών επιχειρησιακών συστημάτων. Η Μηχανική Μάθηση προσφέρει μεθοδολογίες για την ανάλυση ιστορικών δεδομένων, τον εντοπισμό προτύπων και τη δημιουργία προβλεπτικών μοντέλων, ενώ τα Μεγάλα Γλωσσικά Μοντέλα συμβάλλουν στην κατανόηση και επεξεργασία μη δομημένης γλωσσικής πληροφορίας.

Η συνδυαστική αξιοποίηση των δύο αυτών τεχνολογιών οδηγεί στην ανάπτυξη υβριδικών συστημάτων λήψης αποφάσεων, στα οποία η ποσοτική στατιστική αξιολόγηση συνδυάζεται με τη σημασιολογική κατανόηση. Στα συστήματα αυτά, τα μοντέλα ML αναλαμβάνουν την ταξινόμηση, την εκτίμηση πιθανοτήτων και την ποσοτική αξιολόγηση των δεδομένων, ενώ τα LLMs υποστηρίζουν την ερμηνεία φυσικής γλώσσας, τη σύνοψη πληροφορίας και την αναγνώριση σημασιολογικών σχέσεων μεταξύ ετερογενών περιγραφών.

Παράλληλα, η αποκλειστική χρήση ενός μόνο μοντέλου τεχνητής νοημοσύνης δεν θεωρείται επαρκής για κρίσιμα επιχειρησιακά περιβάλλοντα. Τα Μεγάλα Γλωσσικά Μοντέλα λειτουργούν βάσει θεωρίας πιθανοτήτων και μηχανισμών στατιστικής πρόβλεψης και συνεπώς ενδέχεται να παράγουν ανακριβείς ή μη επαληθεύσιμες απαντήσεις. Για τον

λόγο αυτό, η σύγχρονη βιβλιογραφία προκρίνει υβριδικές αρχιτεκτονικές στις οποίες η λειτουργία των LLMs πλαισιώνεται από ντετερμινιστικούς κανόνες, κατώφλια αποδοχής (thresholds) και μηχανισμούς επαλήθευσης.

Μέσω αυτής της πολυεπίπεδης προσέγγισης, η λήψη αποφάσεων δεν βασίζεται ούτε αποκλειστικά στις στατιστικές πιθανότητες ούτε αποκλειστικά στη γλωσσική ερμηνεία, αλλά σε ένα ελεγχόμενο σύστημα συνδυαστικής αξιολόγησης.

Εφαρμόζοντας παρόμοια υβριδικά μοντέλα μπορούμε να έχουμε: ταχύτερη αξιολόγηση προμηθευτών, αυτοματοποιημένη αντιστοίχιση προϊόντων, υποστήριξη κοστολόγησης, μείωση ανθρώπινων σφαλμάτων και γενικότερα ενίσχυση του ανταγωνιστικού πλεονεκτήματος.

1.3.1 Η Μηχανική Μάθηση ως Πλαίσιο Ποσοτικής Αξιολόγησης και Υπολογισμού Σημασιολογικής Ομοιότητας.

Η λήψη αποφάσεων υπό καθεστώς αβεβαιότητας αποτελεί αντικείμενο της Θεωρίας Αποφάσεων (Decision Theory), κάθε επιλογή είναι αβέβαια και εκτιμάται μέσω Πιθανοτήτων. Η απόφαση βασίζεται στη μεγιστοποίηση της αναμενόμενης χρησιμότητας (expected utility), που δίνει το μεγαλύτερο αναμενόμενο όφελος λαμβάνοντας υπόψη τόσο την πιθανότητα επιτυχίας ή αποτυχίας και το κόστος σφάλματος.

Μαθηματικά, η αναμενόμενη χρησιμότητα εκφράζεται ως:

$$EU = \sum p_i \cdot U(x_i)$$

όπου p_i είναι η πιθανότητα εμφάνισης ενός αποτελέσματος x_i και $U(x_i)$ η αντίστοιχη επιχειρησιακή του χρησιμότητα.

Η Μηχανική Μάθηση ενσωματώνει πρακτικά αυτή τη φιλοσοφία, καθώς τα μοντέλα ταξινόμησης και πρόβλεψης βασίζονται σε εκτίμηση πιθανοτήτων για κάθε πιθανό αποτέλεσμα. Για παράδειγμα, σε ένα πρόβλημα αντιστοίχισης προϊόντων, το σύστημα δεν επιστρέφει απλώς μία δυαδική απάντηση τάυτισης ή μη τάυτισης, αλλά μια ποσοτική πιθανότητα ότι δύο περιγραφές αναφέρονται στο ίδιο αντικείμενο. Η τελική απόφαση λαμβάνεται όταν η πιθανότητα αυτή υπερβεί ένα προκαθορισμένο κατώφλι αποδοχής, το οποίο αντανakλά το επιτρεπτό επίπεδο επιχειρησιακού ρίσκου.

Παράλληλα, η Θεωρία Πιθανοτήτων και ειδικότερα το Bayesian reasoning προσφέρει ένα επιπλέον μαθηματικό θεμέλιο. Σύμφωνα με το θεώρημα του Bayes:

$$P(H|D) = [P(D|H) \cdot P(H)] / P(D)$$

όπου H είναι μία υπόθεση, D τα παρατηρούμενα δεδομένα, $P(H)$ η εκ των προτέρων πιθανότητα και $P(H|D)$ η εκ των υστέρων πιθανότητα μετά την νέα πληροφορία.

Στα επιχειρησιακά συστήματα ML, τα μοντέλα ουσιαστικά επιχειρούν να εκτιμήσουν το $P(H|D)$, δηλαδή την πιθανότητα ορθότητας μιας υπόθεσης με βάση τα διαθέσιμα δεδομένα.

Η λογική αυτή είναι ιδιαίτερα χρήσιμη σε δυναμικά περιβάλλοντα, όπου νέες πληροφορίες μπορούν να μεταβάλουν την αρχική εκτίμηση του συστήματος.

Στην πράξη, η επιλογή του κατάλληλου threshold αποτελεί στρατηγικής σημασίας επιχειρησιακή απόφαση, καθώς συνδέεται άμεσα με δύο τύπους πιθανών σφαλμάτων: το ψευδώς θετικό αποτέλεσμα (Type I error) και το ψευδώς αρνητικό αποτέλεσμα (Type II error). Σε ένα σύστημα αντιστοίχισης προϊόντων, ένα ψευδώς θετικό ταίριασμα μπορεί να οδηγήσει σε λανθασμένη κοστολόγηση ή εσφαλμένη ταύτιση εγγραφών-κειμένου, ενώ ένα ψευδώς αρνητικό αποτέλεσμα μπορεί να συνεπάγεται απώλεια επιχειρηματικής ευκαιρίας.

Στο πλαίσιο της παρούσας εργασίας, η ποσοτική αυτή αξιολόγηση υλοποιείται μέσω τεχνικών υπολογισμού σημασιολογικής ομοιότητας. Οι περιγραφές προϊόντων μετατρέπονται σε πυκνές διανυσματικές αναπαραστάσεις (dense text embeddings) και συγκρίνονται σε πολυδιάστατο μαθηματικό χώρο μέσω μετρικών όπως η ομοιότητα συνημίτονου (Cosine Similarity). Με τον τρόπο αυτό, το σύστημα αποκτά τη δυνατότητα να εκτιμά όχι μόνο λεξιλογικές αλλά και βαθύτερες σημασιολογικές ομοιότητες μεταξύ ετερογενών περιγραφών.

Η παραγόμενη αριθμητική βαθμολογία ομοιότητας λειτουργεί ως το βασικό στατιστικό επίπεδο λήψης απόφασης, πάνω στο οποίο στη συνέχεια μπορούν να εφαρμοστούν συμπληρωματικοί σημασιολογικοί ή γλωσσικοί μηχανισμοί επαλήθευσης.

<https://link.springer.com/article/10.1007/s11280-022-01004-7>

(bi-directional Bayesian probabilistic model based hybrid grained semantic matchmaking for Web service discovery)

1.3.2 Ο Ρόλος των LLMs και ο Κίνδυνος των Ψευδαισθήσεων (Hallucinations).

Η έλευση των Μεγάλων Γλωσσικών Μοντέλων (LLMs) άλλαξε ριζικά τον τρόπο διαχείρισης της μη δομημένης πληροφορίας από τα πληροφοριακά συστήματα. Ενώ μέχρι πρόσφατα η Μηχανική Μάθηση περιοριζόταν κυρίως σε δομημένα δεδομένα, τα LLMs μας

επιτρέπουν να «αποκωδικοποιήσουμε» και το νόημα που κρύβεται σε τεχνικές περιγραφές, αναφορές και κάθε είδους επιχειρησιακά έγγραφα που δεν ακολουθούν μια αυστηρή φόρμα ή μορφή. Μέσω μηχανισμών βαθιάς γλωσσικής αναπαράστασης μπορούν να εντοπίζουν σημασιολογικές συσχετίσεις, να συνοψίζουν πληροφορίες και να υποστηρίζουν διαδικασίες εξαγωγής γνώσης από μεγάλα κείμενα.

Επιχειρησιακά μπορούν να συμβάλλουν ενεργά όπου η πληροφορία δεν εμφανίζεται με ενιαία μορφή, αλλά χαρακτηρίζεται από ασάφεια, συντομογραφίες, ελλιπή τεχνικά χαρακτηριστικά και διαφοροποιήσεις ορολογίας παράδειγμα μεταξύ των προμηθευτών μιας επιχείρησης. Σε τέτοιες περιπτώσεις, τα LLMs μπορούν να λειτουργήσουν ως σημασιολογικός μεταφραστής μεταξύ ετερογενών πηγών δεδομένων, διευκολύνοντας την κατανόηση του πραγματικού περιεχομένου μιας περιγραφής.

Ωστόσο, παρά τις σημαντικές δυνατότητές τους, τα μοντέλα αυτά δεν λειτουργούν ντετερμινιστικά ούτε βασίζονται σε ενσωματωμένο μηχανισμό λογικής επαλήθευσης. Η παραγωγή απάντησης στηρίζεται σε θεωρίες πιθανοτήτων και έτσι γίνεται πρόβλεψη γλωσσικής συνέχειας με βάση το παρεχόμενο συμφραζόμενο πλαίσια. Ως αποτέλεσμα, είναι δυνατόν να παραχθούν πληροφορίες που εμφανίζονται γλωσσικά συνεκτικές αλλά δεν ανταποκρίνονται στην πραγματικότητα ή δεν τεκμηριώνονται από τα διαθέσιμα δεδομένα. Το φαινόμενο αυτό αναφέρεται στη βιβλιογραφία ως hallucinations και αποτελεί έναν από τους βασικότερους περιορισμούς των LLMs σε εφαρμογές υψηλής επιχειρησιακής αξιοπιστίας.

Οι ψευδαισθήσεις αυτές μπορεί να αφορούν είτε λανθασμένη ερμηνεία υπαρκτών δεδομένων είτε παραγωγή ανύπαρκτων συσχετίσεων και μη τεκμηριωμένων ισχυρισμών. Σε συστήματα που σχετίζονται με τιμολόγηση, συμμόρφωση, χρηματοοικονομική ανάλυση ή αυτοματοποιημένη αντιστοίχιση προϊόντων, η άμεση και ανεξέλεγκτη αποδοχή τέτοιων απαντήσεων δημιουργεί σημαντικό λειτουργικό κίνδυνο.

Για τον λόγο αυτό, η σύγχρονη ερευνητική προσέγγιση δεν αντιμετωπίζει τα LLMs ως μηχανισμούς τελικής απόφασης, αλλά ως συμβουλευτικά συστήματα γλωσσικής κατανόησης που απαιτούν εξωτερική επαλήθευση από διάφορα ντετερμινιστικά μοντέλα ή ανθρώπους.

https://journals.ekb.eg/article_406701_7e2e276640a5060a8d512dc3150c7b25.pdf
(Hallucination Mitigation Techniques in Large Language Models)

Πίνακας 1. Κατηγοριοποίηση βασικών μεθόδων ελέγχου και διασφάλισης αξιοπιστίας των LLM's.

Τύπος Ελέγχου LLM.	Στρατηγικές Μετριασμού Κινδύνου (Mitigation Strategies).	Περιγραφή και Εφαρμογή.
Έλεγχοι κατά την Ανάπτυξη / Εκπαίδευση (Development Time Controls).	Ενσωμάτωση Γράφων Γνώσης (Knowledge Graph Integration).	Χρήση δομημένων δικτύων γνώσης κατά την εκπαίδευση ή προσαρμογή του μοντέλου ώστε οι παραγόμενες απαντήσεις να συσχετίζονται με επαληθεύσιμα επιχειρησιακά δεδομένα.
Έλεγχοι κατά την Ανάπτυξη & Εκπαίδευση (Development Time Controls).	Βελτιστοποίηση με βάση την πραγματικότητα. (Faithfulness-Oriented Objectives).	Βελτιστοποίηση ώστε να μειώνεται η παραγωγή μη τεκμηριωμένων ή εσωτερικά εσφαλμένων αποτελεσμάτων. Χρήση συναρτήσεων κόστους κατά τη διάρκεια της εκπαίδευσης. (DRESS/RHO).
Έλεγχοι κατά την Παραγωγή και Συμπεράσματα. (Inference Time Controls)	Επαυξημένη Παραγωγή Μέσω Ανάκτησης (Retrieval-Augmented Generation - RAG)	Το μοντέλο αντλεί πριν από την οποία απάντηση πληροφορίες από ενημερωμένη εταιρική βάση δεδομένων, έγγραφα ή αποθετήρια γνώσης, περιορίζοντας την εξάρτηση από την παραμετρική του μνήμη.
Πλέον Έλεγχοι κατά την Παραγωγή και Συμπεράσματα. (Inference Time Controls)	Μηχανισμοί Επίλυσης και Αυτοελέγχου (Self-Refinement and Critique)	Πλέον αξιολόγηση της παραγόμενης απάντησης μέσω βήματος ελέγχου για εντοπισμό λογικών αντιφάσεων ή αδύναμων τεκμηριώσεων. Συστηματική αξιοποίηση καθοδηγούμενων ερωτημάτων τα οποία αναγκάζουν το ίδιο το μοντέλο να επανεξετάσει και να επικυρώσει την ίδια του την απάντηση πριν την παραδώσει, εντοπίζοντας σφάλματα.
Λειτουργικοί / Διαδικαστικοί Έλεγχοι Συστήματος. (Operational Controls).	Πολιτικές Κατωφλίου Εμπιστοσύνης. Πολιτικές Αποχής. (Abstention Policies).	Εφαρμογή confidence thresholds ώστε απαντήσεις χαμηλής αξιοπιστίας να μη γίνονται αυτόματα αποδεκτές αλλά να παραπέμπονται σε ανθρώπινο έλεγχο.

<https://aclanthology.org/2025.acl-srw.53/>

(Mitigating Hallucination by Integrating Knowledge Graphs into LLM)

1.3.3 Ο Υβριδικός Χαρακτήρας και η Αρχιτεκτονική RAG.

Προκειμένου να περιοριστούν οι όποιοι κίνδυνοι των Μεγάλων Γλωσσικών Μοντέλων, η προτείνεται η υιοθέτηση υβριδικών αρχιτεκτονικών. (hybrid orchestrated architectures), στις οποίες τα LLMs δεν λειτουργούν απομονωμένα αλλά ενσωματώνονται σε ένα ευρύτερο περιβάλλον ελεγχόμενων υπολογιστικών μηχανισμών.

Το σύστημα Μηχανικής Μάθησης ή οι κανόνες λογικού ελέγχου και οι σχετικοί αλγόριθμοι, αναλαμβάνουν τις κρίσιμες διεργασίες σε πραγματικό χρόνο, δηλαδή λειτουργίες που σχετίζονται με υπολογισμό πιθανοτήτων, thresholds, ταξινόμηση και έλεγχο εγκυρότητας αποτελεσμάτων, κ.ο.κ. Οι διεργασίες αυτές απαιτούν σταθερή συμπεριφορά, επαναληψιμότητα και χαμηλό χρόνο απόκρισης. Αντίθετα, το LLM τοποθετείται σε συμβουλευτικό και σημασιολογικό επίπεδο, όπου χρησιμοποιείται για κατανόηση φυσικής γλώσσας, ερμηνεία αδόμητων περιγραφών, γλωσσική σύνοψη και δημιουργία προτάσεων

προς αξιολόγηση. Με αυτόν τον λειτουργικό διαχωρισμό, η τελική επιχειρησιακή απόφαση δεν λαμβάνεται μόνο από το LLM, αλλά κυρίως από το εποπτεύον ντετερμινιστικό σύστημα.

Συνεπώς η υβριδική αρχιτεκτονική βασίζεται σε τρία διακριτά επίπεδα λειτουργίας.

- Το πρώτο επίπεδο αφορά την ποσοτική στατιστική αξιολόγηση μέσω ML, όπου παράγονται scores ομοιότητας ή πιθανότητες απόφασης.
- Το δεύτερο επίπεδο αφορά τη σημασιολογική ερμηνεία μέσω LLMs, όπου αναλύονται μη δομημένα δεδομένα και εξάγονται νοηματικές συσχετίσεις.
- Το τρίτο επίπεδο είναι το επίπεδο ελέγχου, στο οποίο εφαρμόζονται προκαθορισμένοι κανόνες αποδοχής, όρια εμπιστοσύνης και διαδικασίες επικύρωσης πριν την οριστική ενέργεια του συστήματος.

Η λογική αυτή επιτρέπει στο σύστημα να αξιοποιεί τα πλεονεκτήματα των γλωσσικών μοντέλων χωρίς να εκθέτει την επιχείρηση σε ανεξέλεγκτα errors ή hallucinations.

Κεντρικό ρόλο στην παραπάνω υβριδική λογική κατέχει η αρχιτεκτονική Επαυξημένης Παραγωγής Μέσω Ανάκτησης (Retrieval-Augmented Generation – RAG).

Η αρχιτεκτονική αυτή αναπτύχθηκε για να αντιμετωπίσει τον περιορισμό της στατικής γνώσης των προεκπαιδευμένων γλωσσικών μοντέλων, τα οποία αδυνατούν να γνωρίζουν δεδομένα που δημιουργήθηκαν μετά το πέρας της εκπαίδευσής τους ή πληροφορίες που βρίσκονται αποκλειστικά εντός των εσωτερικών συστημάτων μιας επιχείρησης.

Στο RAG, κάθε εισερχόμενο ερώτημα μετατρέπεται αρχικά σε διανυσματική αναπαράσταση (embedding), ακολουθώντας διανυσματική αναζήτηση σε εταιρικές βάσεις δεδομένων ή αποθετήρια γνώσης. Τα ανακτηθέντα αυτά δεδομένα τροφοδοτούν στη συνέχεια το LLM ως εξωτερικό πλαίσιο αναφοράς (context grounding), με αποτέλεσμα η παραγόμενη απάντηση να βασίζεται σε πραγματικά και επικαιροποιημένα στοιχεία και όχι αποκλειστικά στην παραμετρική μνήμη του μοντέλου.

Η πρακτική σημασία του RAG στο επιχειρησιακό περιβάλλον είναι ιδιαίτερα υψηλή, διότι επιτρέπει την ασφαλή διασύνδεση του γλωσσικού μοντέλου με ERP, βάσεις δεδομένων, τεχνικά έγγραφα, λίστες προϊόντων, συμβάσεις ή ενημερωμένα αρχεία προμηθευτών.

Με τον τρόπο αυτό το LLM παύει να λειτουργεί ως γενικός παραγωγός απαντήσεων και μετατρέπεται σε μηχανισμό γλωσσικής αξιοποίησης εσωτερικής πληροφορίας.

Η διαδικασία αυτή μειώνει το ποσοστό ψευδαισθήσεων, καθώς θεωρητικά κάθε απάντηση-output μπορεί να συνδεθεί με συγκεκριμένες πηγές δεδομένων του οργανισμού.

Για τον λόγο αυτό, η σύγχρονη ερευνητική κατεύθυνση δεν αντιμετωπίζει πλέον τα LLMs ως αυτόνομους λήπτες αποφάσεων, αλλά ως σημασιολογικά υποσυστήματα εντός ενός αυστηρά ελεγχόμενου συστήματος.

Η επιχειρησιακή αξιοπιστία προκύπτει από τη συνεργασία τεσσάρων πυλώνων:

Της στατιστικής πρόβλεψης μέσω ML.

Της γλωσσικής κατανόησης μέσω LLM.

Της ανάκτησης τεκμηριωμένης γνώσης μέσω RAG.

Και της τελικής επικύρωσης μέσω deterministic rules.

Επομένως, η υβριδική αρχιτεκτονική δεν αποτελεί απλώς τεχνική επιλογή αλλά αναγκαία συνθήκη για την ασφαλή ενσωμάτωση της γενετικής τεχνητής νοημοσύνης στις σύγχρονες επιχειρησιακές διαδικασίες.

<https://arxiv.org/abs/2603.06669>

(Hybrid Orchestration of Edge AI and Microservices)

1.3.4 Σφάλματα Τύπου I & II και το Κατώφλι Αποδοχής (Thresholding).

Η επιτυχία της αυτοματοποίησης μέσω της ανάπτυξης υβριδικών μοντέλων τεχνητής νοημοσύνης κρίνεται, εν τέλει, αποκλειστικά σε πραγματικές συνθήκες.

Όταν για παράδειγμα το σύστημα εξαγάγει το συμπέρασμα και αναφέρει ότι δύο διαφορετικά προϊόντα είναι επί ίδια (διατυπώνοντας δηλαδή μια Θετική Πρόβλεψη) ενώ στην πραγματικότητα διαφέρουν, τότε έχουμε το:

- Σφάλμα Τύπου I (Ψευδώς Θετικό - False Positive - FP).

Αντίστοιχα, όταν το σύστημα αποτυγχάνει πλήρως να αναγνωρίσει και να συνδέσει δύο ίδια προϊόντα, λόγω χρήσης διαφορετικού λεξιλογίου, τότε έχουμε:

- Σφάλμα Τύπου II (Ψευδώς Αρνητικό - False Negative - FN).

Πίνακας 2. Κατηγοριοποίηση αποτελεσμάτων πρόβλεψης και ανάλυση σφαλμάτων (Τύπου I και II).

Πραγματικότητα εις το Περιβάλλον \ Πρόβλεψη Υβριδικού Συστήματος	Θετική Πρόβλεψη (Το σύστημα αποφαινεται: Ταυτίζονται)	Αρνητική Πρόβλεψη (Το σύστημα αποφαινεται: Δεν ταυτίζονται)
Θετική Πραγματικότητα (Τα αντικείμενα όντως Ταυτίζονται)	Αληθώς Θετικό (True Positive - TP)	Ψευδώς Αρνητικό (False Negative - FN) - Σφάλμα Τύπου II

Αρνητική Πραγματικότητα (Τα αντικείμενα Δεν ταυτίζονται)	Ψευδώς Θετικό (False Positive - FP) - Σφάλμα Τύπου I	Αληθώς Αρνητικό (True Negative - TN)
--	--	--------------------------------------

Από τον άνωθεν πίνακα 3, προκύπτουν η Ακρίβεια (Precision) και η Ανάκληση (Recall). Η Ακρίβεια (Precision) υπολογίζεται ως:

$$\text{Precision} = \frac{TP}{TP+FP}$$

εκφράζει το ποσοστό των θετικών προβλέψεων του συστήματος που είναι πράγματι σωστές. Σε επιχειρησιακά περιβάλλοντα όπου μια λανθασμένη αντιστοίχιση μπορεί να οδηγήσει σε αλλοίωση εταιρικών δεδομένων, σε λανθασμένη κοστολόγηση ή σε λανθασμένες αυτοματοποιημένες ενέργειες, η διατήρηση υψηλής ακρίβειας θεωρείται ιδιαίτερα σημαντική.

Η Ανάκληση (Recall) υπολογίζεται ως:

$$\text{Recall} = \frac{TP}{TP+FN}$$

και δείχνει το ποσοστό των πραγματικών ταυτίσεων που κατάφερε να εντοπίσει το σύστημα. Υψηλή ανάκληση συνεπάγεται ότι το μοντέλο δεν παραλείπει μεγάλο αριθμό έγκυρων αντιστοιχίσεων, μειώνοντας έτσι την ανάγκη για εκτεταμένο επανέλεγχο μέσω ανθρώπινου δυναμικού.

Η ισορροπία μεταξύ Ακρίβειας και Ανάκλησης αποτυπώνεται συχνά μέσω του δείκτη F1-Score, ο οποίος αποτελεί τον αρμονικό μέσο των δύο παραπάνω τύπων:

$$F1 = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

Στη θεωρία ταξινόμησης είναι γνωστό ότι μεταξύ των 2 παραπάνω τύπων υπάρχει αναπόφευκτο trade-off: η προσπάθεια αύξησης της Ακρίβειας συχνά μειώνει την Ανάκληση, ενώ η αύξηση της Ανάκλησης μπορεί να οδηγήσει σε περισσότερα ψευδώς θετικά αποτελέσματα. Για τον λόγο αυτό χρησιμοποιούνται καμπύλες Precision–Recall και συνολικές μετρικές AUC-PR, οι οποίες επιτρέπουν την αξιολόγηση της συμπεριφοράς του μοντέλου σε διαφορετικά επίπεδα απόφασης.

Σε αυτό το σημείο εισέρχεται η σημασία του Κατωφλίου Αποδοχής (Decision Threshold). Ένα μοντέλο Μηχανικής Μάθησης δεν αποδίδει μόνο δυαδική απόφαση, αλλά παράγει συνήθως μία πιθανότητα ή score ομοιότητας. Η τελική απόφαση λαμβάνεται όταν η παραγόμενη τιμή υπερβεί ένα προκαθορισμένο όριο αποδοχής. Για παράδειγμα, εάν το

σύστημα εκτιμήσει πιθανότητα ταύτισης 88% και το threshold έχει τεθεί στο 95%, τότε η πρόβλεψη απορρίπτεται προσωρινά ή προωθείται σε ανθρώπινο έλεγχο.

Ο καθορισμός του threshold δεν αποτελεί μόνο τεχνική παράμετρο, αλλά συνδέεται άμεσα με την επιχειρησιακή πολιτική διαχείρισης ρίσκου. Εάν το κόστος ενός Ψευδώς Θετικού αποτελέσματος είναι υψηλό, τότε το κατώφλι τίθεται συνήθως αυστηρότερα ώστε να μειωθούν οι λανθασμένες αποδοχές. Αντίθετα, όταν μεγαλύτερο πρόβλημα αποτελεί η απώλεια πραγματικών ταυτίσεων, μπορεί να επιλεγεί χαμηλότερο threshold με σκοπό την αύξηση της Ανάκλησης.

Συνεπώς, η βελτιστοποίηση του συγκεκριμένου σημείου απόφασης αποτελεί συνδυασμό στατιστικής αξιολόγησης και επιχειρησιακής στρατηγικής.

Στην παρούσα μελέτη, η υβριδική προσέγγιση αξιοποιεί αυτό ακριβώς το πλαίσιο. Το επίπεδο Μηχανικής Μάθησης υπολογίζει τις πιθανότητες αντιστοίχισης, το σημασιολογικό επίπεδο embeddings βελτιώνει την αναπαράσταση των κειμενικών δεδομένων, ενώ το τελικό επίπεδο κανόνων εφαρμόζει το thresholding και τους ελέγχους αποδοχής πριν την οριστική επιχειρησιακή ενέργεια. Με τον τρόπο αυτό, η τελική απόφαση δεν βασίζεται σε μία μεμονωμένη πρόβλεψη αλλά σε συνδυασμό στατιστικής εκτίμησης, σημασιολογικής ανάλυσης και προκαθορισμένων ορίων ασφάλειας.

<https://www.nature.com/articles/s41598-024-56706-x>

(Evaluation metrics and statistical)

<https://link.springer.com/article/10.1007/s41060-024-00584-z>

(Comparison of thresholds for a convolutional neural network)

1.4 Στόχοι και ερευνητικά ερωτήματα.

Στόχος της εν λόγω διπλωματικής είναι η εφαρμογή στην πράξη και αξιολόγηση υβριδικών συστημάτων, τα οποία συνδυάζουν Μεγάλα Γλωσσικά Μοντέλα (LLMs), τεχνικές σημασιολογικής ενσωμάτωσης (embeddings) και αλγορίθμους Μηχανικής Μάθησης.

Η προσέγγιση αυτή αποσκοπεί στη βελτίωση της αυτοματοποίησης επιχειρησιακών διαδικασιών, διατηρώντας παράλληλα υψηλό επίπεδο αξιοπιστίας και ελέγχου.

Για την επίτευξη του παραπάνω σκοπού, τίθενται οι ακόλουθοι επιμέρους στόχοι:

1. Σχεδιασμός ολοκληρωμένης αρχιτεκτονικής συστήματος (end-to-end architecture):
2. Ανάπτυξη μίας ολοκληρωμένης ροής δεδομένων (data pipeline), ικανής να συλλέγει, να καθαρίζει και να ενοποιεί ετερογενή δεδομένα προμηθευτών σε ενιαίο επιχειρησιακό περιβάλλον έτοιμο για ανάλυση.

3. Υλοποίηση συστήματος σημασιολογικής αντιστοίχισης (semantic matching pipeline):
4. Αξιολόγηση σύγχρονων embedding μοντέλων. Για την αναπαράσταση προϊόντων σε διανυσματική μορφή, καθώς και αξιοποίηση του αλγορίθμων για βελτίωση της ακρίβειας αντιστοίχισης.

Πειραματική αξιολόγηση και συγκριτική ανάλυση:

- Μέτρηση δεικτών απόδοσης όπως Precision, Recall και F1-score σε πραγματικά επιχειρησιακά δεδομένα. Παράλληλα, πραγματοποιείται συγκριτική αξιολόγηση μεταξύ:
- συστημάτων αποκλειστικά βασισμένων σε ML (ML-only),
- συστημάτων αποκλειστικά βασισμένων σε LLMs (LLM-only),
- και της προτεινόμενης υβριδικής προσέγγισης.

Αποτίμηση επιχειρησιακής και οικονομικής αξίας:

Διερεύνηση της επίδρασης του υβριδικού συστήματος στη μείωση λειτουργικού κόστους, στον περιορισμό των ψευδαισθήσεων (hallucinations) και στη βελτίωση της διαδικασίας λήψης αποφάσεων μέσω δεδομένων.

Με βάση τους παραπάνω στόχους, το βασικό ερευνητικό ερώτημα της εργασίας αφορά το κατά πόσο η ενσωμάτωση αρχιτεκτονικών Retrieval-Augmented Generation (RAG) σε συνδυασμό με ensemble αλγορίθμους Μηχανικής Μάθησης μπορεί να περιορίσει σημαντικά τα Σφάλματα Τύπου I (False Positives) και να αυξήσει την αξιοπιστία των διαδικασιών σημασιολογικής αντιστοίχισης σε περιβάλλοντα με μεγάλο όγκο αδόμητων δεδομένων. Παράλληλα, εξετάζεται κατά πόσο η υβριδική αυτή προσέγγιση ενισχύει τη συνολική επιχειρησιακή αποδοτικότητα και την ποιότητα λήψης αποφάσεων.

<https://www.mdpi.com/2673-2688/7/2/51>

(Hybrid AI and LLM-Enabled Agent-Based Real-Time Decision)

1.5 Μεθοδολογική προσέγγιση.

Για την επίτευξη των ερευνητικών στόχων της παρούσας εργασίας, η μεθοδολογική προσέγγιση διαχωρίζεται σε δύο βασικούς άξονες: τη θεωρητική βιβλιογραφική ανάλυση και την πρακτική υλοποίηση του συστήματος.

Αρχικά, πραγματοποιείται βιβλιογραφική ανασκόπηση σύγχρονων επιστημονικών μελετών σχετικά με τα Μεγάλα Γλωσσικά Μοντέλα (LLMs), τις embedding-based τεχνικές, τη

Μηχανική Μάθηση και τη Μπεϋζιανή θεωρία αποφάσεων. Παράλληλα, εξετάζονται έννοιες που σχετίζονται με την Ευφυή Αυτοματοποίηση (Intelligent Automation), καθώς και σύγχρονες μέθοδοι ανάκτησης πληροφορίας (Information Retrieval). Η θεωρητική ανάλυση συμβάλλει στην κατανόηση τόσο των δυνατοτήτων όσο και των περιορισμών των συστημάτων τεχνητής νοημοσύνης, με ιδιαίτερη έμφαση στο φαινόμενο των ψευδαισθήσεων (hallucinations).

Ο δεύτερος άξονας αφορά την τεχνική υλοποίηση του προτεινόμενου υβριδικού συστήματος. Η ανάπτυξη πραγματοποιείται στη γλώσσα προγραμματισμού Python, η οποία χρησιμοποιείται ευρέως σε εφαρμογές Επιστήμης Δεδομένων και Μηχανικής Μάθησης. Για τη διαχείριση των δομημένων επιχειρησιακών δεδομένων χρησιμοποιείται Microsoft SQL Server, ενώ η επικοινωνία με εξωτερικές υπηρεσίες και η άντληση δυναμικών δεδομένων πραγματοποιείται μέσω APIs ή και bots.

Για την υποστήριξη λειτουργιών χαμηλής καθυστέρησης (low latency) και τον συντονισμό πολλαπλών υποσυστημάτων, χρησιμοποιούνται τεχνολογίες in-memory αποθήκευσης δεδομένων, όπως το Redis. Η χρήση του επιτρέπει την υλοποίηση μηχανισμών caching και τη διαχείριση ασύγχρονων ροών δεδομένων και συμβάντων μεταξύ των επιμέρους μονάδων του συστήματος.

Υπολογίζονται και αναλύονται δείκτες απόδοσης, όπως Precision, Recall και F1-score, υπό διαφορετικές ρυθμίσεις κατωφλίων (threshold tuning) και μέσω της πειραματικής διαδικασίας αξιολογείται η συνολική αποτελεσματικότητα και η επιχειρησιακή εφαρμογή.

1.6 Δομή της διπλωματικής εργασίας.

Η παρούσα διπλωματική εργασία αποτελείται από πέντε κεφάλαια, εξετάζεται θεωρητικά αλλά και εφαρμοσμένα – τεχνικά.

- Πρώτο Κεφάλαιο:

Παρουσιάζεται το αντικείμενο της έρευνας και αναλύονται τα βασικά επιχειρησιακά προβλήματα που σχετίζονται με τη διαχείριση ετερογενών δεδομένων. Παράλληλα, διατυπώνονται οι στόχοι της εργασίας, τα ερευνητικά ερωτήματα και η μεθοδολογική προσέγγιση που ακολουθείται.

- Δεύτερο Κεφάλαιο:
Αναπτύσσεται το θεωρητικό υπόβαθρο της έρευνας. Εξετάζονται οι μηχανισμοί λειτουργίας των Μεγάλων Γλωσσικών Μοντέλων (LLMs), οι embedding-based τεχνικές, βασικές έννοιες της Μηχανικής Μάθησης και της Θεωρίας Πιθανοτήτων, καθώς και οι κίνδυνοι που σχετίζονται με τα φαινόμενα hallucinations στα συστήματα τεχνητής νοημοσύνης.
- Τρίτο Κεφάλαιο:
Περιγράφεται ο αρχιτεκτονικός σχεδιασμός και η τεχνική υλοποίηση του προτεινόμενου υβριδικού συστήματος. Παρουσιάζονται τα μοντέλα και οι τεχνολογίες που αξιοποιούνται, όπως οι αλγόριθμοι σημασιολογικής αντιστοίχισης, τα embedding μοντέλα και τα πλαίσια Retrieval-Augmented Generation (RAG), καθώς και ο τρόπος διασύνδεσής τους στο συνολικό σύστημα.
- Τέταρτο Κεφάλαιο:
- Παρουσιάζονται τα πειραματικά αποτελέσματα και η αξιολόγηση του συστήματος με χρήση στατιστικών μετρικών, όπως Precision, Recall, F1-score και AUC-PR. Επιπλέον, πραγματοποιείται συγκριτική ανάλυση μεταξύ της προτεινόμενης υβριδικής προσέγγισης και άλλων μεθόδων που βασίζονται σε ML ή LLMs.
- Πέμπτο Κεφάλαιο:
- Παρατίθενται τα τελικά συμπεράσματα της έρευνας και αξιολογείται η δυνατότητα αξιοποίησης των υβριδικών συστημάτων AI σε πραγματικά επιχειρησιακά περιβάλλοντα. Τέλος, προτείνονται κατευθύνσεις για μελλοντική έρευνα και περαιτέρω βελτίωση της αρχιτεκτονικής και των μεθόδων που παρουσιάζονται στην εργασία.
Το κεφάλαιο συνοδεύεται από παραρτήματα που περιλαμβάνουν υποστηρικτικό υλικό, δομές δεδομένων και τμήματα του κώδικα υλοποίησης.

2. Θεωρία LLMs, MLs και τεχνολογίες εφαρμογής των μοντέλων:

Η εξέλιξη της Τεχνητής Νοημοσύνης τα τελευταία χρόνια έχει επηρεάσει σε μεγάλο βαθμό τους τρόπους με τους οποίους επιχειρήσεις, οργανισμοί και πληροφοριακά συστήματα επεξεργάζονται δεδομένα και υποστηρίζουν διαδικασίες λήψης αποφάσεων. Η μετάβαση από παραδοσιακές στατιστικές και αλγοριθμικές προσεγγίσεις σε σύγχρονα συστήματα που συνδυάζουν νευρωνικά δίκτυα, γλωσσικά μοντέλα και μηχανισμούς κανόνων έχει οδηγήσει στην ανάπτυξη νέων πιο ευέλικτων και αποδοτικών εφαρμογών.

Στο παρόν κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο των βασικών τεχνολογιών που αξιοποιούνται στην εργασία, με έμφαση στα Μεγάλα Γλωσσικά Μοντέλα (Large Language Models – LLMs), στη Μηχανική Μάθηση (Machine Learning – ML) και στις τεχνικές διανυσματικής αναπαράστασης κειμένου (embeddings). Παράλληλα, αναλύονται ζητήματα που σχετίζονται με τη σημασιολογική ομοιότητα, τη χρήση cosine similarity, τα προβλήματα αξιοπιστίας των LLMs και το φαινόμενο των hallucinations.

Επιπλέον, εξετάζονται τεχνικές όπως το Prompt Engineering, οι αρχιτεκτονικές Retrieval-Augmented Generation (RAG) και οι υβριδικές neuro-symbolic προσεγγίσεις, οι οποίες χρησιμοποιούνται όλο και περισσότερο σε επιχειρησιακά περιβάλλοντα όπου απαιτείται συνδυασμός σημασιολογικής κατανόησης, στατιστικής αξιολόγησης και ελεγχόμενης λήψης αποφάσεων.

Το θεωρητικό αυτό πλαίσιο συνδέεται άμεσα με την πρακτική υλοποίηση της εργασίας, καθώς αρκετές από τις τεχνολογίες που παρουσιάζονται αξιοποιούνται στα μοντέλα αντιστοίχισης προϊόντων, στους μηχανισμούς embeddings και στις διαδικασίες αξιολόγησης που αναπτύσσονται στα επόμενα κεφάλαια.

2.1 Μεγάλα Γλωσσικά Μοντέλα (LLMs).

Τα Μεγάλα Γλωσσικά Μοντέλα (Large Language Models – LLMs) αποτελούν μία από τις σημαντικότερες εξελίξεις στον τομέα της Επεξεργασίας Φυσικής Γλώσσας (Natural Language Processing – NLP), επηρεάζοντας τον τρόπο αλληλεπίδρασης ανθρώπου και πληροφοριακών συστημάτων – μηχανής. Η αξιοποίησή τους επεκτείνεται πλέον σε ευρύ φάσμα εφαρμογών, όπως η αναζήτηση πληροφορίας, η ανάλυση κειμένου, η παραγωγή περιεχομένου, η ανάπτυξη λογισμικού ή υποστήριξη επιχειρησιακών διαδικασιών, κ.α.

Τα μοντέλα αυτά βασίζονται σε βαθιά νευρωνικά δίκτυα (deep neural networks) και εκπαιδεύονται πάνω σε εξαιρετικά μεγάλους όγκους ετερογενών δεδομένων, οι οποίοι προέρχονται από διαδικτυακές πηγές, επιστημονικά κείμενα και οργανωμένες βάσεις γνώσης. Η κλιμάκωση (scaling) των μοντέλων σε εκατοντάδες δισεκατομμύρια παραμέτρους έχει αλλάξει ριζικά το τοπίο. Δεν μιλάμε πια μόνο για καλύτερες επιδόσεις, αλλά για μια ικανότητα κατανόησης μοτίβων και παραγωγής λόγου που πλησιάζει, σε πολλές περιπτώσεις, την ανθρώπινη ροή και συνοχή και συμπεριφορά. Παράλληλα, εμφανίζονται δυνατότητες όπως η παραγωγή πηγαίου κώδικα, ανάλυση λογικών σχέσεων, μετάφραση και υποστήριξη σύνθετων ερωτημάτων.

Η λειτουργία των LLMs στηρίζεται κυρίως σε στατιστικές εκτιμήσεις βασιζόμενες στη θεωρία πιθανοτήτων. Με βάση αυτή τη διαδικασία, το μοντέλο δεν «κατανοεί» την πληροφορία με ανθρώπινο τρόπο, αλλά υπολογίζει πιθανότητες εμφάνισης γλωσσικών μοτίβων που έχουν παρατηρηθεί κατά την εκπαίδευσή του. Για τον λόγο αυτό, τα αποτελέσματα που παράγονται μπορεί να είναι γλωσσικά ορθά, χωρίς όμως να είναι πάντοτε πραγματολογικά ακριβή ή με λογική συνέπεια τουλάχιστον κατά την ανθρώπινη μέση λογική.

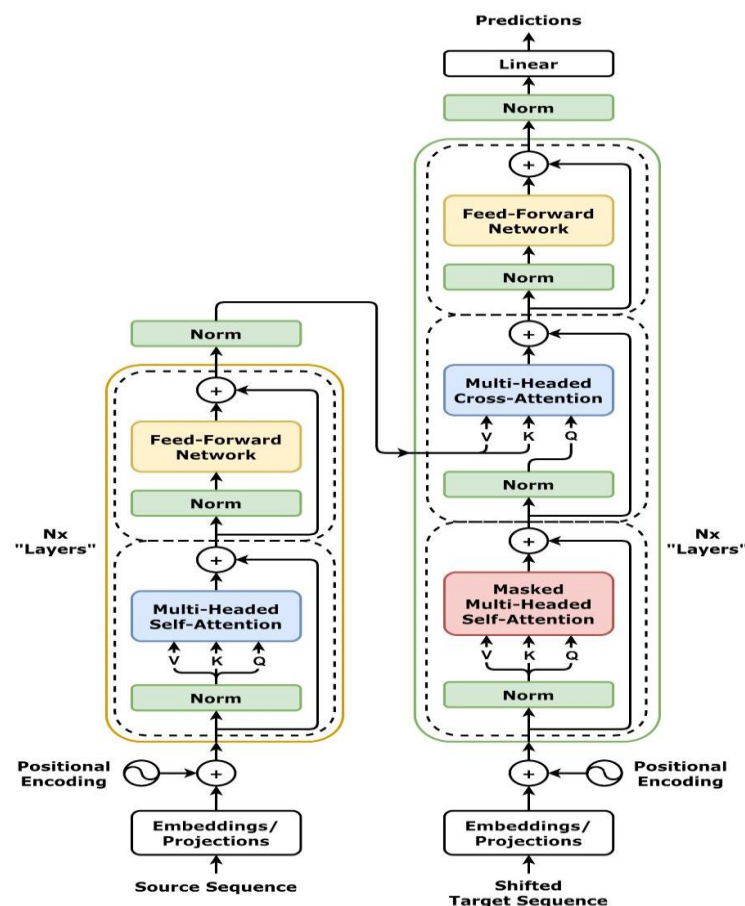
Η συγκεκριμένη ιδιότητα οδηγεί σε φαινόμενα παραγωγής ανακριβών ή ανυπόστατων πληροφοριών, γνωστά ως hallucinations. Το πρόβλημα αυτό αποκτά ιδιαίτερη σημασία σε περιβάλλοντα όπου απαιτείται αυστηρή αξιοπιστία, όπως σε επιχειρησιακά πληροφοριακά συστήματα, οικονομικές εφαρμογές ή διαδικασίες αυτοματοποιημένης λήψης αποφάσεων. Για τον λόγο αυτό, η σύγχρονη πρακτική δεν αντιμετωπίζει τα LLMs ως αυτόνομα συστήματα λήψης αποφάσεων, αλλά ως υποσυστήματα εντός υβριδικών αρχιτεκτονικών, όπου συνδυάζονται με μηχανισμούς ελέγχου, κανόνες επικύρωσης και παραδοσιακούς αλγορίθμους Μηχανικής Μάθησης (Machine Learning – ML).

Τα LLMs χρησιμοποιούνται κυρίως ως μηχανισμοί σημασιολογικής κατανόησης και υποστήριξης-αλληλεπίδρασης μέσω φυσικής γλώσσας. Η τελική επαλήθευση των αποτελεσμάτων υλοποιείται με ντετερμινιστικά συστήματα, τα οποία αξιολογούν τα παραγόμενα αποτελέσματα με βάση συγκεκριμένους κανόνες, thresholds και τα δεδομένα αναφοράς. <https://arxiv.org/abs/2501.04040>

(A Survey on Large Language Models with some Insights on their Capabilities and Limitations)

2.1.1 Ορισμός και βασικές αρχές λειτουργίας.

Η λειτουργία των σύγχρονων Μεγάλων Γλωσσικών Μοντέλων (Large Language Models – LLMs) βασίζεται κυρίως στην αρχιτεκτονική Transformer, αποτέλεσε σημείο καμπής για την Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing – NLP). Η αρχιτεκτονική αυτή αντικατέστησε τις παλαιότερες σειριακές προσεγγίσεις, όπως τα Αναδρομικά Νευρωνικά Δίκτυα (Recurrent Neural Networks – RNNs) και τα Long Short-Term Memory networks (LSTMs), επιτρέποντας την παράλληλη επεξεργασία μεγάλων ακολουθιών κειμένου. Με τον τρόπο αυτό βελτιώθηκαν οι χρόνοι εκπαίδευσης και η δυνατότητα κλιμάκωσης των μοντέλων με δισεκατομμύρια παραμέτρους.



Εικόνα 1. Τυπική αρχιτεκτονική μετασχηματιστή, όπου στα αριστερά φαίνεται ένας κωδικοποιητής και στα δεξιά ένας αποκωδικοποιητής.

<https://wccftech.com/evolution-of-artificial-intelligence-handwritten-rules-to-generative-and-agentic-machines/>

Κεντρικό στοιχείο της αρχιτεκτονικής Transformer αποτελεί ο μηχανισμός προσοχής (Attention Mechanism), και ειδικότερα η αυτο-προσοχή (Self-Attention). Ο μηχανισμός αυτός επιτρέπει στο μοντέλο να εκτιμά τη σημασία κάθε λέξης μέσα σε μία πρόταση σε σχέση με τις υπόλοιπες, ανεξάρτητα από τη θέση ή την απόστασή τους μέσα στο κείμενο. Κατά την επεξεργασία, το κείμενο μετατρέπεται σε μικρότερες υπολογιστικές μονάδες, τα λεγόμενα tokens, πάνω στα οποία εφαρμόζεται ο υπολογισμός της προσοχής.

Η διαδικασία αυτή βασίζεται στους πίνακες Query (Q), Key (K) και Value (V), μέσω των οποίων υπολογίζεται η συσχέτιση μεταξύ των tokens:

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

όπου d_k αντιστοιχεί στη διάσταση των διανυσμάτων Key. Μέσω αυτής της διαδικασίας δημιουργούνται συμφραζόμενες σημασιολογικές αναπαραστάσεις (contextual embeddings), οι οποίες επιτρέπουν στο μοντέλο να προσαρμόζει δυναμικά τη σημασία μιας λέξης ανάλογα με το περιβάλλον στο οποίο εμφανίζεται.

Για παράδειγμα, στην πρόταση «Ο τραπεζικός υπάλληλος περπάτησε κοντά στην όχθη και κάθισε στην τράπεζα», το μοντέλο μπορεί να αποδώσει διαφορετική σημασία στη λέξη «τράπεζα» ανάλογα με τις υπόλοιπες λέξεις του κειμένου. Η σύνδεση με όρους όπως «όχθη» ή «ποτάμι» αυξάνει την πιθανότητα η λέξη να αναφέρεται σε αντικείμενο ή σημείο καθίσματος και όχι σε χρηματοπιστωτικό οργανισμό.

Οι σύγχρονες αρχιτεκτονικές LLMs διακρίνονται κυρίως σε δύο βασικές κατηγορίες: τα μοντέλα αποκωδικοποιητή (decoder-only) και τα μοντέλα κωδικοποιητή-αποκωδικοποιητή (encoder-decoder). Τα decoder-only μοντέλα, όπως οι οικογένειες GPT και LLaMA, λειτουργούν με αυτοπαλλινδρόμηση (autoregressive), προβλέποντας διαδοχικά το επόμενο token με βάση τα προηγούμενα δεδομένα εισόδου. <https://arxiv.org/pdf/1706.03762>

(Attention Is All You Need)

Η βασική αρχή λειτουργίας τους εκφράζεται μέσω της πιθανότητας:

$$P(w_t | w_1, w_2, \dots, w_{t-1})$$

δηλαδή της πιθανότητας εμφάνισης της επόμενης λέξης w_t , με βάση τις προηγούμενες λέξεις της ακολουθίας.

Τα τελευταία έτη, η ερευνητική δραστηριότητα έχει επεκταθεί και σε αρχιτεκτονικές encoder-decoder, καθώς παρουσιάζουν ιδιαίτερα καλή απόδοση σε διαδικασίες fine-tuning και εξειδικευμένων εφαρμογών. Παράλληλα, λόγω της αυξημένης υπολογιστικής πολυπλοκότητας του μηχανισμού attention, αναπτύχθηκαν τεχνικές όπως η sparse attention και η linear attention, οι οποίες περιορίζουν το υπολογιστικό κόστος και επιτρέπουν την επεξεργασία πολύ μεγάλων ακολουθιών κειμένου.

Η εκπαίδευση των LLMs πραγματοποιείται συνήθως σε δύο κύρια στάδια.

1. Το πρώτο είναι η προεκπαίδευση (pre-training), κατά την οποία το μοντέλο εκπαιδεύεται σε τεράστιους όγκους κειμένου ώστε να μάθει τις στατιστικές δομές της γλώσσας και να ελαχιστοποιεί το σφάλμα πρόβλεψης του επόμενου token (Mao et al., 2024; Huang, 2025).
2. Το δεύτερο στάδιο είναι η μικρο-ρύθμιση (fine-tuning) και η ευθυγράμμιση (alignment), όπου το μοντέλο προσαρμόζεται σε πιο εξειδικευμένα δεδομένα και εκπαιδεύεται ώστε να ακολουθεί οδηγίες και επιχειρησιακούς κανόνες. Στο πλαίσιο αυτό αξιοποιούνται τεχνικές όπως το Supervised Fine-Tuning (SFT) και το Reinforcement Learning from Human Feedback. RLHF.

Ιδιαίτερη σημασία στη σύγχρονη βιβλιογραφία έχουν αποκτήσει οι μέθοδοι Parameter-Efficient Fine-Tuning (PEFT), όπως το LoRA (Low-Rank Adaptation) και το QLoRA. Οι τεχνικές αυτές επιτρέπουν την προσαρμογή μεγάλων γλωσσικών μοντέλων χωρίς να απαιτείται η επανεκπαίδευση όλων των παραμέτρων τους. Στο LoRA, τα βασικά βάρη του μοντέλου παραμένουν σταθερά και εκπαιδεύονται μόνο μικροί πίνακες χαμηλού βαθμού, σύμφωνα με τη σχέση:

$$\Delta W = AB$$

όπου οι πίνακες A και B αναπαριστούν τις μεταβολές των αρχικών βαρών.

Το QLoRA επεκτείνει αυτή την προσέγγιση μέσω κβαντοποίησης των βαρών σε χαμηλότερη ακρίβεια (π.χ. 4-bit), μειώνοντας σημαντικά τις απαιτήσεις μνήμης και επιτρέποντας την εκπαίδευση μεγάλων μοντέλων ακόμη και σε περιορισμένο υπολογιστικό εξοπλισμό. <https://arxiv.org/pdf/2303.18223>

(*A Survey of Large Language Models*)

2.1.2 Ντετερμινιστικός χαρακτήρας και περιορισμοί.

Σε αντίθεση με τα παραδοσιακά πληροφοριακά συστήματα, τα οποία λειτουργούν βάσει αυστηρά ντετερμινιστικών κανόνων και παράγουν πάντοτε το ίδιο αποτέλεσμα για την ίδια είσοδο, τα Μεγάλα Γλωσσικά Μοντέλα (LLMs) παρουσιάζουν στοχαστική συμπεριφορά. Η παραγωγή της απόκρισης βασίζεται σε θεωρίες πιθανοτήτων και δειγματοληψία, γεγονός που σημαίνει ότι το ίδιο ερώτημα μπορεί να οδηγήσει σε διαφορετικές απαντήσεις κατά την επαναλαμβανόμενη εκτέλεση του μοντέλου. Η ιδιότητα αυτή αποτελεί έναν από τους βασικούς περιορισμούς των LLMs σε περιβάλλοντα όπου απαιτείται πλήρης προβλεψιμότητα και επαναληψιμότητα αποτελεσμάτων.

Ο βαθμός της τυχαιότητας ελέγχεται κυρίως μέσω της παραμέτρου θερμοκρασίας (temperature). Η παράμετρος αυτή εφαρμόζεται πριν από τη συνάρτηση softmax, κατά το στάδιο μετατροπής των logits σε πιθανότητες επιλογής του επόμενου token:

$$P(x_i) = \frac{\exp(z_i / T)}{\sum_j \exp(z_j / T)}$$

όπου z_i , αντιστοιχεί στο score του token και T στη θερμοκρασία.

Όταν η τιμή της θερμοκρασίας είναι χαμηλή ($T \rightarrow 0$), η κατανομή πιθανοτήτων γίνεται πιο «αιχμηρή», με αποτέλεσμα το μοντέλο να επιλέγει σχεδόν πάντα το πιθανότερο token.

Η προσέγγιση αυτή χρησιμοποιείται συνήθως σε εφαρμογές που απαιτούν σταθερότητα και συνέπεια, όπως η εξαγωγή δομημένων δεδομένων, η ταξινόμηση ή η αντιστοίχιση προϊόντων.

Αντίθετα, υψηλότερες τιμές temperature οδηγούν σε πιο ομοιόμορφη κατανομή πιθανοτήτων, ενισχύοντας τη γλωσσική ποικιλία και τη δημιουργικότητα του παραγόμενου κειμένου, αλλά αυξάνοντας παράλληλα την πιθανότητα σφαλμάτων ή ασυνεπειών.

Πέρα από τη στοχαστική φύση τους, τα LLMs παρουσιάζουν και επιπρόσθετους περιορισμούς. Ένας βασικός περιορισμός αφορά τη δυσκολία εκτέλεσης αυστηρού λογικού και μαθηματικού συλλογισμού. Παρότι τα μοντέλα επιτυγχάνουν υψηλή απόδοση σε γλωσσικές εργασίες, η επεξεργασία αριθμητικών ή αλγεβρικών σχέσεων παραμένει προβληματική, καθώς οι αριθμοί αντιμετωπίζονται κυρίως ως γλωσσικά σύμβολα και όχι ως πραγματικές μαθηματικές οντότητες.

Επιπλέον, τα LLMs περιορίζονται από το λεγόμενο knowledge cutoff, δηλαδή το χρονικό σημείο μέχρι το οποίο εκπαιδεύτηκαν τα δεδομένα τους. Χωρίς τη χρήση εξωτερικών μηχανισμών ανάκτησης πληροφορίας, όπως αρχιτεκτονικές Retrieval-Augmented Generation (RAG), τα μοντέλα αδυνατούν να γνωρίζουν νεότερα γεγονότα ή εξειδικευμένα εσωτερικά δεδομένα ενός οργανισμού.

Ιδιαίτερα σημαντικός θεωρείται και ο κίνδυνος παραγωγής ψευδών ή ανυπόστατων πληροφοριών, γνωστός ως hallucination. Στις περιπτώσεις αυτές, το μοντέλο παράγει απαντήσεις που εμφανίζονται γλωσσικά ορθές και πειστικές, χωρίς όμως να βασίζονται σε πραγματικά δεδομένα ή τεκμηριωμένη γνώση. Το φαινόμενο αυτό καθιστά προβληματική τη χρήση των LLMs σε κρίσιμα περιβάλλοντα, όπως ιατρικά, οικονομικά ή νομικά συστήματα, χωρίς την ύπαρξη επιπρόσθετων μηχανισμών ελέγχου.

Για τον λόγο αυτό, η σύγχρονη βιβλιογραφία προτείνει την ενσωμάτωση των LLMs σε υβριδικές αρχιτεκτονικές, όπου οι αποκρίσεις τους επικυρώνονται μέσω ντετερμινιστικών αλγορίθμων, κανόνων συμμόρφωσης και μηχανισμών επαλήθευσης. Η προσέγγιση αυτή επιτρέπει την αξιοποίηση των σημασιολογικών δυνατοτήτων των γλωσσικών μοντέλων, περιορίζοντας παράλληλα τους κινδύνους που προκύπτουν από τη στοχαστική λειτουργία τους.

(Attention Is All You Need)

2.1.3 Οι Ψευδαισθήσεις (Hallucinations) και Στρατηγικές Ελέγχου.

Οι ψευδαισθήσεις (hallucinations) αποτελούν ένα από τα σημαντικότερα προβλήματα των Μεγάλων Γλωσσικών Μοντέλων. Ο όρος αναφέρεται στην παραγωγή πληροφοριών που παρουσιάζονται ως γλωσσικά και συντακτικά ορθές, χωρίς όμως να ανταποκρίνονται στην πραγματικότητα ή στα διαθέσιμα δεδομένα. Το φαινόμενο αυτό δεν αποτελεί μεμονωμένη αστοχία, αλλά συνδέεται με τον ίδιο τον τρόπο λειτουργίας των LLMs, τα οποία βασίζονται στη στατιστική πρόβλεψη ακολουθιών λέξεων και όχι σε πραγματική κατανόηση ή επαλήθευση γνώσης.

Οι ψευδαισθήσεις εμφανίζονται συχνότερα σε περιπτώσεις όπου το μοντέλο καλείται να απαντήσει σε εξειδικευμένα ερωτήματα, να συμπληρώσει ελλείψεις πληροφορίες ή να παράγει περιεχόμενο για το οποίο δεν διαθέτει επαρκή δεδομένα εκπαίδευσης. Σε επιχειρησιακά περιβάλλοντα, το πρόβλημα αποκτά ιδιαίτερη σημασία, καθώς μια

λανθασμένη πληροφορία μπορεί να οδηγήσει σε εσφαλμένες οικονομικές, νομικές ή λειτουργικές αποφάσεις.

Η βιβλιογραφία αποδίδει το φαινόμενο σε τρεις βασικές κατηγορίες αιτιών:

1. Περιορισμοί των δεδομένων εκπαίδευσης:

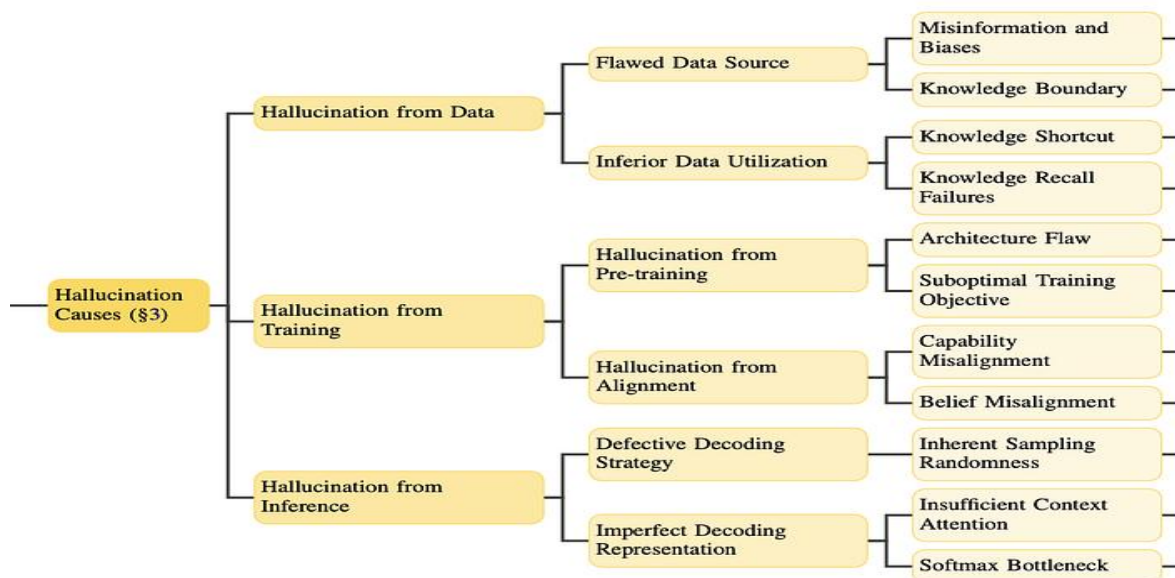
Τα μοντέλα εκπαιδεύονται σε τεράστιους όγκους δεδομένων που περιλαμβάνουν ανακρίβειες, αντιφάσεις και θόρυβο. Ως αποτέλεσμα, μέρος αυτών των ασυνεπειών μεταφέρεται και στη συμπεριφορά του μοντέλου.

2. Αρχιτεκτονικοί περιορισμοί των LLMs:

Παρότι οι μηχανισμοί attention επιτρέπουν τη διαχείριση μεγάλων ακολουθιών, τα μοντέλα δυσκολεύονται να διατηρήσουν πλήρη συνοχή σε εκτεταμένα συμφραζόμενα (long context reasoning), ειδικά όταν απαιτείται πολύπλοκος συλλογισμός.

3. Μηχανισμοί αποκωδικοποίησης (decoding):

Οι τεχνικές δειγματοληψίας, όπως το top-k sampling ή το nucleus sampling, αυξάνουν τη γλωσσική ποικιλία αλλά μπορούν να οδηγήσουν και στην παραγωγή λανθασμένων ή ανυπόστατων πληροφοριών.



Εικόνα 2. Αιτίες ψευδαισθήσεων (Hallucinations) σε όλα τα στάδια ανάπτυξης ενός LLM: από τη συλλογή δεδομένων έως τον μηχανισμό παραγωγής.

<https://generativeai.pub/what-causes-llm-hallucinations-26c10bbb03ae>

Για την αντιμετώπιση του προβλήματος έχουν αναπτυχθεί τεχνικές εντοπισμού και μείωσης των ψευδαισθήσεων.

- Μέθοδοι εντοπισμού (Detection Methods):

1. Μία από τις πιο διαδεδομένες προσεγγίσεις είναι η διασταύρωση της απάντησης με εξωτερικές πηγές γνώσης μέσω Retrieval-Augmented αξιολόγησης. Το παραγόμενο αποτέλεσμα συγκρίνεται με έγγραφα ή δεδομένα που θεωρούνται αξιόπιστα, ώστε να εκτιμηθεί η συνέπεια και η ορθότητά του.
2. Επιπλέον, χρησιμοποιούνται τεχνικές αυτο-συνέπειας (self-consistency), κατά τις οποίες το ίδιο ερώτημα υποβάλλεται πολλαπλές φορές με μικρές διαφοροποιήσεις. Υψηλή απόκλιση μεταξύ των απαντήσεων υποδηλώνει αυξημένη αβεβαιότητα του μοντέλου.
3. Σε πιο εξειδικευμένα περιβάλλοντα εφαρμόζονται και τεχνικές ανάλυσης της εσωτερικής συμπεριφοράς του μοντέλου, όπως η μελέτη των attention layers και των activation patterns, με στόχο τον εντοπισμό περιοχών χαμηλής βεβαιότητας.

- Στρατηγικές μετριασμού (Mitigation Strategies):

1. Η σημαντικότερη στρατηγική περιορισμού των hallucinations είναι η αρχιτεκτονική Retrieval-Augmented Generation (RAG). Στην προσέγγιση αυτή, το LLM δεν βασίζεται αποκλειστικά στην παραμετρική του γνώση, αλλά λαμβάνει επιπλέον πληροφορίες από εξωτερικές βάσεις δεδομένων ή εταιρικά έγγραφα πριν παράγει απάντηση.
2. Με τον τρόπο αυτό, η παραγωγή κειμένου βασίζεται σε επικαιροποιημένα και ελεγχόμενα δεδομένα, μειώνοντας σημαντικά την πιθανότητα δημιουργίας ανυπόστατων πληροφοριών. Παράλληλα, εφαρμόζονται μηχανισμοί ελέγχου (guardrails), κανόνες επικύρωσης και deterministic validation pipelines, ώστε το τελικό αποτέλεσμα να ελέγχεται πριν χρησιμοποιηθεί από επιχειρησιακά συστήματα.

Στα σύγχρονα υβριδικά περιβάλλοντα, οι τεχνικές αυτές συνδυάζονται με αλγορίθμους Μηχανικής Μάθησης και σχετικούς κανόνες συμμόρφωσης, δημιουργώντας ένα πολυεπίπεδο πλαίσιο ελέγχου που περιορίζει ουσιαστικά τον κίνδυνο σφαλμάτων.

2.1.4 Μηχανική Προτροπών (Prompt Engineering) και Στρατηγικές.

Η ευρεία διάδοση των Μεγάλων Γλωσσικών Μοντέλων οδήγησε στην ανάπτυξη ενός νέου πεδίου εφαρμοσμένης έρευνας, γνωστού ως Μηχανική Προτροπών (Prompt Engineering). Το πεδίο αυτό εξετάζει τον τρόπο με τον οποίο η διατύπωση της εισόδου επηρεάζει την ποιότητα, τη συνέπεια και τη δομή της παραγόμενης απάντησης.

Στην πράξη, ακόμη και μικρές διαφοροποιήσεις στη διατύπωση μιας προτροπής μπορούν να επηρεάσουν σημαντικά το τελικό αποτέλεσμα. Για τον λόγο αυτό, η σχεδίαση προτροπών αντιμετωπίζεται πλέον ως διαδικασία βελτιστοποίησης, ιδιαίτερα σε εφαρμογές που απαιτούν αυστηρή μορφοποίηση και υψηλή ακρίβεια.

Η σύγχρονη βιβλιογραφία διακρίνει αρκετές βασικές κατηγορίες τεχνικών prompt engineering:

1. Zero-shot Prompting:

Στην προσέγγιση zero-shot, το μοντέλο καλείται να επιλύσει μια εργασία χωρίς να του παρέχονται παραδείγματα. Η συμπεριφορά του βασίζεται αποκλειστικά στις γενικές γλωσσικές γνώσεις που απέκτησε κατά την προεκπαίδευση. Η μέθοδος είναι γρήγορη και αποδοτική, αλλά παρουσιάζει περιορισμούς σε σύνθετες ή αυστηρά δομημένες εργασίες.

2. Few-shot Prompting:

Στο few-shot prompting παρέχονται στο μοντέλο ενδεικτικά παραδείγματα εισόδου και επιθυμητής εξόδου. Μέσω αυτής της διαδικασίας, το μοντέλο προσαρμόζει τη συμπεριφορά του στο ζητούμενο μοτίβο χωρίς να απαιτείται επανεκπαίδευση των παραμέτρων του. Η τεχνική αυτή χρησιμοποιείται εκτενώς σε επιχειρησιακά περιβάλλοντα, όπως η ταξινόμηση εγγράφων, η αντιστοίχιση προϊόντων και η παραγωγή δομημένων απαντήσεων σε μορφή JSON ή XML.

3. Chain-of-Thought Prompting (CoT):

Η τεχνική Chain-of-Thought καθοδηγεί το μοντέλο να αναπτύξει σταδιακά τη συλλογιστική του πορεία αντί να παράγει άμεσα το τελικό αποτέλεσμα. Με τον τρόπο αυτό, σύνθετα προβλήματα διασπώνται σε επιμέρους βήματα, βελτιώνοντας τη λογική συνοχή και μειώνοντας τα σφάλματα συλλογισμού. Η προσέγγιση αυτή έχει αποδειχθεί ιδιαίτερα αποτελεσματική σε μαθηματικά προβλήματα, λογικές ακολουθίες και διαδικασίες ανάλυσης πολλών σταδίων.

4. Role Prompting:

Στο role prompting αποδίδεται στο μοντέλο ένας συγκεκριμένος ρόλος, όπως «αναλυτής δεδομένων», «νομικός σύμβουλος» ή «μηχανικός λογισμικού». Η τεχνική αυτή επηρεάζει τον τρόπο παραγωγής της απάντησης και συχνά οδηγεί σε πιο συνεπή αποτελέσματα.

5. Automatic Prompt Optimization (APO)

Λόγω της αυξανόμενης πολυπλοκότητας των prompts, έχουν αναπτυχθεί τεχνικές αυτόματης βελτιστοποίησης προτροπών. Στις μεθόδους αυτές, διαφορετικές εκδοχές prompts αξιολογούνται αυτόματα ως προς την απόδοσή τους σε συγκεκριμένα validation datasets, με στόχο την επιλογή της αποτελεσματικότερης μορφής.

Στα υβριδικά συστήματα, το prompt engineering δεν περιορίζεται στη διατύπωση μόνο της φυσικής γλώσσας. Αντίθετα, λειτουργεί ως μηχανισμός ελέγχου της μορφής, της δομής και της προβλεπόμενης εξόδου του θα παραγάγει το μοντέλο, με σκοπό τα παραγόμενα αποτελέσματα να μπορούν να αξιοποιηθούν και να δώσουν πραγματική αξία στις επιχειρήσεις ή οργανισμούς.

<https://arxiv.org/pdf/2406.06608>

(A Systematic Survey of Prompt Engineering Techniques)

2.2 Μηχανική Μάθηση (Machine Learning).

Παρά τη σημαντική πρόοδο των Μεγάλων Γλωσσικών Μοντέλων στην επεξεργασία φυσικής γλώσσας και στην ανάλυση δεδομένων, η κλασική Μηχανική Μάθηση (Machine Learning – ML) εξακολουθεί να αποτελεί βασικό πυλώνα των σύγχρονων πληροφοριακών συστημάτων. Οι τεχνικές ML χρησιμοποιούνται ευρέως σε προβλήματα ταξινόμησης, πρόβλεψης, ανίχνευσης προτύπων και ανάλυσης δομημένων δεδομένων, όπου απαιτούνται σταθερότητα, ερμηνεία και επαναληψιμότητα αποτελεσμάτων.

Η βασική αρχή της Μηχανικής Μάθησης στηρίζεται στην ανάπτυξη αλγορίθμων που αποκτούν γνώση μέσα από δεδομένα εκπαίδευσης, χωρίς να έχουν προγραμματιστεί ρητά για κάθε πιθανή περίπτωση. Κατά τη διαδικασία εκπαίδευσης, το μοντέλο προσπαθεί να

εντοπίσει σχέσεις και μοτίβα μεταξύ των εισόδων και των επιθυμητών αποτελεσμάτων, προσαρμόζοντας σταδιακά τις εσωτερικές παραμέτρους του.

Η διαδικασία υλοποιείται μέσω της ελαχιστοποίησης μιας συνάρτησης σφάλματος (loss function), η οποία εκφράζει τη διαφορά μεταξύ της προβλεπόμενης τιμής και της πραγματικής τιμής αναφοράς (ground truth). Με συνεχείς επαναληπτικές διαδικασίες βελτιστοποίησης, όπως ο gradient descent και οι παραλλαγές του, το μοντέλο βελτιώνει σταδιακά την απόδοσή του.

Στα σύγχρονα υβριδικά συστήματα, η Μηχανική Μάθηση λειτουργεί συμπληρωματικά προς τα LLMs. Τα γλωσσικά μοντέλα αναλαμβάνουν την εξαγωγή σημασιολογικής πληροφορίας από αδόμητα κείμενα, ενώ τα μοντέλα ML αξιοποιούν τα παραγόμενα αριθμητικά χαρακτηριστικά —όπως embeddings, similarity scores ή structured features— για τη λήψη πιο σταθερών και ερμηνεύσιμων αποφάσεων.

Η προσέγγιση αυτή είναι ιδιαίτερα σημαντική σε εφαρμογές αντιστοίχισης προϊόντων και ανάλυσης επιχειρησιακών δεδομένων, όπου απαιτείται συνδυασμός σημασιολογικής κατανόησης και ντετερμινιστικής αξιολόγησης. Για παράδειγμα, ένα embedding μοντέλο μπορεί να εντοπίσει σημασιολογική ομοιότητα μεταξύ δύο περιγραφών προϊόντων, ενώ ένας αλγόριθμος ταξινόμησης ML μπορεί να αποφασίσει αν η ομοιότητα αυτή υπερβαίνει ή όχι το αποδεκτό επιχειρησιακό κατώφλι.

Η υλοποίηση των παραπάνω θα υλοποιηθεί εδώ μέσω της Python, όπου διαθέτει βιβλιοθήκες για επεξεργασία δεδομένων και ανάπτυξη μοντέλων Μηχανικής Μάθησης.

- Η βιβλιοθήκη scikit-learn χρησιμοποιείται εκτενώς για την ανάπτυξη μοντέλων ταξινόμησης και παλινδρόμησης, καθώς και για διαδικασίες preprocessing, feature extraction και αξιολόγησης μετρικών όπως Accuracy, Precision, Recall και F1-score.
- Η βιβλιοθήκη Pandas παρέχει μηχανισμούς διαχείρισης και μετασχηματισμού δομημένων δεδομένων μέσω DataFrames, διευκολύνοντας τον καθαρισμό, τη συγχώνευση και την προετοιμασία δεδομένων πριν από την εκπαίδευση των μοντέλων.
- Παράλληλα, η NumPy χρησιμοποιείται για τη διαχείριση πολυδιάστατων πινάκων και την επιτάχυνση αριθμητικών υπολογισμών, ιδιαίτερα σε περιπτώσεις διανυσματικών πράξεων και similarity calculations.

- Σε προβλήματα υψηλής πολυπλοκότητας και μεγάλου όγκου δεδομένων αξιοποιούνται συχνά βιβλιοθήκες gradient boosting, όπως οι XGBoost και LightGBM, οι οποίες προσφέρουν υψηλή απόδοση σε εργασίες ταξινόμησης και πρόβλεψης πάνω σε δομημένα δεδομένα.

Τα παραπάνω θα χρησιμοποιηθούν πρακτικά και στην εν λόγω Δ.Ε.

<https://jmlr.csail.mit.edu/papers/volume12/pedregosa11a/pedregosa11a.pdf>

(Scikit-learn: Machine Learning in Python)

2.2.1 Επιβλεπόμενη (Supervised) και Μη Επιβλεπόμενη (Unsupervised) Μάθηση.

Οι προσεγγίσεις στη μηχανική μάθηση διαχωρίζονται ανάλογα με τη διαθεσιμότητα των "απαντήσεων" (labels) κατά τη φάση της εκπαίδευσης:

Στην **Επιβλεπόμενη Μάθηση** (Supervised Learning), το μοντέλο εκπαιδεύεται πάνω σε δεδομένα για τα οποία είναι ήδη γνωστή η επιθυμητή έξοδος. Ο αλγόριθμος επιχειρεί να μάθει τη σχέση μεταξύ των χαρακτηριστικών εισόδου (features) και της σωστής απάντησης, ώστε να μπορεί αργότερα να προβλέπει αντίστοιχες περιπτώσεις σε άγνωστα δεδομένα. Στο πλαίσιο αντιστοίχισης προϊόντων, για παράδειγμα, ένα σύστημα μπορεί να εκπαιδευτεί χρησιμοποιώντας ιστορικά ζεύγη περιγραφών που έχουν ήδη ταυτοποιηθεί με συγκεκριμένα ProductIDs. Με αυτόν τον τρόπο, αλγόριθμοι όπως το Random Forest ή το XGBoost μαθαίνουν να αναγνωρίζουν πρότυπα ομοιότητας μεταξύ διαφορετικών περιγραφών προϊόντων. Η προσέγγιση αυτή επιτυγχάνει συνήθως υψηλή ακρίβεια, προϋποθέτει όμως την ύπαρξη μεγάλων και σωστά επισημασμένων συνόλων δεδομένων.

Αντίθετα, στη **Μη Επιβλεπόμενη Μάθηση** (Unsupervised Learning), το μοντέλο δεν διαθέτει προκαθορισμένες απαντήσεις και καλείται να εντοπίσει μόνο του δομές, ομάδες ή συσχετίσεις μέσα στα δεδομένα. Η διαδικασία αυτή χρησιμοποιείται κυρίως για ομαδοποίηση (clustering), ανίχνευση ανωμαλιών και εξαγωγή λανθανόντων προτύπων. Σε εφαρμογές σημασιολογικής αντιστοίχισης, οι περιγραφές μετατρέπονται αρχικά σε διανυσματικές αναπαραστάσεις (embeddings) και στη συνέχεια ομαδοποιούνται με αλγορίθμους όπως ο K-Means, βάσει της εγγύτητάς τους στον διανυσματικό χώρο. Με αυτόν τον τρόπο, το σύστημα μπορεί να εντοπίζει περιγραφές που αναφέρονται στο ίδιο ή σε παρόμοιο προϊόν, ακόμη και όταν δεν υπάρχει ιστορικό αντιστοίχισης. Η προσέγγιση αυτή είναι ιδιαίτερα χρήσιμη σε περιβάλλοντα όπου εισάγονται συνεχώς νέα προϊόντα ή νέες κατηγορίες δεδομένων για τις οποίες δεν υπάρχουν διαθέσιμες ετικέτες εκπαίδευσης.

Στα σύγχρονα υβριδικά συστήματα, οι δύο προσεγγίσεις συνδυάζονται συχνά συμπληρωματικά. Η μη επιβλεπόμενη μάθηση αξιοποιείται για την αρχική αναγνώριση προτύπων και τη σημασιολογική ομαδοποίηση, ενώ η επιβλεπόμενη μάθηση χρησιμοποιείται για την τελική ταξινόμηση και επικύρωση των αποτελεσμάτων με μεγαλύτερη αξιοπιστία.

<https://arxiv.org/pdf/2308.10619>

(A Comprehensive Survey on Machine Learning Techniques)

2.2.2 Ensemble-Hybrid Αλγόριθμοι: Ανθεκτικότητα και Αντιπαραβολή XGBoost με Random Forest.

Τα πραγματικά επιχειρησιακά δεδομένα χαρακτηρίζονται συχνά από ασυνέπειες, ελλιπείς τιμές, θόρυβο και έντονα μη ισορροπημένες κατηγορίες. Σε τέτοια περιβάλλοντα, τα απλά μοντέλα ταξινόμησης παρουσιάζουν περιορισμένη ικανότητα γενίκευσης και αυξημένη πιθανότητα σφαλμάτων. Για τον λόγο αυτό, ιδιαίτερη σημασία έχουν αποκτήσει οι Ensemble αλγόριθμοι, δηλαδή τεχνικές που συνδυάζουν πολλαπλά επιμέρους μοντέλα με στόχο τη βελτίωση της συνολικής ακρίβειας και ανθεκτικότητας.

Οι Ensemble μέθοδοι βασίζονται κυρίως στη συνεργασία πολλών δέντρων αποφάσεων (decision trees). Αντί να λαμβάνεται απόφαση από ένα μόνο μοντέλο, η τελική πρόβλεψη προκύπτει από τον συνδυασμό πολλαπλών επιμέρους προβλέψεων, γεγονός που μειώνει την ευαισθησία στον θόρυβο και περιορίζει την υπερεκπαίδευση (overfitting).

Δύο από τους σημαντικότερους αλγορίθμους αυτής της κατηγορίας είναι το Random Forest και το XGBoost.

Το Random Forest βασίζεται στη λογική του bagging. Δημιουργεί μεγάλο αριθμό δέντρων αποφάσεων, όπου κάθε δέντρο εκπαιδεύεται σε διαφορετικό τυχαίο υποσύνολο δεδομένων και χαρακτηριστικών. Η τελική πρόβλεψη προκύπτει μέσω ψηφοφορίας μεταξύ των δέντρων. Η προσέγγιση αυτή προσφέρει υψηλή σταθερότητα και περιορίζει σημαντικά το overfitting, ιδιαίτερα σε μικρότερα ή μέτρια σύνολα δεδομένων.

Αντίθετα, το XGBoost ακολουθεί τη φιλοσοφία του boosting. Τα δέντρα δημιουργούνται διαδοχικά και κάθε νέο δέντρο επιχειρεί να διορθώσει τα σφάλματα των προηγούμενων. Η διαδικασία αυτή επιτρέπει στο μοντέλο να εντοπίζει πολύπλοκες και λεπτές σχέσεις μέσα στα δεδομένα, επιτυγχάνοντας ιδιαίτερα υψηλή απόδοση σε προβλήματα ταξινόμησης και πρόβλεψης.

Σύμφωνα με πρόσφατες συγκριτικές μελέτες, το XGBoost παρουσιάζει καλύτερες επιδόσεις σε ανισόρροπα επιχειρησιακά δεδομένα, ειδικά όταν συνδυάζεται με τεχνικές εξισορρόπησης όπως το SMOTE. Παρότι απαιτεί πιο προσεκτική ρύθμιση υπερπαραμέτρων (hyperparameter tuning), επιτυγχάνει συνήθως υψηλότερο F1-score και καλύτερη ικανότητα εντοπισμού σπάνιων περιπτώσεων σε σχέση με το Random Forest

Στην παρούσα εργασία, η λογική των ensemble-hybrid αρχιτεκτονικών αξιοποιείται ώστε τα στατιστικά μοντέλα μηχανικής μάθησης να λειτουργούν συμπληρωματικά με τις σημασιολογικές δυνατότητες των LLMs και των embeddings. Τα γλωσσικά μοντέλα αναλαμβάνουν την κατανόηση και τη σημασιολογική ερμηνεία των περιγραφών, ενώ οι

ensemble αλγόριθμοι χρησιμοποιούνται για την τελική αξιολόγηση και επιβεβαίωση της αντιστοίχισης, παρέχοντας πιο σταθερές αποφάσεις.

Πίνακας 3 Ανάλυση τεχνικών χαρακτηριστικών και μετρικών απόδοσης των μοντέλων Random Forest και XGBoost, με έμφαση σε ανισόρροπα (imbalanced) σύνολα δεδομένων.

Χαρακτηριστικά Random Forest (Bagging)

XGBoost (Boosting)

Μηχανισμός	Κατασκευάζει πολλαπλά δέντρα <i>παράλληλα</i> , όπου το καθένα εκπαιδεύεται σε τυχαία υποσύνολα δεδομένων και χαρακτηριστικών. Η τελική απόφαση λαμβάνεται με ψηφοφορία.	Κατασκευάζει δέντρα <i>ακολουθιακά</i> . Κάθε νέο δέντρο δημιουργείται ειδικά για να διορθώσει τα σφάλματα (υπόλοιπα) του προηγούμενου δέντρου.
Πλεονεκτήματα	Εξαιρετικά ανθεκτικό στο overfitting (υπερεκπαίδευση). Εύκολο στη ρύθμιση, χρειάζεται ελάχιστο hyperparameter tuning. Αποδοτικό σε μικρά σύνολα.	Κορυφαία ακρίβεια. Ικανότητα διαχείρισης ακραία ανισόρροπων δεδομένων όταν συνδυάζεται με τεχνικές εξισορρόπησης (όπως SMOTE). Υψηλή ταχύτητα.
Μειονεκτήματα	Καταναλώνει περισσότερη μνήμη. Δυσκολεύεται να αναγνωρίσει πολύπλοκες, λεπτές συσχετίσεις σε δύσκολα, σπάνια γεγονότα.	Επιρρεπές σε overfitting εάν τα δέντρα γίνουν πολύ βαθιά. Απαιτεί προσεκτική ρύθμιση πολυάριθμων παραμέτρων (learning rate, depth).
Απόδοση (F1-Score / Accuracy)	Εμφανίζει καλή ισορροπία, αλλά καταρρέει συχνά σε ακραία ανισόρροπα (imbalanced) δεδομένα (π.χ. < 5% churn rate). Σε συγκριτικές δοκιμές (2024) σημείωσε ακρίβεια 52%.	Κυριαρχεί συντριπτικά. Με κατάλληλη ρύθμιση και τεχνικές (GNUS, SMOTE) επιτυγχάνει το μέγιστο F1-score. Στην ίδια δοκιμή σημείωσε 61.5% και υψηλότερα.

Συνοπτικά:

Χαρακτηριστικό	Random Forest	XGBoost
Σειρά Εκπαίδευσης	Παράλληλα (Ταυτόχρονα)	Σειριακά (Ένα-ένα, διορθώνοντας το προηγούμενο)
Τελική Απόφαση	Πλειοψηφία (Μέσος Όρος / Ψηφοφορία)	Άθροισμα των διορθώσεων όλων των δέντρων
Overfitting (Υπερεκπαίδευση)	Πολύ δύσκολα υπερεκπαίδευεται (Είναι «ασφαλές»)	Μπορεί να υποστεί overfitting αν δεν ρυθμιστεί σωστά
Πού υπερέχει;	Σε μικρά datasets, είναι εύκολο και παίζει καλά χωρίς πολλές ρυθμίσεις.	Σε δύσκολα, ανισόρροπα (imbalanced) δεδομένα. Έχει την κορυφαία ακρίβεια στη βιομηχανία.

<https://arxiv.org/pdf/1603.02754>

(XGBoost: A Scalable Tree Boosting System)

2.2.3 Ο Ρόλος της Μηχανικής Μάθησης σε Επιχειρησιακά Δεδομένα.

Στις σύγχρονες επιχειρήσεις, τα δεδομένα χαρακτηρίζονται από τεράστιο όγκο και πολυμορφία (τιμές, προμηθευτές, περιγραφές, κωδικοί, τοποθεσίες, κ.α.). Η κλασική ML (σε αντίθεση με τη γενετική τεχνητή νοημοσύνη) δεν επικεντρώνεται στη δημιουργία νέου περιεχομένου, αλλά στη διάγνωση, την ταξινόμηση και την πρόβλεψη. Οι επιχειρήσεις την αξιοποιούν για την πρόβλεψη της ζήτησης, την ανίχνευση απάτης ή ανωμαλιών στα συστήματα λογιστικής, την πρόβλεψη αστοχίας εξοπλισμού (predictive maintenance) , και τη βελτιστοποίηση των αποθεμάτων.

Το σημαντικότερο πλεονέκτημα των παραδοσιακών ML μοντέλων στον επιχειρηματικό χώρο είναι η Επεξηγησιμότητα (Explainability). Ενώ ένα LLM μπορεί να δώσει μια σύνδεση προϊόντων χωρίς καμία μαθηματική τεκμηρίωση, αλγόριθμοι όπως τα Random Forests επιτρέπουν στον ελεγκτή να κατανοήσει ποια μεταβλητή (feature) είχε τη μεγαλύτερη βαρύτητα στην τελική απόφαση (feature importance). Συνδυάζοντας την ML με εταιρικούς κανόνες.

2.3 Διανυσματικές Αναπαραστάσεις Embeddings και Semantic Matching

Η σύνδεση μεταξύ της επεξεργασίας φυσικής γλώσσας και της μηχανικής μάθησης επιτυγχάνεται κυρίως μέσω των διανυσματικών αναπαραστάσεων (embeddings). Τα υπολογιστικά συστήματα δεν μπορούν να επεξεργαστούν άμεσα το νόημα του κειμένου· απαιτείται η μετατροπή του σε αριθμητική μορφή ώστε να καταστεί δυνατή η μαθηματική του ανάλυση. Τα embeddings αποτελούν ακριβώς αυτή τη γέφυρα, μετασχηματίζοντας λέξεις, προτάσεις ή ολόκληρα έγγραφα σε πολυδιάστατα διανύσματα πραγματικών αριθμών.

Σε αντίθεση με τις παραδοσιακές τεχνικές αναζήτησης που βασίζονται σε λέξεις-κλειδιά, οι σύγχρονες προσεγγίσεις semantic matching εστιάζουν στο νόημα και στα συμφραζόμενα. Έτσι, διαφορετικές διατυπώσεις που αναφέρονται στην ίδια έννοια, όπως οι όροι «κινητό τηλέφωνο» και «smartphone», μπορούν να αναγνωριστούν ως σημασιολογικά ισοδύναμες, ακόμη και όταν δεν μοιράζονται κοινές λέξεις.

2.3.1 Αναπαράσταση κειμένου σε διανύσματα (embeddings).

Τα embeddings δημιουργούνται μέσω εξειδικευμένων γλωσσικών μοντέλων τα οποία εκπαιδεύονται σε μεγάλης κλίμακας σύνολα δεδομένων. Κατά τη διαδικασία αυτή, το

μοντέλο μαθαίνει να τοποθετεί στον ίδιο διανυσματικό χώρο λέξεις και προτάσεις με παρόμοιο νόημα. Ως αποτέλεσμα, έννοιες που σχετίζονται σημασιολογικά καταλαμβάνουν γειτονικές θέσεις στον χώρο των διανυσμάτων.

Στη σύγχρονη βιβλιογραφία χρησιμοποιούνται ευρέως μοντέλα όπως τα BERT, Sentence-BERT (SBERT), MiniLM και νεότερες αρχιτεκτονικές που παράγουν υψηλής ποιότητας embeddings για εφαρμογές αναζήτησης, ταξινόμησης και αντιστοίχισης δεδομένων. Η διάσταση των διανυσμάτων κυμαίνεται συνήθως από μερικές εκατοντάδες έως αρκετές χιλιάδες χαρακτηριστικά, ανάλογα με το μοντέλο και τις απαιτήσεις της εφαρμογής.

Η εξέλιξη των embeddings τα τελευταία χρόνια έχει οδηγήσει στη δημιουργία αναπαραστάσεων που αποτυπώνουν με μεγαλύτερη ακρίβεια τη σημασιολογική πληροφορία, επιτρέποντας την αποτελεσματική σύγκριση κειμένων ακόμη και όταν οι διατυπώσεις διαφέρουν σημαντικά σε λεξιλογικό επίπεδο. Η δυνατότητα αυτή αποτελεί βασικό συστατικό σύγχρονων συστημάτων αναζήτησης γνώσης, recommendation systems και εφαρμογών επιχειρησιακής αντιστοίχισης δεδομένων.

2.3.2 Ομοιότητα Συνημιτόνου (Cosine Similarity) και Προκλήσεις.

Η συνηθέστερη μέθοδος σύγκρισης δύο embeddings είναι η Ομοιότητα Συνημιτόνου (Cosine Similarity). Η μετρική αυτή υπολογίζει τη γωνιακή ομοιότητα μεταξύ δύο διανυσμάτων και εκφράζει τον βαθμό σημασιολογικής τους συνάφειας.

$$\text{Cosine Similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Η τιμή της κυμαίνεται από -1 έως 1, όπου τιμές κοντά στο 1 υποδηλώνουν πολύ υψηλή σημασιολογική ομοιότητα. Το βασικό της πλεονέκτημα είναι ότι συγκρίνει την κατεύθυνση των διανυσμάτων και όχι το μέγεθός τους, γεγονός που την καθιστά ιδιαίτερα αποτελεσματική στην επεξεργασία κειμένου.

Παρά την ευρεία υιοθέτησή της, η χρήση της Cosine Similarity δεν στερείται περιορισμών. Σε ορισμένες περιπτώσεις, σημαντική πληροφορία μπορεί να κωδικοποιείται όχι μόνο στην κατεύθυνση αλλά και στο μήκος των διανυσμάτων. Επομένως, η αποκλειστική χρήση της συγκεκριμένης μετρικής ενδέχεται να οδηγήσει σε λανθασμένες αντιστοιχίσεις ή σε ψευδώς θετικά αποτελέσματα.

Για τον λόγο αυτό, σύγχρονες υλοποιήσεις συνδυάζουν την Cosine Similarity με πρόσθετες τεχνικές αξιολόγησης, όπως η Ευκλείδεια Απόσταση (Euclidean Distance), η χρήση κατωφλίων αξιοπιστίας (thresholds) και η αξιοποίηση επιπλέον επιχειρησιακών χαρακτηριστικών, όπως κατασκευαστής, κατηγορία προϊόντος ή μοναδικοί κωδικοί αναγνώρισης.

2.3.3 Semantic Matching σε Περιγραφές Προϊόντων και Βάσεις Διανυσμάτων.

Η σημασιολογική αντιστοίχιση (Semantic Matching) αποτελεί μία από τις σημαντικότερες πρακτικές εφαρμογές των embeddings σε επιχειρησιακά περιβάλλοντα. Στην πράξη, το ίδιο

προϊόν μπορεί να εμφανίζεται με διαφορετικές περιγραφές ανά προμηθευτή, διαφορετική γλώσσα ή εναλλακτικές εμπορικές ονομασίες.

Οι παραδοσιακές μέθοδοι σύγκρισης συμβολοσειρών, όπως οι Levenshtein Distance ή Jaccard Similarity, συχνά αδυνατούν να αναγνωρίσουν τέτοιες αντιστοιχίσεις όταν η λεκτική ομοιότητα είναι χαμηλή. Αντίθετα, τα embeddings επιτρέπουν την αξιολόγηση της σημασιολογικής σχέσης μεταξύ δύο περιγραφών, ανεξάρτητα από τις ακριβείς λέξεις που χρησιμοποιούνται.

Η τυπική διαδικασία semantic matching περιλαμβάνει:

1. Τη μετατροπή των υφιστάμενων περιγραφών σε embeddings.
2. Τη μετατροπή της νέας περιγραφής στο ίδιο διανυσματικό χώρο.
3. Τον υπολογισμό της ομοιότητας με τα διαθέσιμα embeddings.
4. Την επιλογή των επικρατέστερων υποψηφίων αντιστοιχίσεων.
5. Την τελική επιβεβαίωση μέσω πρόσθετων κανόνων ή μοντέλων μηχανικής μάθησης.

Όταν ο όγκος των δεδομένων ανέρχεται σε εκατομμύρια εγγραφές, η αποδοτική αναζήτηση απαιτεί εξειδικευμένες βάσεις διανυσμάτων (Vector Databases). Συστήματα όπως το Pinecone, το Milvus, το Weaviate και το Qdrant αξιοποιούν αλγορίθμους Approximate Nearest Neighbor (ANN), επιτρέποντας την ταχύτατη εύρεση των πιο κοντινών διανυσμάτων ακόμη και σε πολύ μεγάλα σύνολα δεδομένων.

Πίνακας 4 Ανάλυση αρχιτεκτονικής και βέλτιστων περιπτώσεων χρήσης των κύριων συστημάτων διαχείρισης διανυσματικών δεδομένων.

Βάση Αρχιτεκτονική / Βέλτιστη Περίπτωση Χρήσης
Δεδομένων Φιλοσοφία

Pinecone	Cloud-native, Fully Managed (SaaS)	Εφαρμογές παραγωγής που απαιτούν ταχύτητα ανάπτυξης (zero ops) με ισχυρό indexing. Το κόστος αυξάνεται στις μεγάλες κλίμακες.
Milvus	Open-source, Highly Scalable, Distributed	Ιδανικό για αναζητήσεις δισεκατομμυρίων διανυσμάτων, με υποστήριξη GPU acceleration. Απαιτεί εξειδίκευση στο στήσιμο (ops).
Weaviate	Open-source, Object-oriented, GraphQL	Βέλτιστο για Υβριδική Αναζήτηση (Hybrid Search), συνδυάζοντας σημασιολογική ομοιότητα με φιλτράρισμα βάσει μεταδεδομένων (metadata filtering).
Qdrant	Γραμμένο σε Rust, Performance-focused	Ανάγκες για υπερ-χαμηλό latency και high-throughput, με φιλικό open-source μοντέλο.

2.4 Υβριδικά Συστήματα Hybrid Systems: Γέφυρα μεταξύ LLM και ML

Η σύγχρονη τάση στην τεχνητή νοημοσύνη δεν εστιάζει πλέον στην αποκλειστική χρήση ενός μόνο μοντέλου, αλλά στον συνδυασμό διαφορετικών τεχνολογιών σε ενιαίες αρχιτεκτονικές. Τα υβριδικά συστήματα συνδυάζουν τις δυνατότητες κατανόησης φυσικής γλώσσας των LLMs με τη στατιστική ακρίβεια και την επαναληψιμότητα των αλγορίθμων μηχανικής μάθησης.

Η φιλοσοφία αυτή συνδέεται με το πεδίο της Neuro-Symbolic AI, όπου η μάθηση από δεδομένα συνδυάζεται με κανόνες λογικής, επιχειρησιακούς περιορισμούς και διαδικασίες επαλήθευσης.

2.4.1 Συνδυασμός στατιστικής-πιθανοτήτων και ντετερμινιστικών μεθόδων.

Τα LLMs είναι εξαιρετικά στην κατανόηση αδόμητου κειμένου, αλλά λειτουργούν με πιθανοτικό τρόπο. Αντίθετα, οι αλγόριθμοι μηχανικής μάθησης και οι επιχειρησιακοί κανόνες παρέχουν σταθερότητα, επαναληψιμότητα και ελεγχόμενη λήψη αποφάσεων.

Σε ένα τυπικό σύστημα αντιστοίχισης προϊόντων:

- Το LLM αναλύει και κανονικοποιεί την περιγραφή.
- Τα embeddings μετατρέπουν το κείμενο σε αριθμητικά χαρακτηριστικά.
- Το ML μοντέλο υπολογίζει πιθανότητες αντιστοίχισης.
- Οι επιχειρησιακοί κανόνες λαμβάνουν την τελική απόφαση.

Με αυτόν τον τρόπο αξιοποιούνται τα πλεονεκτήματα κάθε τεχνολογίας, ενώ περιορίζονται οι αδυναμίες της.

2.4.2 Πλεονεκτήματα στην Επιχειρησιακή Εφαρμογή.

Η υιοθέτηση υβριδικών αρχιτεκτονικών προσφέρει σημαντικά οφέλη:

- Μειωμένο ποσοστό σφαλμάτων και ψευδαισθήσεων.
- Υψηλότερη αξιοπιστία σε κρίσιμες διαδικασίες.
- Βελτιωμένη επεξηγησιμότητα των αποφάσεων.
- Καλύτερη αξιοποίηση υπολογιστικών πόρων.
- Δυνατότητα ανθρώπινης παρέμβασης σε αμφίβολες περιπτώσεις.

Τα χαρακτηριστικά αυτά είναι ιδιαίτερα σημαντικά σε περιβάλλοντα ERP, logistics, ηλεκτρονικού εμπορίου και διαχείρισης αποθεμάτων.

2.4.3 Ρόλος κανόνων (rules) και thresholds.

Η αυτοματοποίηση επιχειρησιακών διαδικασιών απαιτεί σαφείς μηχανισμούς ελέγχου. Για τον λόγο αυτό χρησιμοποιούνται κατώφλια εμπιστοσύνης (confidence thresholds), τα οποία καθορίζουν πότε μια απόφαση μπορεί να εκτελεστεί αυτόματα και πότε απαιτείται ανθρώπινος έλεγχος.

Μια συνηθισμένη προσέγγιση περιλαμβάνει:

- Ζώνη υψηλής βεβαιότητας, όπου η αντιστοίχιση πραγματοποιείται αυτόματα.
- Ενδιάμεση ζώνη, όπου απαιτείται δευτερογενής έλεγχος.
- Ζώνη χαμηλής βεβαιότητας, όπου η απόφαση παραπέμπεται σε χειροκίνητη αξιολόγηση.

Η προσέγγιση αυτή επιτρέπει υψηλά ποσοστά αυτοματοποίησης χωρίς να θυσιάζεται η αξιοπιστία των αποτελεσμάτων.

<https://arxiv.org/pdf/1908.10084>

(Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks)

--

2.5 Τεχνολογίες και βιβλιοθήκες υλοποίησης.

Η ανάπτυξη σύγχρονων υβριδικών συστημάτων Τεχνητής Νοημοσύνης προϋποθέτει τον συνδυασμό πολλών τεχνολογικών επιπέδων, τα οποία καλύπτουν τη διαχείριση δεδομένων, την εκπαίδευση μοντέλων, την επεξεργασία φυσικής γλώσσας και τη διασύνδεση με επιχειρησιακά συστήματα. Η αποτελεσματικότητα ενός τέτοιου οικοσυστήματος δεν εξαρτάται αποκλειστικά από την ποιότητα των αλγορίθμων, αλλά και από την κατάλληλη επιλογή γλωσσών προγραμματισμού, βιβλιοθηκών, βάσεων δεδομένων και πρωτοκόλλων επικοινωνίας. Τα τελευταία χρόνια, η εξέλιξη των εργαλείων αυτών έχει επιτρέψει την υλοποίηση ολοένα και πιο σύνθετων εφαρμογών, οι οποίες συνδυάζουν δυνατότητες μηχανικής μάθησης, σημασιολογικής αναζήτησης και αυτοματοποιημένης λήψης αποφάσεων.

2.5.1 Python, ML Frameworks και Πράκτορες (AI Agents).

Η Python αποτελεί σήμερα την κυρίαρχη γλώσσα προγραμματισμού στον χώρο της Τεχνητής Νοημοσύνης και της Μηχανικής Μάθησης. Η δημοτικότητά της οφείλεται στην απλότητα του συντακτικού της, στη μεγάλη κοινότητα χρηστών και κυρίως στο πλούσιο οικοσύστημα βιβλιοθηκών που διαθέτει.

Βιβλιοθήκες όπως η NumPy και η Pandas χρησιμοποιούνται για αριθμητικούς υπολογισμούς και επεξεργασία δομημένων δεδομένων, ενώ η scikit-learn παρέχει πληθώρα αλγορίθμων ταξινόμησης, παλινδρόμησης και ομαδοποίησης. Παράλληλα, πλατφόρμες όπως η XGBoost και η LightGBM έχουν καθιερωθεί ως πρότυπα για την ανάπτυξη μοντέλων υψηλής απόδοσης σε επιχειρησιακά δεδομένα.

Η εξέλιξη των Μεγάλων Γλωσσικών Μοντέλων οδήγησε στη δημιουργία μιας νέας κατηγορίας λογισμικού, των Πρακτόρων Τεχνητής Νοημοσύνης (AI Agents). Οι πράκτορες αυτοί δεν περιορίζονται στην παραγωγή απαντήσεων, αλλά μπορούν να εκτελούν σύνθετες ακολουθίες ενεργειών, όπως αναζήτηση πληροφοριών, πρόσβαση σε βάσεις δεδομένων, εκτέλεση κώδικα και λήψη αποφάσεων βάσει επιχειρησιακών κανόνων.

Για την υλοποίηση τέτοιων συστημάτων έχουν αναπτυχθεί εξειδικευμένα frameworks όπως το LangChain, το LlamaIndex και το PydanticAI. Τα εργαλεία αυτά επιτρέπουν τη δημιουργία ολοκληρωμένων ροών εργασίας, στις οποίες το μοντέλο αλληλεπιδρά με εξωτερικά δεδομένα, εκτελεί λογικούς ελέγχους και επιστρέφει δομημένα αποτελέσματα κατάλληλα για επιχειρησιακή αξιοποίηση.

2.5.2 Βάσεις Δεδομένων: Ο Μετασχηματισμός του SQL Server

Οι σχεσιακές βάσεις δεδομένων εξακολουθούν να αποτελούν τον πυρήνα των περισσότερων επιχειρησιακών πληροφοριακών συστημάτων. Η αποθήκευση, η ακεραιότητα και η διαχείριση μεγάλων όγκων δεδομένων εξακολουθούν να βασίζονται σε συστήματα RDBMS, με τον Microsoft SQL Server να συγκαταλέγεται μεταξύ των πιο διαδεδομένων λύσεων.

Τα τελευταία χρόνια παρατηρείται σταδιακή ενσωμάτωση δυνατοτήτων τεχνητής νοημοσύνης απευθείας στο επίπεδο της βάσης δεδομένων. Η εξέλιξη αυτή επιτρέπει την αποθήκευση και επεξεργασία διανυσματικών αναπαραστάσεων (embeddings), διευκολύνοντας τη σημασιολογική αναζήτηση χωρίς την ανάγκη συνεχούς μεταφοράς δεδομένων σε εξωτερικά συστήματα.

Η υποστήριξη διανυσματικών δεδομένων επιτρέπει την αποθήκευση embeddings μαζί με τα παραδοσιακά επιχειρησιακά δεδομένα, δημιουργώντας ένα ενοποιημένο περιβάλλον όπου μπορούν να συνδυαστούν κλασικά SQL ερωτήματα και σημασιολογικές αναζητήσεις. Η δυνατότητα αυτή είναι ιδιαίτερα σημαντική σε εφαρμογές semantic matching, recommendation systems και Retrieval-Augmented Generation (RAG).

Παράλληλα, η αυξανόμενη ενσωμάτωση υπηρεσιών τεχνητής νοημοσύνης σε περιβάλλοντα βάσεων δεδομένων μειώνει την ανάγκη για ενδιάμεσες εφαρμογές, απλοποιώντας τη συνολική αρχιτεκτονική και βελτιώνοντας την απόδοση των συστημάτων.

Πίνακας 5 εξέλιξη χαρακτηριστικών σημασιολογικής αναζήτησης, ενσωμάτωσης Γλωσσικών Μοντέλων και χειρισμού JSON στο οικοσύστημα του SQL Server.

Δυνατότητες	SQL Server 2022	SQL Server 2025
Σημασιολογική Αναζήτηση	Full-text Search, Regex μη εγγενής	Εγγενής τύπος VECTOR, υποστήριξη διανυσματικής αναζήτησης
Κλήση Γλωσσικών Μοντέλων	Μέσω εξωτερικών Python scripts, χρήση sp_execute_external_script	Απευθείας T-SQL κλήσεις (π.χ. NLP extraction) μέσω EXTERNAL MODEL

Δομημένα	JSON χειρισμός ως απλό κείμενο (string)	Νέος τύπος (Native JSON) με πλήρη
Δεδομένα (Κείμενο)		υποστήριξη Regex στο T-SQL

2.5.3 Χρήση Excel ως ενδιάμεσο εργαλείο επιχειρησιακής γνώσης.

Παρά την ανάπτυξη προηγμένων πλατφορμών ανάλυσης δεδομένων, το Microsoft Excel εξακολουθεί να αποτελεί ένα από τα σημαντικότερα εργαλεία στον επιχειρησιακό χώρο. Χρησιμοποιείται ευρέως για τη συλλογή, τον καθαρισμό, τον έλεγχο και την προκαταρκτική ανάλυση δεδομένων πριν αυτά εισαχθούν σε πιο εξειδικευμένα πληροφοριακά συστήματα.

Στην πράξη, μεγάλο μέρος των δεδομένων προμηθευτών, πελατών και αποθεμάτων διακινείται ακόμη μέσω αρχείων Excel ή CSV. Για τον λόγο αυτό, τα αρχεία αυτά λειτουργούν συχνά ως το πρώτο επίπεδο επεξεργασίας μέσα σε μια υβριδική αρχιτεκτονική τεχνητής νοημοσύνης.

Η αξιοποίηση του Excel επιτρέπει στους επιχειρησιακούς χρήστες να πραγματοποιούν βασικούς ελέγχους ποιότητας δεδομένων, να διορθώνουν σφάλματα και να επιβεβαιώνουν αποτελέσματα πριν την εκτέλεση πιο σύνθετων διαδικασιών μηχανικής μάθησης ή σημασιολογικής ανάλυσης. Με τον τρόπο αυτό διατηρείται η ανθρώπινη εποπτεία σε κρίσιμα στάδια της διαδικασίας.

2.5.4 APIs και διασύνδεση εξωτερικών συστημάτων.

Η λειτουργία ενός σύγχρονου υβριδικού συστήματος απαιτεί συνεχή επικοινωνία μεταξύ διαφορετικών εφαρμογών, υπηρεσιών και βάσεων δεδομένων. Η επικοινωνία αυτή υλοποιείται συνήθως μέσω διεπαφών προγραμματισμού εφαρμογών (Application Programming Interfaces – APIs).

Τα REST APIs έχουν καθιερωθεί ως το επικρατέστερο πρότυπο διασύνδεσης λόγω της απλότητας, της επεκτασιμότητας και της ευρείας υποστήριξής τους. Μέσω αυτών, ένα σύστημα μπορεί να ανταλλάσσει δεδομένα με ERP πλατφόρμες, συστήματα διαχείρισης αποθεμάτων, ηλεκτρονικά καταστήματα και υπηρεσίες τεχνητής νοημοσύνης.

Ιδιαίτερη σημασία αποκτά η ασύγχρονη επεξεργασία (asynchronous processing), καθώς οι λειτουργίες που σχετίζονται με μεγάλα γλωσσικά μοντέλα απαιτούν συνήθως περισσότερο χρόνο σε σχέση με τα παραδοσιακά πληροφοριακά συστήματα. Για τον λόγο αυτό, πολλές

εφαρμογές υιοθετούν μηχανισμούς ουρών εργασίας (job queues) και background workers ώστε να βελτιώνεται η εμπειρία χρήστη και να αποφεύγονται καθυστερήσεις.

Μία από τις σημαντικότερες εξελίξεις των τελευταίων ετών είναι η εμφάνιση του Model Context Protocol (MCP). Το πρωτόκολλο αυτό δημιουργήθηκε με στόχο την τυποποίηση της επικοινωνίας μεταξύ γλωσσικών μοντέλων και εξωτερικών πηγών δεδομένων. Μέσω του MCP, ένα μοντέλο μπορεί να αλληλεπιδρά με βάσεις δεδομένων, αποθετήρια εγγράφων, εταιρικά APIs και άλλες υπηρεσίες χρησιμοποιώντας ένα ενιαίο και ασφαλές πρότυπο επικοινωνίας.

Η υιοθέτηση τέτοιων προτύπων αναμένεται να διαδραματίσει καθοριστικό ρόλο στην ανάπτυξη της επόμενης γενιάς υβριδικών συστημάτων, όπου τα γλωσσικά μοντέλα δεν θα λειτουργούν απομονωμένα αλλά ως ενεργά μέλη ενός ολοκληρωμένου επιχειρησιακού οικοσυστήματος.

<https://www.anthropic.com/engineering/building-effective-agents>

Building Effective AI Agents

3. Υλοποίηση & Εφαρμογή του Προτεινόμενου Συστήματος Αντιστοίχισης Προϊόντων.

Στο προηγούμενο κεφάλαιο παρουσιάστηκαν οι θεωρητικές έννοιες που σχετίζονται με τη Μηχανική Μάθηση (Machine Learning), τα Μεγάλα Γλωσσικά Μοντέλα (Large Language Models - LLMs), τα embeddings, τις τεχνικές σημασιολογικής αναζήτησης (Semantic Search) και τις σύγχρονες αρχιτεκτονικές πρακτόρων (Agent-Based Systems). Η κατανόηση των τεχνολογιών αυτών αποτελεί τη θεωρητική βάση για την ανάπτυξη προηγμένων συστημάτων λήψης αποφάσεων σε επιχειρησιακά περιβάλλοντα.

Στο παρόν κεφάλαιο παρουσιάζεται η σχεδίαση, η ανάπτυξη και η πειραματική υλοποίηση ενός υβριδικού συστήματος αντιστοίχισης προϊόντων (Hybrid Product Matching System), το οποίο αναπτύχθηκε με σκοπό την αυτοματοποίηση της διαδικασίας αναγνώρισης και αντιστοίχισης προϊόντων μεταξύ διαφορετικών πηγών δεδομένων. Το πρόβλημα αυτό εμφανίζεται συχνά σε εμπορικά πληροφοριακά συστήματα, όπου οι περιγραφές προϊόντων διαφέρουν σημαντικά μεταξύ προμηθευτών, ηλεκτρονικών καταστημάτων και εσωτερικών εταιρικών βάσεων δεδομένων.

Η προτεινόμενη λύση βασίζεται σε μία υβριδική αρχιτεκτονική η οποία συνδυάζει τεχνικές σημασιολογικής αναζήτησης μέσω embeddings, αλγορίθμους υπολογισμού ομοιότητας, μηχανισμούς μάθησης από ιστορικά δεδομένα, επιχειρησιακούς κανόνες και διαδικασίες επαλήθευσης μέσω Μεγάλων Γλωσσικών Μοντέλων. Η προσέγγιση αυτή επιλέχθηκε διότι η αποκλειστική χρήση είτε κλασικών αλγορίθμων είτε LLMs δεν επαρκεί από μόνη της για την παραγωγή αξιόπιστων αποτελεσμάτων σε περιβάλλον παραγωγής.

Η υλοποίηση πραγματοποιήθηκε χρησιμοποιώντας Python ως κύρια γλώσσα προγραμματισμού, SQL Server ως σύστημα διαχείρισης βάσης δεδομένων και αρχεία Excel ως ενδιάμεσο μέσο ανταλλαγής και επαλήθευσης δεδομένων. Επιπλέον, αναπτύχθηκαν αυτοματοποιημένοι μηχανισμοί συλλογής δεδομένων από προμηθευτές, διαδικασίες μετασχηματισμού δεδομένων (ETL), μηχανισμοί δημιουργίας και αποθήκευσης embeddings, καθώς και υποσυστήματα λήψης αποφάσεων βασισμένα σε κανόνες και τεχνικές τεχνητής νοημοσύνης.

Η συνολική αρχιτεκτονική ακολουθεί τις αρχές των σύγχρονων Agentic AI συστημάτων, όπου επιμέρους εξειδικευμένοι πράκτορες συνεργάζονται για την εκτέλεση διακριτών εργασιών, όπως η ανάκτηση δεδομένων, η δημιουργία embeddings, η αξιολόγηση ομοιότητας, η λογική επαλήθευση και η τελική λήψη απόφασης. Με τον τρόπο αυτό επιτυγχάνεται υψηλός βαθμός επεκτασιμότητας, ευελιξίας και συντηρησιμότητας του συστήματος.

Στις επόμενες ενότητες παρουσιάζονται αναλυτικά η αρχιτεκτονική του συστήματος, τα επιμέρους στάδια επεξεργασίας, οι τεχνικές που χρησιμοποιήθηκαν, καθώς και πρακτικά παραδείγματα λειτουργίας σε πραγματικά δεδομένα προϊόντων. Ιδιαίτερη έμφαση δίνεται στην αλληλεπίδραση μεταξύ τεχνικών Μηχανικής Μάθησης, Μεγάλων Γλωσσικών Μοντέλων και επιχειρησιακών κανόνων, καθώς αυτή αποτελεί τον βασικό παράγοντα που καθορίζει την ακρίβεια και την αξιοπιστία του προτεινόμενου συστήματος.

3.1 Συνολική αρχιτεκτονική συστήματος.

Η παρούσα εργασία υλοποιεί ένα υβριδικό σύστημα αντιστοίχισης προϊόντων, το οποίο συνδυάζει τεχνικές Μηχανικής Μάθησης (Machine Learning), Μεγάλα Γλωσσικά Μοντέλα (Large Language Models - LLMs), επιχειρησιακούς κανόνες και μηχανισμούς αυτοματοποίησης δεδομένων.

Κατά τη διάρκεια της πειραματικής υλοποίησης διαπιστώθηκε ότι η αποκλειστική χρήση ενός LLM δεν επαρκεί για την παραγωγή αξιόπιστων αποτελεσμάτων σε περιβάλλον παραγωγής. Παρότι τα LLMs διαθέτουν ισχυρές δυνατότητες κατανόησης φυσικής γλώσσας, βασίζονται σε πιθανοκρατικές διαδικασίες και συνεπώς δεν αποτελούν ντετερμινιστικούς αλγόριθμους.

Για τον λόγο αυτό σχεδιάστηκε μία υβριδική αρχιτεκτονική η οποία συνδυάζει:

- Semantic Matching μέσω embeddings.
- Cosine Similarity.
- Επιχειρησιακούς κανόνες (Business Rules).
- Thresholds αποδοχής.
- Έλεγχο αριθμητικών χαρακτηριστικών (Numeric Validation).
- Επαλήθευση μέσω LLM.
- Μηχανισμούς caching.
- SQL-based deterministic ελέγχους.

Η προσέγγιση αυτή επιτρέπει στο σύστημα να αξιοποιεί τα πλεονεκτήματα των LLMs χωρίς να εξαρτάται αποκλειστικά από αυτά. Συνεπώς, στο προτεινόμενο σύστημα τα LLMs αξιοποιούνται επικουρικά και όχι ως πηγή «αλήθειας».

End-to-End Αρχιτεκτονική Ροής Δεδομένων:

Με τον όρο end-to-end αναφερόμαστε στη συνολική διαδρομή των δεδομένων: από τη στιγμή δημιουργίας τους έως τη λήψη επιχειρησιακής απόφασης.

Παράδειγμα ροής:

Προμηθευτής → Σύστημα → Ανάλυση → Απόφαση → Αυτοματοποίηση

Αναλυτικά στάδια:

1. Συλλογή Δεδομένων.

- Λήψη αρχείων προμηθευτών.
- Ανάκτηση εσωτερικών δεδομένων προϊόντων από SQL Server.
- Ενοποίηση διαφορετικών πηγών πληροφορίας.

2. Επεξεργασία Δεδομένων.

- Καθαρισμός περιγραφών (normalization).
- Αφαίρεση ειδικών χαρακτήρων.
- Ενοποίηση μορφών δεδομένων.
- Έλεγχος ελλειπόν τιμών.

3. Μετατροπή σε Embeddings.

- Μετατροπή περιγραφών-κειμένου σε διανυσματικές αναπαραστάσεις.

- Δημιουργία πίνακα χαρακτηριστικών.
- Χρήση caching για αποθήκευση embeddings ώστε:
να αποφεύγονται επαναλαμβανόμενοι υπολογισμοί,
να μειώνεται το κόστος API,
να αυξάνεται η ταχύτητα απόκρισης.

4. Semantic Matching.

- Υπολογισμός cosine similarity μεταξύ embeddings.
- Εντοπισμός πιθανότερου ProductID.
- Κατάταξη υποψηφίων αποτελεσμάτων.

5. Έλεγχος με Κανόνες και Thresholds.

- Ορισμός κατωφλίου ομοιότητας (π.χ. similarity > 0.80).
- Απόρριψη χαμηλής βεβαιότητας αντιστοιχίσεων.
- Εφαρμογή επιχειρησιακών κανόνων (π.χ. ίδια κατηγορία, ίδιο brand).

Ωστε το στάδιο αυτό λειτουργεί ως φίλτρο αξιοπιστίας.

6. LLM Verification.

- Περιορισμένο prompting.
- Ερώτηση τύπου Yes/No για επιβεβαίωση αντιστοίχισης.
- Έλεγχος συνέπειας περιγραφών.

Εδώ το LLM χρησιμοποιείται ως μηχανισμός λογικής επαλήθευσης και όχι αριθμητικού υπολογισμού.

7. Τελική Επιχειρησιακή Απόφαση.

- Αποδοχή ή απόρριψη αντιστοίχισης.
- Σύγκριση τιμών.
- Υποστήριξη κοστολόγησης.
- Υποβοήθηση παραγγελιών και προμηθειών.

Σε αυτό το στάδιο το σύστημα παράγει επιχειρησιακή αξία.

8. Αποθήκευση και Επαναχρησιμοποίηση Γνώσης.

- Ενημέρωση learning dataset.
- Αποθήκευση επιβεβαιωμένων αντιστοιχίσεων.
- Βελτίωση μελλοντικών προβλέψεων.
- Εμπλουτισμός δεδομένων εκπαίδευσης.

Η συνεχής ανατροφοδότηση (feedback loop) καθιστά το σύστημα ευέλικτο και εξελίξιμο.

Συμπερασματική Αρχιτεκτονική Προσέγγιση:

Το προτεινόμενο σύστημα δεν είναι αμιγώς LLM-based ούτε αποκλειστικά ML-based. Πρόκειται για υβριδικό σύστημα λήψης αποφάσεων που:

- Συνδυάζει πιθανοκρατικά μοντέλα (LLMs).

- Ενσωματώνει στατιστική μάθηση (ML).
- Εφαρμόζει ντετερμινιστικούς ελέγχους.
- Χρησιμοποιεί κανόνες και thresholds.
- Ελέγχει και επαληθεύει τα αποτελέσματα πριν από την τελική απόφαση.

Με αυτόν τον τρόπο επιτυγχάνεται ισορροπία μεταξύ ευφυΐας, ταχύτητας και αξιοπιστίας, στοιχείο απαραίτητο για επιχειρησιακά πληροφοριακά συστήματα.

3.1.1 Περιγραφή end-to-end ροής δεδομένων. Agents.

Η έννοια της end-to-end ροής δεδομένων αναφέρεται στη συνολική διαδρομή που ακολουθεί η πληροφορία από το στάδιο της συλλογής της έως τη λήψη επιχειρησιακής απόφασης. Στο προτεινόμενο σύστημα, η ροή αυτή είναι πλήρως αυτοματοποιημένη και αποτελείται από διακριτά αλλά αλληλοσυνδεόμενα στάδια.

Η γενική ροή μπορεί να αποδοθεί ως εξής:

Προμηθευτής → Συλλογή Δεδομένων → Μετασχηματισμός → Semantic Matching → Επαλήθευση → Απόφαση → Αποθήκευση Γνώσης.

Στάδιο 1: Συλλογή Δεδομένων:

Η συλλογή δεδομένων πραγματοποιείται μέσω αυτοματοποιημένου μηχανισμού (Supplier_Bot). Το σύστημα:

- Εκτελεί αυτοματοποιημένη σύνδεση (login) στην πύλη του προμηθευτή.
- Εντοπίζει και κατεβάζει το αρχείο XML προϊόντων.
- Μεταφέρει το αρχείο στο εταιρικό περιβάλλον.
- Ενεργοποιεί αποθηκευμένη διαδικασία (stored procedure) στον SQL Server για την επεξεργασία του.

Το στάδιο αυτό διασφαλίζει τη συνεχή και ελεγχόμενη τροφοδότηση του συστήματος με επικαιροποιημένα δεδομένα.

Στάδιο 2: Μετασχηματισμός και Ενοποίηση:

Η αποθηκευμένη διαδικασία XmlSupplier υλοποιεί ETL διαδικασία (Extract – Transform – Load):

- Ανάγνωση XML.
- Μετατροπή XML σε δομημένη μορφή μέσω XML Shredding και XQuery.

- Εφαρμογή επιχειρησιακής λογικής.
- Φόρτωση σε προσωρινό view για περαιτέρω χρήση.

Στη συνέχεια, μέσω Python (pandas & openpyxl), πραγματοποιείται:

- Καθαρισμός τιμών.
- Μετατροπή τιμών σε αριθμητική μορφή.
- Αναδιάταξη στηλών.
- Παραγωγή τελικού αρχείου XLSX.

Στάδιο 3: Διανυσματική Αναπαράσταση:

Οι περιγραφές προϊόντων μετατρέπονται σε embeddings μέσω του μοντέλου SentenceTransformer('all-MiniLM-L6-v2'). Το στάδιο αυτό επιτρέπει τη μετατροπή του φυσικού λόγου σε μαθηματικές αναπαραστάσεις.

Στάδιο 4: Semantic Matching:

Υπολογίζεται cosine similarity μεταξύ:

- Περιγραφών προμηθευτή.
- Εσωτερικής βάσης προϊόντων.
- Εντοπίζεται το πλησιέστερο προϊόν ώστε να κάνει match.

Στάδιο 5: Έλεγχος Κανόνων και Thresholds:

Το αποτέλεσμα περνά από filters-φίλτρα:

- Similarity threshold (π.χ. > 0.80)
- Επιχειρησιακούς κανόνες
- Έλεγχος αριθμητικών χαρακτηριστικών μέσω `def extract_numbers`.

Στάδιο 6: LLM Verification:

- Το LLM (Gemini 2.5 Flash) χρησιμοποιείται ως μηχανισμός Boolean λογικής επιβεβαίωσης (Yes/No) και σημασιολογικής επαλήθευσης.

Στάδιο 7: Τελική Απόφαση και Ανατροφοδότηση:

Η τελική απόφαση βασίζεται κυρίως στον κανόνα:

- Αν Similarity $> 80\%$ - 95% και η απόφαση του LLM = Yes $\rightarrow$ Match.
- Αν Similarity $\geq 95\%$ και συμφωνούν και οι αριθμοί τότε Yes. Διαφορετικά No.

- Αλλιώς → No Match δηλ. αν $\text{Similarity} \leq 80\%$.

Τα επιβεβαιωμένα αποτελέσματα αποθηκεύονται σε αρχείο Excel.

3.1.2 Αρχιτεκτονική Βασισμένη σε Πράκτορες (Agent-Based Architecture).

Η αρχιτεκτονική του συστήματος ακολουθεί τις σύγχρονες αρχές των Agentic AI Systems. Αντί της υλοποίησης ενός μονολιθικού λογισμικού, το σύστημα διασπάται σε επιμέρους λειτουργικές οντότητες (agents), καθεμία από τις οποίες είναι υπεύθυνη για συγκεκριμένο τμήμα της επεξεργασίας.

Η προσέγγιση αυτή προσφέρει:

- Διαχωρισμό ευθυνών (Separation of Concerns).
- Ευκολότερη συντήρηση.
- Δυνατότητα μελλοντικής επέκτασης.
- Ανεξάρτητο testing υποσυστημάτων.
- Βελτιωμένη επαναχρησιμοποίηση κώδικα.

Οι βασικοί πράκτορες του συστήματος είναι:

Data Retrieval Agent.

Υπεύθυνος για:

- Σύνδεση με SQL Server.
- Ανάκτηση προϊόντων.
- Ανάκτηση ιστορικών δεδομένων.
- Παροχή δομημένων δεδομένων στους επόμενους agents.

Embedding Agent.

Υπεύθυνος για:

- Δημιουργία embeddings.
- Διαχείριση cache embeddings.
- Batch processing.

Similarity Agent.

Υπεύθυνος για:

- Υπολογισμό cosine similarity.
- Εύρεση Top-K υποψηφίων αποτελεσμάτων.
- Κατάταξη πιθανών αντιστοιχίσεων.

LLM Reasoning Agent.

Υπεύθυνος για:

- Semantic verification.
- Natural Language Inference.
- Έλεγχο περιπτώσεων αβεβαιότητας.

Validation Agent.

Υπεύθυνος για:

- Εξαγωγή αριθμητικών χαρακτηριστικών από τις περιγραφές προϊόντων.

- Σύγκριση τεχνικών χαρακτηριστικών (χωρητικότητα, RAM, διαστάσεις, μοντέλα).
- Αποτροπή ψευδώς θετικών αντιστοιχίσεων που παρουσιάζουν υψηλή cosine similarity αλλά διαφορετικά αριθμητικά χαρακτηριστικά.

Decision Agent.

Υπεύθυνος για:

- Εφαρμογή επιχειρησιακών κανόνων και κατωφλίων (thresholds).
- Συνδυασμό αποτελεσμάτων από Similarity, Validation και LLM Reasoning Agents.
- Παραγωγή τελικής απόφασης.

Orchestrator Agent: Αποτελεί τον κεντρικό συντονιστή του συστήματος μέσω της main process_supplier_matching και συντονίζει τη σειρά εκτέλεσης όλων των υπολοίπων agents.

Βασική σχεδιαστική αρχή του προτεινόμενου συστήματος αποτελεί η επιλεκτική χρήση του Μεγάλου Γλωσσικού Μοντέλου. Το LLM δεν χρησιμοποιείται σε κάθε αντιστοίχιση, αλλά ενεργοποιείται μόνο όταν η μαθηματική αξιολόγηση μέσω cosine similarity δεν επαρκεί για την ασφαλή λήψη απόφασης ή όταν εντοπίζονται αποκλίσεις στα αριθμητικά χαρακτηριστικά των περιγραφών. Με τον τρόπο αυτό επιτυγχάνεται σημαντική μείωση του αριθμού των κλήσεων προς το LLM, περιορίζεται το υπολογιστικό κόστος και διατηρείται υψηλή αξιοπιστία στις τελικές αποφάσεις αντιστοίχισης.

3.1.3 Αρχιτεκτονική Τριών Σταδίων (Three-Stage Pipeline).

Η πρακτική υλοποίηση LLM που θα εξετάσουμε παρακάτω βασίζεται σε τρία κύρια στάδια.

- Στάδιο 1 – Προσομοίωση ΒΔ και αρχείου προμηθευτή.
- Στάδιο 2 – Μάθηση του μοντέλου και παραγωγή αποτελεσμάτων.
- Στάδιο 3 – Παραγωγή αποτελεσμάτων μέσω RAG.

3.2 Εφαρμογή Αντιστοίχισης Προϊόντων LLMs.

Στα πλαίσια της εν λόγω ακαδημαϊκής έρευνας και εργασίας για την αξιολόγηση του όλου συστήματος δημιουργήθηκε αρχικά ένα ελεγχόμενο περιβάλλον δοκιμών.

3.2.1 Δημιουργία Εικονικής ΒΔ και Εικονικού Αρχείου Προμηθευτή.

Στάδιο 1 – Προσομοίωση ΒΔ και αρχείου προμηθευτή:

Το πρώτο στάδιο είναι υπεύθυνο για τη δημιουργία και προετοιμασία των δεδομένων που θα χρησιμοποιηθούν από το υπόλοιπο σύστημα. Σε πραγματικό περιβάλλον παραγωγής, το στάδιο αυτό θα περιλάμβανε την ανάκτηση αρχείων XML, JSON ή CSV μέσω API ή άλλων

μηχανισμών διασύνδεσης με τους προμηθευτές, καθώς πιθανή μεταφορά δεδομένων μέσω sprou στον SQL Server για περαιτέρω επεξεργασία.

Για τις ανάγκες της παρούσας εργασίας, η διαδικασία αυτή προσομοιώνεται μέσω βοηθητικών εφαρμογών που αναπτύχθηκαν για ακαδημαϊκούς λόγους πειραματικά.

Αρχικά, το script createProducts.py δημιουργεί συνθετικά προϊόντα, όπως φορητούς υπολογιστές και έξυπνα κινητά τηλέφωνα, επιλέγοντας τυχαίους συνδυασμούς χαρακτηριστικών (κατασκευαστής, μοντέλο, επεξεργαστής, μνήμη RAM, αποθηκευτικός χώρος, κάρτα γραφικών, λειτουργικό σύστημα και τιμή). Τα προϊόντα αυτά αποθηκεύονται στον Microsoft SQL Server και αποτελούν την εσωτερική βάση αναφοράς του συστήματος.

Στη συνέχεια, το script createXmlNR.py ανακτά τυχαία προϊόντα από τη βάση δεδομένων και δημιουργεί ένα εικονικό αρχείο XML προμηθευτή. Κατά τη διαδικασία αυτή εφαρμόζονται τεχνικές παραμόρφωσης των δεδομένων (Data Perturbation), όπως αναδιάταξη χαρακτηριστικών, αλλαγή διαχωριστικών χαρακτήρων, διαφοροποίηση κεφαλαίων και πεζών χαρακτήρων, εισαγωγή πρόσθετων λέξεων και μικρές μεταβολές στις τιμές. Με τον τρόπο αυτό προσομοιώνονται οι πραγματικές διαφοροποιήσεις που συναντώνται στις περιγραφές προϊόντων διαφορετικών προμηθευτών. Τέλος το script XmlImport.py αναλαμβάνει την εισαγωγή των προϊόντων του προμηθευτή σε αρχείο excel LLMEmbMatch.xlsx το οποίο θα χρησιμοποιηθεί στα επόμενα στάδια για τις αντιστοιχίσεις και ως reporting του εμπορικού τμήματος.

Η δημιουργία του XML υλοποιείται με χρήση της Library API xml.etree.ElementTree, η οποία χρησιμοποιείται για τη δυναμική κατασκευή της ιεραρχικής δομής του XML μέσω των κλάσεων Element και SubElement. Στη συνέχεια, η βιβλιοθήκη xml.dom.minidom εφαρμόζει επαναφόρτωση (re-parsing) και μορφοποίηση (Pretty Printing), εξασφαλίζοντας σωστή εσοχή (indentation) και αναγνώσιμο XML αρχείο.

Με την παραπάνω διαδικασία δημιουργείται ένα πλήρως ελεγχόμενο πειραματικό περιβάλλον, το οποίο προσομοιώνει ρεαλιστικά την άφιξη δεδομένων από εξωτερικό προμηθευτή, επιτρέποντας την αξιολόγηση του προτεινόμενου συστήματος υπό συνθήκες που προσεγγίζουν πραγματικά επιχειρησιακά σενάρια.

```
<?xml version="1.0" ?> <product> (Παράδειγμα XML SupplierProducts.xml)

    <name>MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB ram/256GB SSD/RTX3050/Win10 Home</name>

    <price>1754.66</price>
```

ID	Name (Παράδειγμα Table ProductsDemo σε Βάση ThesisIG)	Price
1	MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB/256GB SSD/RTX3050/Win10 Home	1859.12
2	MSI XPS 13, Ryzen 5 5500U/13.3" FHD/16GB/1TB SSD/Intel Iris Xe/Win11 Pro	1611.25
..		...
28	Samsung Galaxy S22 128GB 6.1" OLED Tensor G2 Android	255.00
..		...

```
# createProducts.py: create products in DB.
import random
import os
import pyodbc
from dotenv import load_dotenv
# LOAD ENV
load_dotenv("getenv.env")
# CONNECTION STRING
conn_str = (
    f"DRIVER={{SQL Server}};"
    f"SERVER={os.getenv('DB_SERVER')};"
    f"DATABASE={os.getenv('DB_DATABASE')};"
    "Trusted_Connection=yes;"
)

# DATA POOLS
brands_laptop = ["Dell", "HP", "Lenovo", "Asus", "Acer", "MSI"]
models_laptop = ["XPS 13", "Pavilion 15", "ThinkPad E15", "R06 Strix G15", "Aspire 5", "Katana GF66"]
cpus = ["i5-11350G", "i5-1235U", "i7-11650G", "i7-12700H", "Ryzen 5 5500U", "Ryzen 7 6800H"]
rams = ["8GB", "16GB", "32GB"]
storages = ["256GB SSD", "512GB SSD", "1TB SSD"]
screens = ["13.3" FHD", "14" FHD", "15.6" FHD 144Hz"]
gpus = ["Intel Iris Xe", "AMD Radeon", "RTX3050", "RTX3060"]
oses = ["Win10 Home", "Win11 Home", "Win11 Pro"]
brands_phone = ["Apple", "Samsung", "Xiaomi", "Google"]
models_phone = ["iPhone 13", "iPhone 14 Pro", "Galaxy S22", "Galaxy S23 Ultra", "Pixel 7", "Redmi Note 12"]
storage_phone = ["128GB", "256GB", "512GB"]
screens_phone = ["6.1" OLED", "6.7" AMOLED 120Hz", "6.8" AMOLED"]
cpus_phone = ["A15 Bionic", "A16 Bionic", "Snapdragon 8 Gen 2", "Tensor G2"]

# GENERATE PRODUCTS
def generate_products(n=10):
    products = []
    for _ in range(n):
        if random.random() < 0.5:
            # Laptop
            name = f"{random.choice(brands_laptop)} {random.choice(models_laptop)}, " \
                f"{random.choice(cpus)}/{random.choice(screens)}/{random.choice(rams)}/" \
                f"{random.choice(storages)}/{random.choice(gpus)}/{random.choice(oses)}"
            price = round(random.uniform(600, 2000), 2)
        else:
            # Smartphone
            name = f"{random.choice(brands_phone)} {random.choice(models_phone)}, " \
                f"{random.choice(storage_phone)}, {random.choice(screens_phone)}, " \
                f"{random.choice(cpus_phone)}, Android"
            price = round(random.uniform(200, 1400), 2)
        products.append((name, price)) # tuple
    return products

# INSERT INTO SQL
def insert_products(products):
    conn = pyodbc.connect(conn_str)
    cursor = conn.cursor()
    query = "INSERT INTO ProductsDemo (Name, Price) VALUES (?, ?)"
    for p in products:
        cursor.execute(query, p) # p = (name, price)
    conn.commit()
    cursor.close()
    conn.close()
    print(f"Inserted {len(products)} products into SQL Server")

# MAIN
def main():
    products = generate_products(10)
    # Debug print (optional)
    insert_products(products)
if __name__ == "__main__":
    main()

# Note, before run. Sql:
# CREATE TABLE ProductsDemo (
#     ID INT IDENTITY(1,1) PRIMARY KEY,
#     Name NVARCHAR(500),
#     Price DECIMAL(10,2)
# );
```

```
#createXmlNR.py: create Xml from supplier.
import os
import random
import pyodbc
import xml.etree.ElementTree as ET
from xml.dom import minidom
from dotenv import load_dotenv
# ===== CONFIG =====
load_dotenv("getenv.env")
OUTPUT_FILE = "SupplierProducts.xml"
# ===== SQL =====
conn_str = (
    f"DRIVER={{SQL Server}};"
    f"SERVER={os.getenv('DB_SERVER')};"
    f"DATABASE={os.getenv('DB_DATABASE')};"
    "Trusted_Connection=yes;"
)
def fetch_random_products(limit=5):
    """Fetches random rows from the database."""
    products = []
    try:
        conn = pyodbc.connect(conn_str)
        cursor = conn.cursor()

        # NEWID() efficiently randomizes the row selection in SQL Server
        query = f"""
            SELECT TOP {limit} [Name], [Price]
            FROM [ThesisIG].[dbo].[ProductsDemo]
            ORDER BY NEWID()
        """
        cursor.execute(query)

        for row in cursor.fetchall():
            # Ensure price is handled as a float
            products.append({"name": str(row.Name), "price": float(row.Price)})
        conn.close()
    except Exception as e:
        print(f"✗ Database error: {e}")
    return products
# ===== DATA MUTATION =====
def perturb_data(name, price):
    """Simulates messy supplier formatting and price variations."""
    # 1. Price Modification (+/- 10%)
    variance = random.uniform(0.90, 1.10)
    new_price = round(price * variance, 2)
    # 2. Name Modification (String Obfuscation)
    new_name = name
    # Swap common separators (e.g., turn ' - ' or ' / '
    separators = [",", " ", " - ", " / "]
    if " " in new_name:
        parts = [p.strip() for p in new_name.split(",")]
        # 50% chance to shuffle the attributes (keeping the Brand/Model at the front)
        if len(parts) > 2 and random.random() < 0.5:
            tail = parts[1:]
            random.shuffle(tail)
            parts = [parts[0]] + tail
        new_name = random.choice(separators).join(parts)
    # 50% chance to swap the first two words (e.g., "Google iPhone" → "iPhone google")
    words = new_name.split()
    if len(words) ≥ 2 and random.random() < 0.5:
        # Swap and lowercase to simulate human error
        words[0], words[1] = words[1].lower(), words[0].lower()
        new_name = " ".join(words)
    # 30% chance to inject noise words (e.g., "ram" next to "GB")
    if "GB" in new_name and random.random() < 0.3:
        new_name = new_name.replace("GB", "GB ram", 1)
    return new_name, new_price
# ===== MAIN =====
def main():
    print("Fetching data from SQL Server...")
    db_products = fetch_random_products(5)
    if not db_products:
        print("⚠ No products found. Exiting.")
        return
    # Initialize XML Tree
    root = ET.Element("shop")
    products_el = ET.SubElement(root, "products")
    for p in db_products:
        # Apply the chaotic formatting
        mod_name, mod_price = perturb_data(p["name"], p["price"])
        # Build XML nodes
        product_el = ET.SubElement(products_el, "product")
        name_el = ET.SubElement(product_el, "name")
        name_el.text = mod_name
        price_el = ET.SubElement(product_el, "price")
        price_el.text = str(mod_price)
    # Format and Save XML
    rough_string = ET.tostring(root, 'utf-8')
    reparsed = minidom.parseString(rough_string)
    pretty_xml = reparsed.toprettyxml(indent=" ")
    with open(OUTPUT_FILE, "w", encoding="utf-8") as f:
        f.write(pretty_xml)
    print(f"✅ XML file successfully created: {OUTPUT_FILE}")
if __name__ == "__main__":
    main()
```

```
# XmlImport.py: import Xml in xlsx file.
import os
import xml.etree.ElementTree as ET
from openpyxl import load_workbook
# PATHS
BASE_PATH = r"C:\Users\igIT\Desktop\ig\Edu\Thesis\0code"
XML_PATH = os.path.join(BASE_PATH, "SupplierProducts.xml")
EXCEL_PATH = os.path.join(BASE_PATH, "LLMEmbMatch.xlsx")
SHEET_NAME = "ai"
# STEP 1: READ XML
def read_xml_names(xml_path):
    tree = ET.parse(xml_path)
    root = tree.getroot()
    names = []
    # path: shop/products/product
    for product in root.findall(".//product"):
        name = product.findtext("name")
        if name:
            names.append(name.strip())
    return names
# STEP 2: WRITE TO EXCEL
def write_to_excel(names, excel_path):
    if os.path.exists(excel_path):
        wb = load_workbook(excel_path)
    else:
        from openpyxl import Workbook
        wb = Workbook()
    if SHEET_NAME in wb.sheetnames:
        ws = wb[SHEET_NAME]
    else:
        ws = wb.create_sheet(SHEET_NAME)
    ws["A1"] = "Import"
    # Καθαρισμός παλιών δεδομένων (προαιρετικό)
    ws.delete_rows(2, ws.max_row)
    # Γράψιμο από A2
    row = 2
    for name in names:
        ws[f"A{row}"] = name
        row += 1
    wb.save(excel_path)
    print(f"Written {len(names)} rows to Excel: {excel_path}")
# MAIN
def main():
    names = read_xml_names(XML_PATH)
    print(f"Found {len(names)} products")
    write_to_excel(names, EXCEL_PATH)
if __name__ == "__main__":
    main()
```

3.2.2 Εφαρμογή αντιστοίχισης προϊόντων χωρίς σύνδεση με τη ΒΔ.

Στάδιο 2 – Μάθηση του μοντέλου και παραγωγή αποτελεσμάτων:

Το δεύτερο στάδιο αφορά κυρίως τη διαδικασία μάθησης script: Train_productid.py.

Στο στάδιο αυτό το σύστημα δημιουργεί μία βάση γνώσης από ιστορικά δεδομένα προϊόντων.

Η διαδικασία περιλαμβάνει:

- Δημιουργία embeddings.
- Υπολογισμό cosine similarity.
- Εφαρμογή διπλού threshold <80% με απόρριψη και ≥95% με αποδοχή.
- Δημιουργία learning dataset.

Πρόκειται για example based retrieval βασισμένα σε παραδείγματα αντιστοίχισης και όχι για εκπαίδευση νευρωνικού δικτύου. Χρησιμοποιούνται ιστορικά επιβεβαιωμένες αντιστοιχήσεις ως knowledge base μέσω cosine similarity.

Το αποτέλεσμα του σταδίου είναι η δημιουργία ενός συνόλου επιβεβαιωμένων αντιστοιχίσεων το οποίο χρησιμοποιείται αργότερα ως learning override.

Ο συγκεκριμένος κώδικας υλοποιεί την προτεινόμενη αρχιτεκτονική: ένα Υβριδικό Σύστημα Ταύτισης Οντοτήτων (Hybrid Entity Resolution System), το οποίο γεφυρώνει την κλασική Μηχανική Μάθησης (ML με MiniLM + Cosine Similarity) και την αναλυτική ικανότητα των Μεγάλων Γλωσσικών Μοντέλων (LLMs με Gemini 2.5 Flash). Ο σχεδιασμός αυτός επιτρέπει την ελαχιστοποίηση των κλήσεων στο LLM και άρα του κόστους, διατηρώντας παράλληλα υψηλή ακρίβεια σε οριακές περιπτώσεις.

1. Διανυσματοποίηση & Διαχείριση Πόρων (Embedding & Caching)

Το πρώτο στάδιο της ροής δεδομένων (data pipeline) αφορά τη σημασιολογική μετατροπή του κειμένου.

Vectorization (Διανυσματοποίηση): Αξιοποιείται η βιβλιοθήκη sentence-transformers και συγκεκριμένα το μοντέλο all-MiniLM-L6-v2. Το μοντέλο αυτό επιλέχθηκε λόγω της ισορροπίας μεταξύ ταχύτητας και ακρίβειας (latency-to-accuracy ratio), μετατρέποντας τις λεκτικές περιγραφές των προϊόντων σε dense vector embeddings.

Persistent Caching (Επίμονη Μνήμη): Για την ελαχιστοποίηση του υπολογιστικού κόστους και των κλήσεων στο API, υλοποιείται ένας μηχανισμός τοπικής αποθήκευσης μέσω της βιβλιοθήκης `joblib` (`embedding_cache_local.pkl` και `gemini_cache.pkl`). Ο αλγόριθμος εκτελεί πρώτα έναν έλεγχο (`lookup`) στη μνήμη (`hash map`) και προχωρά σε `inference` (είτε του `MiniLM` είτε του `Gemini`) μόνο αν η εγγραφή δεν υφίσταται (`cache miss`).

2. Μαθηματική Αξιολόγηση (Cosine Similarity Matching)

Η συνάρτηση `find_best_match` αναλαμβάνει το σκέλος της ταύτισης.

Διανυσματική Απόσταση: Υπολογίζεται η Ομοιότητα Συνημιτόνου (Cosine Similarity) μέσω του `framework` της `scikit-learn` (`sklearn.metrics.pairwise`). Το σύστημα συγκρίνει το διάνυσμα του εισερχόμενου προϊόντος με τον πίνακα διανυσμάτων (`matrix`) ολόκληρης της βάσης προϊόντων (Knowledge Base).

Nearest Neighbor (Κοντινότερος Γείτονας): Μέσω της συνάρτησης `np.argmax` της βιβλιοθήκης `NumPy`, το σύστημα εντοπίζει ακαριαία το `index` με το μέγιστο μαθηματικό σκορ (`best score`) και επιστρέφει το αντίστοιχο `ProductID`.

3. Ενорχήστρωση LLM (LLM Orchestration & Prompting)

Η συνάρτηση `ask_gemini_yes_no` λειτουργεί ως ο "έξυπνος πράκτορας" (AI Agent) του συστήματος, επικοινωνώντας με το `Gemini 2.5 Flash` μέσω API.

Zero-Shot Prompting με Strict Constraints: Το `prompt` είναι σχεδιασμένο να περιορίζει τις «παραισθήσεις» (`hallucinations`). Δίνονται αυστηροί κανόνες (`rules`) και ζητείται δυαδική απάντηση (`Binary Output: Yes / No`). Αυτό μετατρέπει το παραγωγικό μοντέλο σε έναν αυστηρό Classifier (ταξινομητή).

Fault Tolerance (Ανοχή σε Σφάλματα): Ενσωματώνεται μηχανισμός `Exception Handling` με δομή επαναλήψεων (`retry loop for in range(3)`) και `time.sleep(45)` για την αποφυγή απόρριψης λόγω υπερφόρτωσης του δικτύου (`Rate Limiting - HTTP Error 429`).

4. Η Κεντρική Λογική Ελέγχου (The Iteration Loop & Risk-Adaptive Triage)

Το σημαντικότερο τμήμα του κώδικα (εκεί που ενώνεται η ML με το LLM) βρίσκεται στον κύριο βρόχο επανάληψης (`for i, desc in enumerate(...)`):

Ο αλγόριθμος διατρέχει (`iterates`) σειριακά ένα προς ένα τα νέα προϊόντα. Για κάθε προϊόν, βρίσκει το καλύτερο μαθηματικό ταίριασμα (`Score`). Εδώ εφαρμόζεται το θεωρητικό

μοντέλο του Risk-Adaptive Triage (Διαχωρισμός βάσει Κινδύνου), ελέγχοντας το Score μέσα από δομές if/elif/else:

Hard Rejection (Score < 0.80): Αν η μαθηματική ομοιότητα είναι χαμηλή, ο αλγόριθμος απορρίπτει την ταύτιση αυτόματα (verdict = "No"). Το LLM δεν καλείται ποτέ, εξοικονομώντας χρόνο και χρήμα.

Hard Acceptance (Score >= 0.95): Αν η ομοιότητα είναι συντριπτική, η ταύτιση γίνεται αυτόματα δεκτή (verdict = "Yes"), βασιζόμενη εξ ολοκλήρου στην επεξηγησιμότητα της κλασικής ML.

Grey Zone Escalation (0.80 <= Score < 0.95): Αυτή είναι η κρίσιμη συνθήκη. Όταν το ML μοντέλο είναι "αναποφάσιστο", το σύστημα κάνει escalate (παραπομπή) το πρόβλημα. Καλεί τη συνάρτηση ask_gemini_yes_no, περνώντας τις δύο περιγραφές στο LLM, ώστε να κρίνει με βάση τη σημασιολογική του κατανόηση (αν π.χ. η διαφορά είναι απλώς ένα τυπογραφικό λάθος ή μια ουσιαστική διαφορά στα χαρακτηριστικά).

Συμπερασματικά: Με την ολοκλήρωση του βρόχου, το δομημένο σύνολο αποτελεσμάτων γίνεται serialize μέσω της pandas και εξάγεται σε μορφή Excel (df_results.to_excel), παραδίδοντας έναν καθαρό πίνακα που περιέχει την αρχική περιγραφή, το προτεινόμενο ProductID, το μαθηματικό Score, και την τελική ετυμηγορία του συστήματος (Final).

```
C:\Users\igIT\AppData\Local\Programs\Python\Python313\python.exe C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\Train_productid.py
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
Loading weights: 100%|██████████| 103/103 [00:00<00:00, 2399.14it/s]
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key | Status | |
-----+-----+---
embeddings.position_ids | UNEXPECTED | |

Notes:
- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.
BASE rows: 6
Product DB loaded: 6
PLUS rows: 100
Matching: 100%|██████████| 100/100 [00:00<00:00, 1855.72it/s]
Saved: C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\LLMEmbMatch_matched.xlsx
Process finished with exit code 0
```

Σε αυτό το script εστιάζουμε κυρίως να μάθουμε το μοντέλο μας ώστε να μπορεί να μετρήσει πόσο ομοιάζουν οι περιγραφές. Supervised learning χωρίς neural network. Κάνει cosine_similarity π.χ. iphone 14 pro με apple iphone 14 pro max similarity = 95% δηλ. σχεδόν ίδια άρα τα κάνει match. Αντίθετα αν ήταν κάτω από 80% θα τα θεωρούσε διαφορετικά. Έχουμε ορίσει για σίγουρο match πάνω από το 95%, ενδιάμεσα ρωτάει το LLM το οποίο και αποφασίζει. def find_best_match

Σε αυτό το script και μέσω τεχνικών Μηχανικής Μάθησης (Machine Learning) ML προσπαθούμε να ταιριάζουμε προϊόντα-περιγραφές διαφορετικών βάσεων δεδομένων.

Συγκεκριμένα, χρησιμοποιούμε το μοντέλο Large Language Model (LLM) SentenceTransformer('all-MiniLM-L6-v2') για τη δημιουργία διανυσματικών αναπαραστάσεων (Embeddings) των περιγραφών των προϊόντων. Η διαδικασία αυτή επιτρέπει τη μετατροπή του κειμένου σε μαθηματικές οντότητες που εμπεριέχουν τη σημασιολογική πληροφορία (semantic context).

Η σύγκριση των προϊόντων πραγματοποιείται μέσω της Cosine Similarity (Συνημίτονο Ομοιότητας) η οποία χρησιμοποιεί μαθηματικά διανυσμάτων για να βρει την "απόσταση" μεταξύ των εννοιών,

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

επιτυγχάνοντας αυτόματο ταίριασμα (matching) προϊόντων ακόμα και όταν οι περιγραφές διαφέρουν λεκτικά.

Το σύστημα ενσωματώνει μηχανισμούς τοπικής προσωρινής αποθήκευσης (Caching) για τη βελτιστοποίηση των υπολογιστικών πόρων και την ελαχιστοποίηση του κόστους χρήσης εξωτερικών APIs, διασφαλίζοντας παράλληλα υψηλή ακρίβεια μέσω καθορισμένων κατωφλίων αποδοχής (thresholding).

Εστιάζοντας στο τεχνικό του κομμάτι χρησιμοποιείται Vector Caching ώστε να μειώνονται τα API credits που κοστίζουν χρήματα και χρόνο. καθώς το script "θυμάται" την απάντηση τοπικά.

Batching (Ομαδοποίηση): Το script στέλνει τα δεδομένα σε πακέτα (batches) των 2.000 εγγραφών. Αυτό μειώνει το overhead των δικτυακών κλήσεων και αποφεύγει τα όρια (rate limits) του API.

Threshold Management: Ορίζοντας ένα threshold=0.8, εφαρμόζετε φίλτρο ποιότητας. Αν η ομοιότητα είναι χαμηλότερη από 80%, το σύστημα θεωρεί το αποτέλεσμα ανακριβές και δεν κάνει το ταίριασμα (No Matching), ελαχιστοποιώντας έτσι τα σφάλματα.

Data Persistence & Backup: Πριν γράψει στο κεντρικό Excel, δημιουργεί αντίγραφο (backup) ώστε να μην χαθούν δεδομένα σε περίπτωση αποτυχίας.

Συνοπτικά το παρόν script:

1. Φορτώνει υπάρχουσες γνώσεις (cache).
2. Μετατρέπει τις περιγραφές των προϊόντων σε διανύσματα (embeddings).
3. Συγκρίνει τρίτων περιγραφές προϊόντων με τη δικής μας βάση δεδομένων της X.
4. Αποφασίζει για το αν ταιριάζουν οι περιγραφές με βάση τη def find_best_match.
5. Ενημερώνει το αρχείο Excel με τα Product IDs που βρήκε αυτόματα.

Ουσιαστικά συγκρίνονται τα προϊόντα του προμηθευτή τα οποία έχουμε ως import στο αρχείο LLMEmbMatch.xlsx με την εκπαίδευση του μοντέλου αρχείο LLMEmbPromptLearn.xlsx

Το σύστημα συγκρίνει τις περιγραφές προϊόντων του προμηθευτή (LLMEmbMatch.xlsx) με ένα σύνολο ήδη ταυτοποιημένων προϊόντων αναφοράς (LLMEmbPromptLearn.xlsx), χρησιμοποιώντας σημασιολογικά embeddings και cosine similarity. Στις περιπτώσεις αμφίβολης ομοιότητας, εφαρμόζεται επιπλέον έλεγχος από μεγάλο γλωσσικό μοντέλο (Gemini), ώστε να επιβεβαιωθεί αν πρόκειται για το ίδιο προϊόν και να αποδοθεί το κατάλληλο ProductID.

Το σύστημα δεν χρησιμοποιεί SQL database. Φορτώνει τα δεδομένα από Excel απευθείας στη μνήμη και συγκρίνει και σχεδιάστηκε για επιχειρησιακούς χρήστες που εργάζονται με Excel, χωρίς να απαιτείται καμία database υποδομή. Η είσοδος και έξοδος LLMEmbMatch_matched.xlsx είναι σε μόνο σε αρχεία .xlsx.

```
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
Loading weights: 100%|██████████| 103/103 [00:00<00:00, 1725.91it/s]
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key          | Status    | |
-----+-----+
embeddings.position_ids | UNEXPECTED | |

Notes:
- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.
BASE rows: 29
Product DB loaded: 29
PLUS rows: 5
Matching: 100%|██████████| 5/5 [00:00<00:00, 560.83it/s]
Saved: C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\LLMEmbMatch_matched.xlsx

Process finished with exit code 0
```

Πρακτικά παραδείγματα:

1.

Import	Matched ProductID	Score	Gemini	Final
Lenovo Aspire 5, Ryzen 7 6800H/13.3" FHD/8GB/512GB SSD/AMD Radeon Graphics/Win11 Pro	25	0,827899992	No	No
Xiaomi Pixel 7, 128GB, 6.8" AMOLED, A16 Bionic, Android	4	0,852699995	No	No
Dell XPS 13, Ryzen 7 5800H/15.6" FHD 144Hz/16GB/512GB SSD/Intel Iris Xe/Win11 Pro	23	0,980199993	Yes	Yes
Apple iPhone 13, 128GB, 6.7" AMOLED 120Hz, A15 Bionic, Android	24	0,966400027	Yes	Yes
Asus Aspire 5, Ryzen 7 5800H/15.6" FHD 144Hz/32GB/512GB SSD/NVIDIA RTX3060/Win11 Home	25	0,973100007	Yes	Yes
Google Galaxy S23 Ultra, 256GB, 6.1" OLED, A16 Bionic, Android	26	0,969600022	Yes	Yes
Samsung Galaxy S22, 512GB, 6.1" OLED, Tensor G2, Android	27	0,965300024	Yes	Yes
Samsung Galaxy S22, 128GB, 6.1" OLED, Tensor G2, Android	27	0,963900003	Yes	Yes
Lenovo XPS 13, i7-12700H/15.6" FHD 144Hz/16GB/1TB SSD/NVIDIA RTX3060/Win10 Home	29	0,965399981	Yes	Yes
Dell Aspire 5, i7-1165G7/15.6" FHD 144Hz/32GB/256GB SSD/NVIDIA RTX3050/Win11 Pro	30	0,976100028	Yes	Yes

Μετά από έλεγχο της βάσης επιβεβαιώνονται ως ορθά τα παραπάνω αποτελέσματα.

2.

Import	Matched ProductID	Score	Gemini	Final
MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB ram/256GB SSD/RTX3050/Win10 Home	1	0,986400008	Yes	Yes
Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB ram - 512GB SSD - NVIDIA RTX3060 - Win11 Home	25	0,986800015	Yes	Yes
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	0,993099988	Yes	Yes
galaxy samsung S22 128GB 6.1" OLED Tensor G2 Android	28	0,99879998	Yes	Yes
galaxy samsung S22 512GB 6.1" OLED Tensor G2 Android	27	0,998700023	Yes	Yes

Η αξιολόγηση του συστήματος έδειξε ότι η προτεινόμενη προσέγγιση σημασιολογικής αντιστοίχισης πέτυχε την ορθή ταυτοποίηση όλων των δοκιμαστικών προϊόντων. Οι τιμές ομοιότητας που προέκυψαν από τα embeddings κυμάνθηκαν από 98,64% έως 99,88%, υποδηλώνοντας πολύ υψηλό βαθμό σημασιολογικής συνάφειας μεταξύ των περιγραφών εισόδου και των προϊόντων αναφοράς. Το σύστημα κατάφερε να αναγνωρίσει επιτυχώς προϊόντα παρά τις διαφοροποιήσεις στη σύνταξη, στη σειρά των λέξεων και στην ύπαρξη πρόσθετων όρων, όπως η λέξη «ram» στις περιγραφές φορητών υπολογιστών. Επιπλέον, αντιμετώπισε αποτελεσματικά περιπτώσεις αναδιάταξης της ονομασίας του κατασκευαστή,

όπως στην περίπτωση «katana dell GF66» έναντι «Dell Katana GF66», καθώς και διαφορετική σειρά εμφάνισης των όρων σε συσκευές κινητής τηλεφωνίας, όπως «galaxy samsung S22» έναντι «Samsung Galaxy S22». Σε όλες τις περιπτώσεις η επαλήθευση από το μοντέλο Gemini κατέληξε σε θετική απόφαση (Yes), επιβεβαιώνοντας ότι οι υψηλές τιμές cosine similarity αντιστοιχούσαν πράγματι στο ίδιο προϊόν. Τα αποτελέσματα καταδεικνύουν ότι η χρήση embeddings σε συνδυασμό με μηχανισμό λογικής επαλήθευσης μέσω LLM μπορεί να επιτύχει υψηλή ακρίβεια αντιστοίχισης ακόμη και όταν οι περιγραφές δεν είναι λεκτικά ταυτόσημες.

3.

Import	Matched ProductID	Score	Gemini	Final
15.6" FHD 144Hz/16GB ram/256GB SSD/Win10 Home/RTX3050/ MSI Pavilion 15, Ryzen 7 6800H	1	0,950500011	Yes	Yes
Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 64GB ram - 512GB SSD - NVIDIA RTX3060 - Win11 Home	25	0,986800015	Yes	Yes
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	0,993099988	Yes	Yes
galaxy samsung S22 1028GB 6.1" OLED Tensor G2 Android	28	0,987500012	Yes	Yes
galaxy samsung S22 512GB 6.1" OLED Tensor G2 // Android	27	0,991699994	Yes	Yes

Πειράχτηκαν εσκεμμένα κάποιες περιγραφές ορθά αντιστοιχήθηκε η 1^η γραμμή με το id 1 αλλά διαπιστώθηκε σημαντικό σφάλμα αναντιστοιχίας με το id 28 Samsung Galaxy S22 | 128GB | 6.1" OLED | Tensor G2 | Android 28. Αλλάχθηκε η Ram σε 1028 γραμμή 4^η Δεν έπρεπε να γίνει match. Αυτό πιθανότατα συνέβη διότι το cosine similarity (semantic embeddings) δεν κατάλαβε την αριθμητική διαφορά καθώς δίνει έμφαση στη σημασιολογία. Παρεμβαίνοντας στον κώδικα και ορίζοντας νέα συνάρτηση def extract_numbers με το κατάλληλο if ώστε αν είναι πάνω από 95% αλλά δεν συμφωνούν οι αριθμοί να απευθύνεται στο LLM έχουμε ορθά αποτελέσματά όπως φαίνεται παρακάτω:

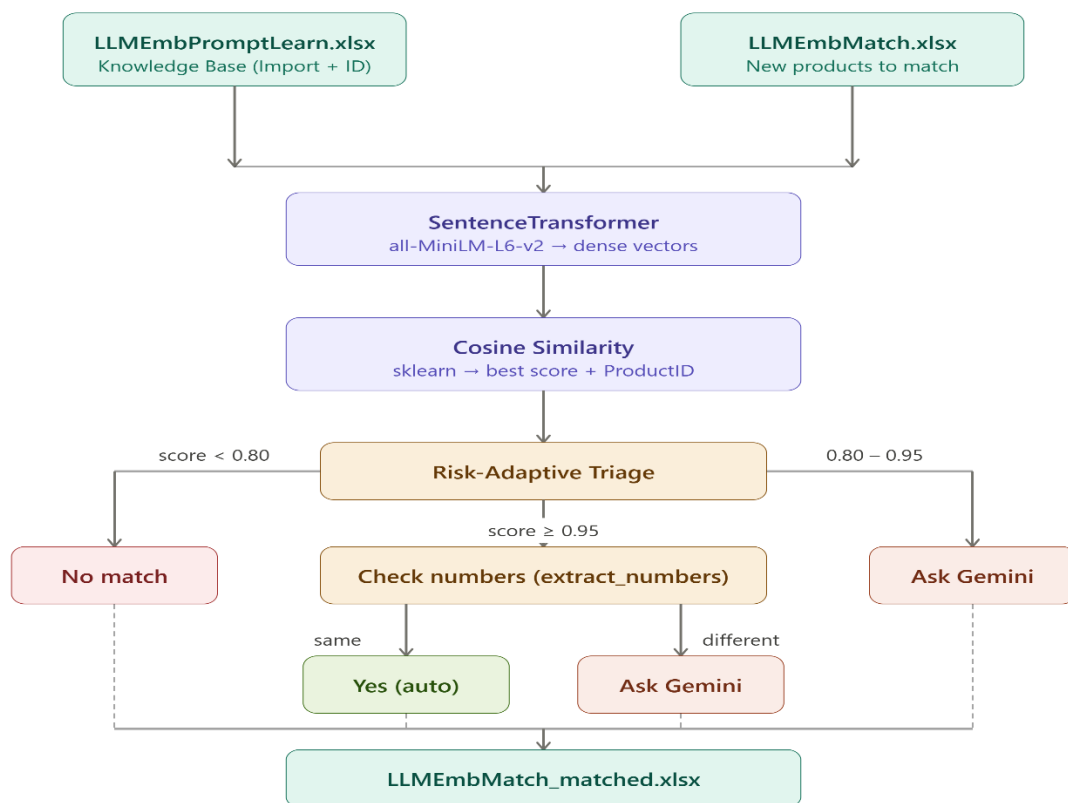
4.

Import	Matched ProductID	Score	Gemini	Final
15.6" FHD 144Hz/16GB ram/256GB SSD/Win10 Home/RTX3050/ MSI Pavilion 15, Ryzen 7 6800H	1	0,950500011	Yes	Yes
Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 64GB ram - 512GB SSD - NVIDIA RTX3060 - Win11 Home	25	0,986800015	No	No
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	0,993099988	Yes	Yes
galaxy samsung S22 1028GB 6.1" OLED Tensor G2 Android	28	0,987500012	No	No
galaxy samsung S22 512GB 6.1" OLED Tensor G2 // Android	27	0,991699994	Yes	Yes

Διορθώθηκε και η 2^η γραμμή. Πράγματι δεν υπάρχει το αντίστοιχο προϊόν είναι 32GB ram όχι 64 ορθά εδώ δεν αντιστοιχήθηκε.

Συνοψίζοντας η **τελική απόφαση (Final)** του αρχείου LLMEmbMatch_matched.xlsx για το αν κάνει matched με το προτεινόμενο Product ID το οποίο έχει προκύψει από την συνάρτηση def find_best_match η οποία χρησιμοποιεί cosine_similarity με το μεγαλύτερο Score (best_score), στηρίζεται στο αν το:

1. Score < 0,80 => No αυτομάτως.
2. Score 0,80 – 0,95 => Απόφαση από Gemini model: gemini-2.5-flash.
3. Score ≥ 0,95:
 - Ίδιοι αριθμοί στην περιγραφή => Yes αυτομάτως.
 - Διαφορετικοί αριθμοί στην περιγραφή => Απόφαση από LLM gemini-2.5-flash.



Επισήμανση: Το συγκεκριμένο script δεν εκπαιδεύει μοντέλο ML χρησιμοποιεί τα προϊόντα του LLMEmbPromptLearn.xlsx ως σύνολο αναφοράς (reference set) και εκτελεί semantic matching με embeddings και επαλήθευση μέσω Gemini.

Στο επόμενο script-στάδιο θα δούμε εφαρμογή με σύνδεση της βάσης δεδομένων.

```
# Train_productid.py: not rag only learn from LLMEmbPromptLearn.xlsx.
import re # because cosine_similarity not understands nums
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn")
import os
import pandas as pd
import joblib
import numpy as np
from tqdm import tqdm
from sklearn.metrics.pairwise import cosine_similarity
import time
from dotenv import load_dotenv
from sentence_transformers import SentenceTransformer
from google import genai

# ===== ENV =====
load_dotenv("gemini.env")
GEMINI_API_KEY = os.getenv("GEMINI_API_KEY")
if not GEMINI_API_KEY:
    raise EnvironmentError("✗ GEMINI_API_KEY not found")
client = genai.Client(api_key=GEMINI_API_KEY)

# ===== CONFIG =====
BASE_PATH = r"C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\LLMEmbPromptLearn.xlsx"
PLUS_PATH = r"C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\LLMEmbMatch.xlsx"
BASE_SHEET = "a16PT"
PLUS_SHEET = "a1"
EMB_CACHE_PATH = "embedding_cache_local.pkl"
GEMINI_CACHE_PATH = "gemini_cache.pkl"
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
# ===== new add after cosine_similarity false match =====

def extract_numbers(text):
    return set(re.findall(r'\d+', text))

# ===== CACHE =====
def load_cache(path):
    return joblib.load(path) if os.path.exists(path) else {}
def save_cache(cache, path):
    joblib.dump(cache, path)

# ===== EMBEDDINGS =====
def get_embeddings_batch(texts, cache):
    embeddings = []
    for text in texts:
        text = str(text)
        if text in cache:
            embeddings.append(cache[text])
        else:
            emb = embedding_model.encode(text)
            cache[text] = emb
            embeddings.append(emb)
    return embeddings

# ===== GEMINI =====
def ask_gemini_yes_no(import_text, matched_text, cache):
    key = f"{import_text}|||{matched_text}"
    if key in cache:
        return cache[key]
    prompt = f"""
You are a strict product matcher.
Do these refer to the EXACT same product?
Rules:
- Same brand + model → possible YES
- Any spec difference → NO
- Different category → NO
Answer ONLY: Yes or No
Import:
{import_text}
Matched:
{matched_text}
"""
    for _ in range(3):
        try:
            response = client.models.generate_content(
                model="gemini-2.5-flash",
                contents=prompt
            )
            reply = response.text.strip().lower()
            result = "Yes" if "yes" in reply else "No"
            cache[key] = result
            return result
        except Exception as e:
            if "429" in str(e):
                print("✗ Rate limit hit, waiting 45s...")
                time.sleep(45)
            else:
                print(f"Gemini error: {e}")
                return "Error"
    return "Error"

# ===== MATCHING =====
def find_best_match(import_embedding, product_db):
    if not product_db:
        return None, 0
    db_embeddings = [p["embedding"] for p in product_db]
    sims = cosine_similarity([import_embedding], db_embeddings)[0]
    best_idx = np.argmax(sims)
    best_score = sims[best_idx]
    best_pid = product_db[best_idx]["ProductID"]
    return best_pid, best_score
```

```
# ===== MAIN =====
if __name__ == "__main__":
    emb_cache = load_cache(EMB_CACHE_PATH)
    gemini_cache = load_cache(GEMINI_CACHE_PATH)
    # ===== BASE DATA =====
    df_base = pd.read_excel(BASE_PATH, sheet_name=BASE_SHEET)
    df_base.columns = df_base.columns.str.strip()
    if "Import" not in df_base.columns or "Correct ID" not in df_base.columns:
        print("✗ BASE columns missing")
        exit()

    print(f"BASE rows: {len(df_base)}")
    base_texts = df_base["Import"].fillna("").astype(str).tolist()
    base_embeddings = get_embeddings_batch(base_texts, emb_cache)
    product_db = []
    for desc, pid, emb in zip(df_base["Import"], df_base["Correct ID"], base_embeddings):
        if pd.isna(pid) or str(pid).strip() == "":
            continue
        product_db.append({
            "description": str(desc),
            "ProductID": str(pid).strip(),
            "embedding": emb
        })
    print(f"Product DB loaded: {len(product_db)}")
    if len(product_db) == 0:
        print("✗ No products in DB")
        exit()

    # ===== NEW DATA =====
    df_plus = pd.read_excel(PLUS_PATH, sheet_name=PLUS_SHEET, engine='openpyxl')
    df_plus.columns = df_plus.columns.str.strip()
    if "Import" not in df_plus.columns:
        print("✗ Import column missing in PLUS file")
        exit()

    plus_texts = df_plus["Import"].fillna("").astype(str).tolist()
    print(f"PLUS rows: {len(plus_texts)}")
    if len(plus_texts) == 0:
        print("⚠ No data to process")
        exit()

    plus_embeddings = get_embeddings_batch(plus_texts, emb_cache)
    save_cache(emb_cache, EMB_CACHE_PATH)
    # ===== MATCHING =====
    results = []
    for i, desc in enumerate(tqdm(plus_texts, desc="🔍 Matching")):
        emb = plus_embeddings[i]
        pid, score = find_best_match(emb, product_db)
        if pid is None:
            results.append((desc, None, 0, "No", "No"))
            continue
        matched_text = next(p["description"] for p in product_db if p["ProductID"] == pid)
        if score < 0.80:
            verdict = "No"
            final = "No"
        elif score ≥ 0.95:
            nums_import = extract_numbers(desc)
            nums_matched = extract_numbers(matched_text)
            if nums_import != nums_matched:
                verdict = ask_gemini_yes_no(desc, matched_text, gemini_cache)
                final = "Yes" if verdict == "Yes" else "No"
            else:
                verdict = "Yes"
                final = "Yes"
        else:
            verdict = ask_gemini_yes_no(desc, matched_text, gemini_cache)
            final = "Yes" if verdict == "Yes" else "No"
        results.append((desc, pid, round(score, 4), verdict, final))
    save_cache(gemini_cache, GEMINI_CACHE_PATH)
    df_results = pd.DataFrame(results, columns=[
        "Import",
        "Matched ProductID",
        "Score",
        "Gemini",
        "Final"
    ])
    output_path = PLUS_PATH.replace(".xlsx", "_matched.xlsx")
    with pd.ExcelWriter(output_path) as writer:
        df_plus.to_excel(writer, sheet_name="NewData", index=False)
        df_results.to_excel(writer, sheet_name="MatchingResults", index=False)
    print(f"✅ Saved: {output_path}")
```

3.2.3 Εφαρμογή αντιστοίχισης προϊόντων συνδέοντας τη ΒΔ.

Στάδιο 3 – Παραγωγή αποτελεσμάτων μέσω RAG:

Το τρίτο στάδιο συνδέεται με τη Βάση δεδομένων και αποτελεί βασικό πυρήνα λήψης αποφάσεων.

Συνδυάζει:

- Semantic Matching.
- Embedding Cache.
- Gemini Verification.
- Decision - Business Rules (Thresholds and numeric validation).

Η λειτουργία του υλοποιεί μία διαδικασία δύο επιπέδων.

Στο πρώτο επίπεδο πραγματοποιείται μαθηματική αξιολόγηση μέσω embeddings.

Στο δεύτερο επίπεδο πραγματοποιείται λογική επαλήθευση μέσω LLM.

Η προσέγγιση αυτή μειώνει σημαντικά τα ψευδώς θετικά αποτελέσματα.

Matching_EMBSEM.py αποτελεί τον κεντρικό πυρήνα (core engine) του συστήματος αντιστοίχισης προϊόντων Product Matching. Η αρχιτεκτονική του υλοποιεί ένα πλήρες Data Pipeline, το οποίο αντλεί δεδομένα από εταιρικές βάσεις, εκτελεί διανυσματικές συγκρίσεις, χρησιμοποιεί LLM όπου απαιτείται και επιστρέφει δομημένα αποτελέσματα.

1. Διασύνδεση Δεδομένων (Data Ingestion & SQL Integration)

Σε αντίθεση με τα πειραματικά σενάρια που βασίζονται αποκλειστικά σε στατικά αρχεία, ο πυρήνας διαθέτει εγγενή διασύνδεση με τον **Microsoft SQL Server** μέσω του πρωτοκόλλου **ODBC** (βιβλιοθήκη pyodbc).

- **Dynamic Query Execution:** Το σύστημα δεν ενσωματώνει (hardcode) τα SQL queries στον κώδικα της Python, αλλά τα διαβάζει δυναμικά από εξωτερικό αρχείο (AIprojectDescr.sql). Αυτό προάγει τον διαχωρισμό των αρμοδιοτήτων (Separation of Concerns).
- **Robust Cursor Handling:** Κατά την εκτέλεση του query, εφαρμόζεται έλεγχος (cursor.nextset()) για τη διασφάλιση της ορθής λήψης του Recordset, αποτρέποντας σφάλματα σε περιπτώσεις όπου η SQL επιστρέφει ενημερωτικά μηνύματα (π.χ. SET NOCOUNT ON) πριν τα πραγματικά δεδομένα.

2. Σημασιολογική Αναπαράσταση (Semantic Embeddings & Caching)

Για τη μετατροπή των περιγραφών των προϊόντων σε μαθηματικά διανύσματα, χρησιμοποιείται το τοπικά εκτελούμενο μοντέλο **all-MiniLM-L6-v2** (μέσω της sentence-transformers).

- **Stateful Caching:** Εφαρμόζεται μηχανισμός μνήμης μέσω της βιβλιοθήκης joblib (embedding_cache_semantic.pkl). Το σύστημα υπολογίζει το διάνυσμα μιας περιγραφής μόνο την πρώτη φορά. Σε επόμενες συναντήσεις του ίδιου λεκτικού (π.χ. κοινά προϊόντα), ανασύρει το διάνυσμα από τη μνήμη, επιτυγχάνοντας $O(1)$ πολυπλοκότητα χρόνου αναζήτησης δηλ. θεωρητικά το $\min T$ και εξοικονομώντας υπολογιστικούς πόρους αλλά και χρηματικό κόστος καθώς χρειάζονται λιγότερα tokens σε κλήσεις API του LLM.

3. Ο Μηχανισμός Δρομολόγησης (Risk-Adaptive Triage & Hybrid Matching)

Ο αλγόριθμος ταύτισης (process_supplier_matching) υλοποιεί την υβριδική λογική ελέγχου, συνδυάζοντας τη Μαθηματική Στατιστική (embeddings και cosine similarity) με την Τεχνητή Νοημοσύνη LLM (Gemini 2.5 Flash).

Αρχικά, υπολογίζεται η **Ομοιότητα Συνημιτόνου (Cosine Similarity)** μεταξύ του εισερχόμενου προϊόντος (από το Excel) και του συνόλου της βάσης δεδομένων. Το μέγιστο σκορ (best_score) καθορίζει τη ροή εκτέλεσης βάσει κατωφλίων (thresholds):

1. **Ασφαλής Απόρριψη (Score < 80%):** Η σημασιολογική απόσταση είναι μεγάλη. Η ταύτιση απορρίπτεται άμεσα ντετερμινιστικά, χωρίς να καταναλωθούν πόροι API.
2. **Ασφαλής Ταύτιση (Score $\geq$ 95%):** Όταν η τιμή της cosine similarity είναι μεγαλύτερη ίση του 95%, το σύστημα θεωρεί ότι υπάρχει πολύ υψηλή σημασιολογική ομοιότητα μεταξύ των δύο περιγραφών. Ωστόσο, πριν από την αυτόματη αποδοχή του αποτελέσματος εφαρμόζεται πρόσθετος έλεγχος αριθμητικών χαρακτηριστικών μέσω του μηχανισμού extract numbers(). Αν οι αριθμητικές τιμές των δύο περιγραφών ταυτίζονται, η αντιστοίχιση εγκρίνεται άμεσα. Σε αντίθετη περίπτωση, η απόφαση μεταφέρεται στο μοντέλο Gemini για τελική σημασιολογική επαλήθευση. Η προσέγγιση αυτή μειώνει σημαντικά τα ψευδώς θετικά αποτελέσματα που προκύπτουν από προϊόντα με παρόμοια ονομασία αλλά διαφορετικά τεχνικά χαρακτηριστικά.
3. **Validation Agent:** Σε συνέχεια του 2 είναι σημαντικό ότι το σύστημα ενσωματώνει και μηχανισμό επικύρωσης αριθμητικών χαρακτηριστικών σαν Validation Agent, ο οποίος εξάγει όλα τα αριθμητικά δεδομένα των δύο περιγραφών μέσω της συνάρτησης extract_numbers(). Έτσι αντιμετωπίζεται η αδυναμία των embeddings, σύμφωνα με την οποία δύο περιγραφές με πολύ υψηλή σημασιολογική ομοιότητα ενδέχεται να αναφέρονται σε διαφορετικά προϊόντα λόγω διαφορών σε αριθμητικά χαρακτηριστικά, όπως χωρητικότητα αποθήκευσης (128GB έναντι 256GB), μέγεθος οθόνης, κοκ. Εφόσον εντοπιστούν αποκλίσεις, η τελική απόφαση μεταφέρεται στο μοντέλο Gemini για σημασιολογική επαλήθευση.
4. **Νευρο-Συμβολική Επαλήθευση (80% $\leq$ Score $\leq$ 95%):** Η ζώνη αβεβαιότητας. Το σύστημα επικαλείται τη συνάρτηση ask_gemini_yes_no. Εφαρμόζεται τεχνική Zero-Shot Strict Prompting, όπου παρέχονται στο LLM οι δύο περιγραφές και μια λίστα αυστηρών εξαιρέσεων (π.χ. διαφορές σε CPU, RAM, Storage). Το μοντέλο εξαναγκάζεται να απαντήσει αποκλειστικά με "Yes" ή "No", λειτουργώντας ως μηχανισμός σημασιολογικής επαλήθευσης βασισμένος στη λογική κατανόηση.

4. Εξαγωγή Δεδομένων (Data Serialization)

Η δομή λήξης του προγράμματος συγκεντρώνει τα αποτελέσματα (Αρχικό Όνομα, Προτεινόμενο ID Βάσης, ML Score, Απόφαση Gemini, Τελική Έγκριση) σε ένα **Pandas DataFrame**. Το τελικό αποτέλεσμα εγγράφεται απευθείας στο αρχικό Excel αρχείο παραγωγής (LLMEmbMatch.xlsx), παρέχοντας στον τελικό χρήστη (ή στο σύστημα ERP) ένα καθαρό, δομημένο και ελέγξιμο output.

```

Loading weights: 100% [██████████] | 103/103 [00:00<00:00, 2454.83it/s]
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key | Status | |
-----+-----+-----+
embeddings.position_ids | UNEXPECTED | |

Notes:
- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.
Executing SQL...
Rows fetched: 20
Loaded products: 20
Matching: 100% [██████████] | 10/10 [00:05<00:00, 1.81it/s]
Done

Process finished with exit code 0

```

Πρακτικά παραδείγματα:

1.

Import	Matching ID	Matching Name	Embedding Score	ini Verifica	inal Match
Google Galaxy S2289, 5	11	Google Galaxy S22, 512GB	97,01000214	Yes	Yes
Xiaomi Galaxy S23 Ultra	12	Xiaomi Galaxy S23 Ultra, 2	99,33999634	Yes	Yes
Dell XPS 13, i7-12700H/	13	Dell XPS 13, i7-12700H/15	98,12999725	Yes	Yes
Xiaomi iPhone 14 Pro,	14	Android Xiaomi iPhone 14	97,88999939	Yes	Yes
Apple iPhone 13, 128G	15	Apple iPhone 13, 128GB, 6	100	Yes	Yes
Acer Pavilion 15, i5-113	16	Acer Pavilion 15, i5-1135G	100	Yes	Yes
Xiaomi iPhone 13, 128G	17	Xiaomi iPhone 13, 128GB,	94,70999908	No	No
Samsung iPhone 14 Pro	18	O 512GB, 6.7" AMOLED 12	95,41000366	Yes	Yes
MSI ROG Strix G15, Ryz	19	MSI ROG Strix G15, Ryzen	100	Yes	Yes
MSI Aspire 5, i7-1165G7	20	MSI Aspire 5, /16GB/512G	96,26000214	Yes	Yes

2.

Import	Matching ID	Matching Name	Embedding Score	Gemini Verification	Final Match
MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB ram/256GB SSD/RTX3050/Win10 Home	1	MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB/256GB SSD/RTX3050/Win10 Home	98,63999939	Yes	Yes
Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB ram - 512GB SSD - NVIDIA RTX3060 - Win11 Home	25	Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB - 512GB SSD - NVIDIA RTX3060 - Win11 Home	98,68000031	Yes	Yes
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	99,30999756	Yes	Yes
galaxy samsung S22 128GB 6.1" OLED Tensor G2 Android	28	Samsung Galaxy S22 128GB 6.1" OLED Tensor G2 Android	99,87999725	Yes	Yes
galaxy samsung S22 512GB 6.1" OLED Tensor G2 Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	99,87000275	Yes	Yes

Τα πειραματικά αποτελέσματα έδειξαν ότι το προτεινόμενο υβριδικό σύστημα αντιστοίχισης προϊόντων κατάφερε να αναγνωρίσει με επιτυχία προϊόντα τα οποία παρουσίαζαν σημαντικές διαφοροποιήσεις στη μορφοποίηση των περιγραφών τους. Σε όλες τις δοκιμαστικές περιπτώσεις, το σύστημα εντόπισε το σωστό ProductID και επέστρεψε πολύ υψηλές τιμές Embedding Score (98,64%–99,88%), γεγονός που υποδηλώνει ισχυρή σημασιολογική ομοιότητα μεταξύ των περιγραφών. Ειδικότερα, αναγνωρίστηκαν ως ταυτόσημα προϊόντα περιγραφές που περιείχαν διαφορετική σειρά λέξεων, επιπλέον όρους (π.χ. «ram»), εναλλαγές κεφαλαίων και πεζών χαρακτήρων, καθώς και διαφοροποιήσεις στην ονομασία του κατασκευαστή, όπως στην περίπτωση «katana dell GF66» και «Dell Katana GF66». Παρόμοια, οι περιγραφές των κινητών τηλεφώνων Samsung Galaxy S22 ταυτοποιήθηκαν επιτυχώς παρά τις διαφοροποιήσεις στη σύνταξη των λέξεων («galaxy samsung S22» έναντι «Samsung Galaxy S22»). Η επιβεβαίωση από το Gemini ήταν θετική σε όλες τις περιπτώσεις, επιβεβαιώνοντας ότι οι υψηλές τιμές cosine similarity αντιστοιχούσαν πράγματι στο ίδιο προϊόν. Τα αποτελέσματα αυτά καταδεικνύουν ότι ο συνδυασμός embeddings, cosine similarity και επαλήθευσης μέσω LLM μπορεί να αντιμετωπίσει όπως φαίνεται εδώ σχετικά αποτελεσματικά προβλήματα ετερογένειας δεδομένων και διαφοροποιήσεων στις περιγραφές προϊόντων που συναντώνται συχνά σε περιβάλλοντα ηλεκτρονικού εμπορίου και ενοποίησης καταλόγων προϊόντων.

3.

```
C:\Users\lgf\AppData\Local\Programs\Python\Python313\python.exe C:\Users\lgf\Desktop\lg\Edu\Thesis\Gcode\Matching_ERBSEB.py
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
Loading weights: 100% [████████████████████] 103/103 [00:00<00:00, 7527.40it/s]
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key | Status |
-----|-----|
embeddings.position_ids | UNEXPECTED |

Notes:
- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.
Executing SQL...
Rows fetched: 20
Loaded products: 20
Matching: 100% [████████████████████] 10/10 [00:00<00:00, 2665.59it/s]
Done
Process finished with exit code 0
```

Import	Matching ID	Matching Name	Embedding Score	Gemini Verification	Final Match
Lenovo Aspire 5, Ryzen 7 6800H/13.3" FHD/8GB/512GB SSD/AMD Radeon Graphics/Win11 Pro	21	Lenovo Aspire 5 - Ryzen 7 6800H - 13.3" FHD - 8GB - 512GB SSD - AMD Radeon Graphics - Win11 Pro	97,06999969	Yes	Yes
Xiaomi Pixel 7, 128GB, 6.8" AMOLED, A16 Bionic, Android	22	Xiaomi Pixel 7 128GB 6.8" AMOLED A16 Bionic Android	96,88999939	Yes	Yes
Dell XPS 13, Ryzen 7 5800H/15.6" FHD 144Hz/16GB/512GB SSD/Intel Iris Xe/Win11 Pro	23	Dell XPS 13 - Ryzen 7 5800H - 15.6" FHD 144Hz - 16GB - 512GB SSD - Intel Iris Xe - Win11 Pro	98,01999664	Yes	Yes
Apple iPhone 13, 128GB, 6.7" AMOLED 120Hz, A15 Bionic, Android	24	Apple iPhone 13 128GB 6.7" AMOLED 120Hz A15 Bionic Android	96,63999939	Yes	Yes
Asus Aspire 5, Ryzen 7 5800H/15.6" FHD 144Hz/32GB/512GB SSD/NVIDIA RTX3060/Win11 Home	25	Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB - 512GB SSD - NVIDIA RTX3060 - Win11 Home	97,30999756	Yes	Yes
Google Galaxy S23 Ultra, 256GB, 6.1" OLED, A16 Bionic, Android	26	Google Galaxy S23 Ultra 256GB 6.1" OLED A16 Bionic Android	96,95999908	Yes	Yes
Samsung Galaxy S22, 512GB, 6.1" OLED, Tensor G2, Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	96,52999878	Yes	Yes
Samsung Galaxy S22, 128GB, 6.1" OLED, Tensor G2, Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	96,38999939	Yes	Yes
Lenovo XPS 13, i7-12700H/15.6" FHD 144Hz/16GB/1TB SSD/NVIDIA RTX3060/Win10 Home	29	Lenovo XPS 13 - i7-12700H - 15.6" FHD 144Hz - 16GB - 1TB SSD - NVIDIA RTX3060 - Win10 Home	96,54000092	Yes	Yes
Dell Aspire 5, i7-1165G7/15.6" FHD 144Hz/32GB/256GB SSD/NVIDIA RTX3050/Win11 Pro	30	Dell Aspire 5 - i7-1165G7 - 15.6" FHD 144Hz - 32GB - 256GB SSD - NVIDIA RTX3050 - Win11 Pro	97,61000061	Yes	Yes

4.

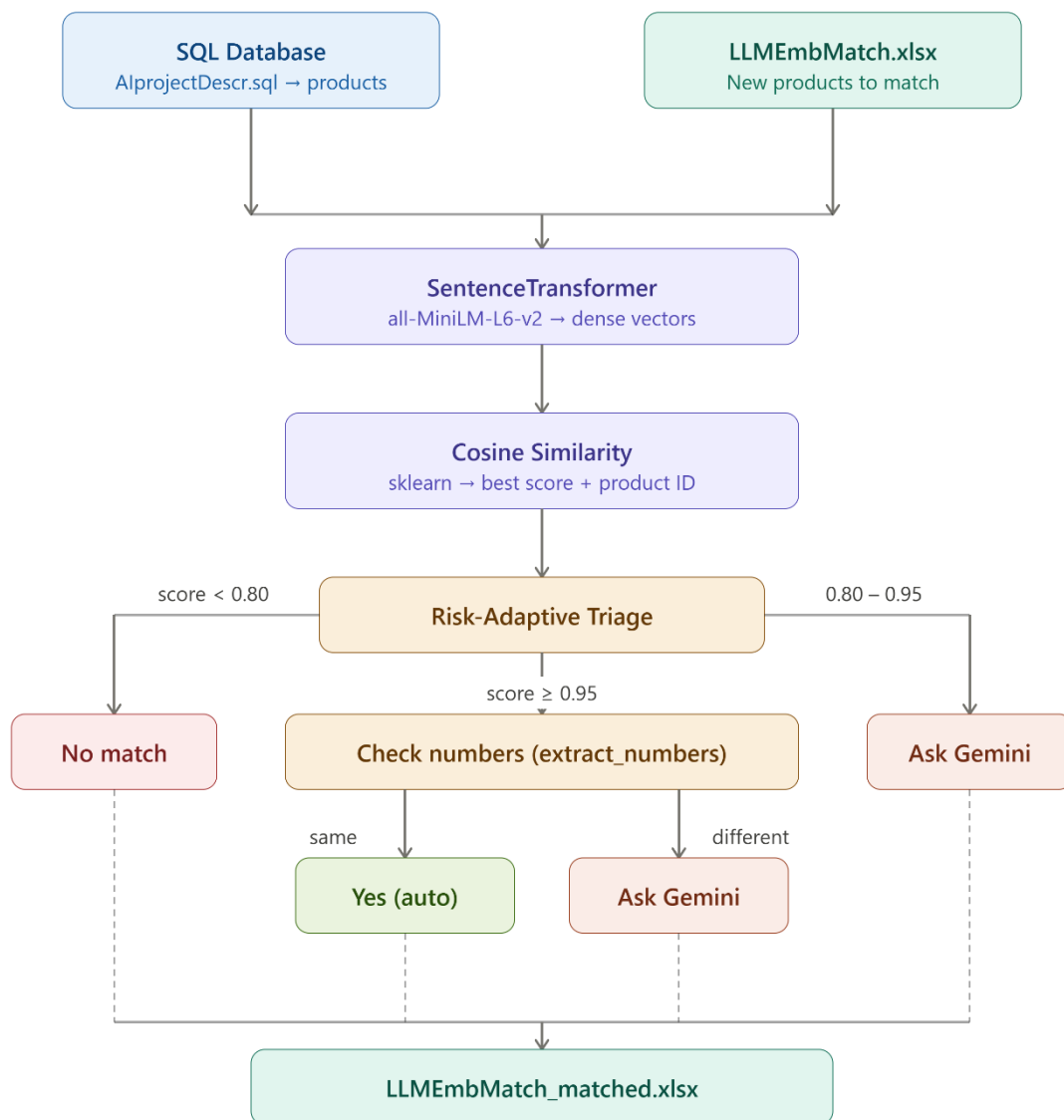
Import	Matching ID	Matching Name	Embedding Score	Gemini Verification	Final Match
15.6" FHD 144Hz/16GB ram/256GB SSD/Win10 Home/RTX3050/ MSI Pavilion 15, Ryzen 7 6800H	1	MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB/256GB SSD/RTX3050/Win10 Home	95,05000305	Yes	Yes
Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 64GB ram - 512GB SSD - NVIDIA RTX3060 - Win11 Home	25	Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB - 512GB SSD - NVIDIA RTX3060 - Win11 Home	98,68000031	No	No
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	99,30999756	Yes	Yes
galaxy samsung S22 1028GB 6.1" OLED Tensor G2 Android	28	Samsud Galaxy S22 128GB 6.1" OLED Tensor G2 Android	98,75	No	No
galaxy samsung S22 512GB 6.1" OLED Tensor G2 // Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	99,16999817	Yes	Yes
galaxy samsung S12 522GB 6.1" OLED Tensor G2 // Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	92,55000305	No	No

Στο παραπάνω παράδειγμα έγιναν δοκιμές σε πραγματικές αλλά και τροποποιημένες περιγραφές προϊόντων, με σκοπό να διερευνηθεί το τελικό αποτέλεσμα (Final Match) σε διαφορετικά επίπεδα σημασιολογικής ομοιότητας. Για κάθε περιγραφή εισόδου (import) το σύστημα υπολογίζει αρχικά το ποσοστό ομοιότητας μέσω των embeddings και της Cosine Similarity και στη συνέχεια, όπου απαιτείται, ενεργοποιεί το Μεγάλο Γλωσσικό Μοντέλο Gemini 2.5 Flash για λογική επαλήθευση της αντιστοίχισης. Στην πρώτη περίπτωση, η περιγραφή του προϊόντος MSI Pavilion 15 Ryzen 7 6800H παρουσίασε ποσοστό ομοιότητας 95,05% και το Gemini επιβεβαίωσε ότι πρόκειται για το ίδιο προϊόν, οδηγώντας

σε επιτυχημένη αντιστοίχιση (Final Match = Yes), παρά τις διαφορές στη σειρά εμφάνισης των χαρακτηριστικών. Αντίθετα, στο δεύτερο παράδειγμα, δύο περιγραφές του ίδιου μοντέλου Asus Aspire 5 εμφάνισαν ιδιαίτερα υψηλή ομοιότητα (98,68%), όμως διέφεραν στη μνήμη RAM (64GB έναντι 32GB). Παρότι η Cosine Similarity θεώρησε τις δύο περιγραφές σχεδόν ταυτόσημες, το Gemini αναγνώρισε ότι πρόκειται για διαφορετικές διαμορφώσεις του προϊόντος και απέρριψε σωστά την αντιστοίχιση (Final Match = No). Αντίστοιχα, στην περίπτωση του Dell Katana GF66, όπου οι διαφορές περιορίζονταν αποκλειστικά στη διατύπωση της περιγραφής, το σύστημα υπολόγισε ομοιότητα 99,31%, το Gemini επιβεβαίωσε τη σημασιολογική ισοδυναμία και πραγματοποιήθηκε επιτυχής αντιστοίχιση. Ιδιαίτερο ενδιαφέρον παρουσιάζουν τα παραδείγματα των συσκευών Samsung Galaxy S22. Στην πρώτη περίπτωση, η περιγραφή περιείχε λανθασμένη χωρητικότητα αποθήκευσης (1028GB αντί 128GB). Παρότι η Cosine Similarity υπολόγισε ποσοστό ομοιότητας 98,75%, ο μηχανισμός ελέγχου αριθμητικών χαρακτηριστικών (extract_numbers) εντόπισε την απόκλιση και ενεργοποίησε το Gemini, το οποίο απέρριψε την αντιστοίχιση. Αντίθετα, όταν οι δύο περιγραφές αφορούσαν πραγματικά το ίδιο προϊόν των 512GB και διέφεραν μόνο στη μορφοποίηση και στη χρήση ειδικών χαρακτήρων, το σύστημα αναγνώρισε σωστά την αντιστοίχιση και την αποδέχθηκε. Τέλος, σε περίπτωση όπου η περιγραφή περιείχε πολλαπλά σφάλματα, όπως ελλιπή ονομασία μοντέλου και λανθασμένη χωρητικότητα (522GB αντί 512GB), το ποσοστό ομοιότητας παρέμεινε σχετικά υψηλό (92,55%), όμως η τελική απόφαση ήταν ορθή απόρριψη, καθώς το Gemini αναγνώρισε ότι οι δύο περιγραφές δεν αντιστοιχούν στο ίδιο προϊόν. Τα αποτελέσματα αυτά καταδεικνύουν ότι η αποκλειστική χρήση της Cosine Similarity δεν επαρκεί για αξιόπιστη αντιστοίχιση προϊόντων σε πραγματικά εμπορικά δεδομένα, καθώς προϊόντα με διαφορετικά τεχνικά χαρακτηριστικά μπορούν να εμφανίζουν πολύ υψηλή σημασιολογική ομοιότητα. Για τον λόγο αυτό, το προτεινόμενο υβριδικό σύστημα συνδυάζει τρία διαδοχικά επίπεδα ελέγχου: τη σημασιολογική αναζήτηση μέσω embeddings, τον έλεγχο κρίσιμων αριθμητικών χαρακτηριστικών μέσω της συνάρτησης extract_numbers() και τη λογική επαλήθευση μέσω του μοντέλου Gemini. Η πολυεπίπεδη αυτή διαδικασία μειώνει σημαντικά τα ψευδώς θετικά αποτελέσματα (False Positives), αυξάνει την αξιοπιστία της τελικής απόφασης και αποδεικνύει στην πράξη ότι ο συνδυασμός κλασικών τεχνικών Μηχανικής Μάθησης με Μεγάλα Γλωσσικά Μοντέλα υπερτερεί έναντι της αποκλειστικής χρήσης οποιασδήποτε από τις δύο προσεγγίσεις. Σημαντικό είναι επίσης και το οριζόμενο prompt συνάρτηση aks_gemini_yes_no ώστε να αποφευχθούν hallucinations. Το LLM

αξιοποιείται μέσω ενός αυστηρά σχεδιασμένου prompt (strict prompting), στο οποίο ορίζονται ρητά οι κανόνες ταύτισης των προϊόντων. Το Gemini λαμβάνει τις δύο περιγραφές και επιστρέφει αποκλειστικά μία δυαδική απάντηση ("Yes" ή "No"), λειτουργώντας ως μηχανισμός σημασιολογικής επαλήθευσης και μειώνοντας τα ψευδώς θετικά αποτελέσματα που δεν μπορούν να εντοπιστούν μόνο μέσω της cosine similarity.

Flowchart του εν λόγω script:



```
# Matching_EMBSEM.py: RAG cosine sim.
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn")
import os
import re # added after cosine_similarity false match on numbers
import pyodbc
import pandas as pd
import numpy as np
import joblib
from tqdm import tqdm
from sklearn.metrics.pairwise import cosine_similarity
from dotenv import load_dotenv
from sentence_transformers import SentenceTransformer
from google import genai

# ===== ENV =====
load_dotenv("gemini.env")
if not os.getenv("GEMINI_API_KEY"):
    raise EnvironmentError("✗ GEMINI_API_KEY not found")
client = genai.Client(api_key=os.getenv("GEMINI_API_KEY"))
# ===== EMBEDDING =====
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
CACHE_PATH = "embedding_cache_semantic.pkl"
# ===== SQL =====
conn_str = (
    f"DRIVER={{SQL Server}};"
    f"SERVER={os.getenv('DB_SERVER')};"
    f"DATABASE={os.getenv('DB_DATABASE')};"
    "Trusted_Connection=yes;"
)

def get_db_connection():
    return pyodbc.connect(conn_str)

def fetch_product_data(): #Data Retrieval Agent
    sql_path = r"C:\Users\ig17\Desktop\ig\Edo\Thesis\0code\AIprojectDescr.sql"
    if not os.path.exists(sql_path):
        raise FileNotFoundError(f"✗ SQL file not found: {sql_path}")
    with open(sql_path, "r", encoding="utf-8") as sql_file:
        sql_query = sql_file.read()
    conn = get_db_connection()
    cursor = conn.cursor()
    print("Executing SQL...")
    cursor.execute(sql_query)
    while cursor.description is None:
        if not cursor.nextset():
            raise ValueError("✗ SQL query did not return SELECT results")
    columns = [col[0] for col in cursor.description]
    rows = cursor.fetchall()
    print(f"Rows fetched: {len(rows)}")
    return [dict(zip(columns, row)) for row in rows]

# ===== CACHE =====
def load_embedding_cache(): # Cache Agent not calculated again embeddings.
    return joblib.load(CACHE_PATH) if os.path.exists(CACHE_PATH) else {}
def save_embedding_cache(cache):
    joblib.dump(cache, CACHE_PATH)

# ===== NUMBER EXTRACTOR =====
# added because cosine_similarity cannot distinguish numeric differences
def extract_numbers(text): # Validation Agent
    return set(re.findall(r'\d+', text))

# ===== EMBEDDINGS =====
def get_embeddings_batch(texts, cache): # Embedding Agent from texts to vectors
    embeddings = []
    for text in texts:
        text = str(text)
        if text in cache:
            embeddings.append(cache[text])
        else:
            emb = embedding_model.encode(text)
            cache[text] = emb
            embeddings.append(emb)
    return embeddings

# ===== GEMINI =====
def ask_gemini_yes_no(import_name, matched_name): # LLM Reasoning Agent.
    prompt = f"""
    You are a strict product matching system.
    Determine if the following two product descriptions refer to the EXACT SAME product.
    Rules:
    - Match must be identical model (brand + model name)
    - Differences in storage (e.g., 128GB vs 256GB) → NO
    - Differences in screen size → NO
    - Different CPU/GPU → NO
    - Different product category (laptop vs smartphone) → NO
    - Minor wording differences are allowed
    Answer ONLY: Yes or No
    Import:
    {import_name}
    Matched:
    {matched_name}
    """
    try:
        response = client.models.generate_content(
            model="gemini-2.5-flash",
            contents=prompt
        )
        reply = response.text.strip().lower()
        return "Yes" if "yes" in reply else "No"
    except Exception as e:
        print(f"Gemini error: {e}")
        return "Error"

# ===== MATCHING =====
def find_best_match(import_embedding, product_dbs, threshold=0.75): # Similarity Agent: calculates cosine similarity
    db_embeddings = [p["embedding"] for p in product_dbs]
    sims = cosine_similarity([import_embedding], db_embeddings)[0]
    best_idx = np.argmax(sims)
    best_pid = product_dbs[best_idx]["id"]
    best_score = sims[best_idx]
    if best_score < threshold:
        return None, best_score
    return best_pid, best_score
```

```
# ===== MAIN =====
def process_supplier_matching():
    excel_path = r"C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\LLMEmbMatch.xlsx"
    sheet_name = "ai"
    df = pd.read_excel(excel_path, sheet_name=sheet_name, engine='openpyxl')
    if 'Import' not in df.columns:
        print("✗ Import column missing")
        return

    embedding_cache = load_embedding_cache()
    product_data = fetch_product_data()
    name_by_id = {p['id']: p['Name'] for p in product_data}
    product_texts = [p['Name'] for p in product_data]
    product_embeddings = get_embeddings_batch(product_texts, embedding_cache)
    product_db = [
        {"id": p['id'], "name": p['Name'], "embedding": emb}
        for p, emb in zip(product_data, product_embeddings)
    ]
    print(f"Loaded products: {len(product_db)}")
    import_texts = df['Import'].fillna("").astype(str).tolist()
    import_embeddings = get_embeddings_batch(import_texts, embedding_cache)
    save_embedding_cache(embedding_cache)
    results = []
    for i, import_text in enumerate(tqdm(import_texts, desc="Matching")):
        emb = import_embeddings[i]
        best_id, score = find_best_match(emb, product_db)
        if best_id is None:
            results.append((None, "No Match", round(score * 100, 2), "No", "No"))
            continue
        matched_name = name_by_id[best_id]
        if score < 0.80: # Decision Agent
            verdict = "No"
            final = "No"
        elif score ≥ 0.95:
            # extract_numbers check before auto-accept
            nums_import = extract_numbers(import_text)
            nums_matched = extract_numbers(matched_name)
            if nums_import ≠ nums_matched:
                verdict = ask_gemini_yes_no(import_text, matched_name)
                final = "Yes" if verdict == "Yes" else "No"
            else:
                verdict = "Yes"
                final = "Yes"
        else:
            verdict = ask_gemini_yes_no(import_text, matched_name)
            final = "Yes" if verdict == "Yes" else "No"
        results.append((
            best_id,
            matched_name,
            round(score * 100, 2),
            verdict,
            final
        ))
    result_df = pd.DataFrame(results, columns=[
        "Matching ID",
        "Matching Name",
        "Embedding Score",
        "Gemini Verification",
        "Final Match"
    ])
    result_df.insert(0, "Import", df['Import'])
    result_df.to_excel(excel_path, sheet_name=sheet_name, index=False) # Reporting Agent
    print("Done")

# ===== RUN =====
if __name__ == "__main__":
    process_supplier_matching() #Orchestrator
```

3.3 Υλοποίηση μέσω Πρακτόρων Agents.

Συμπερασματικά το σύστημα υλοποιεί μία πολυεπίπεδη υβριδική αρχιτεκτονική λήψης αποφάσεων, η οποία συνδυάζει σημασιολογική αναζήτηση μέσω embeddings, επιχειρησιακούς κανόνες επικύρωσης, μηχανισμούς προσωρινής αποθήκευσης και επαλήθευση μέσω Μεγάλου Γλωσσικού Μοντέλου LLM. Η αρχιτεκτονική μπορεί να αποδοθεί και ως προσέγγιση Agent-Based Processing, όπου κάθε agent αναλαμβάνει μία οριοθετημένη ευθύνη.

Κάθε πράκτορας αναλαμβάνει μία συγκεκριμένη λειτουργία εντός της συνολικής διαδικασίας αντιστοίχισης προϊόντων. Ο διαχωρισμός αυτός επιτρέπει καλύτερη οργάνωση του κώδικα, αυξημένη επεκτασιμότητα, ευκολότερη συντήρηση και μεγαλύτερη αξιοπιστία στη λήψη αποφάσεων.

- **Data Retrieval Agent:**

Ο πράκτορας αυτός είναι υπεύθυνος για την ανάκτηση των δεδομένων από τις πηγές πληροφορίας. Συνδέεται με τον SQL Server μέσω της βιβλιοθήκης pyodbc, εκτελεί προκαθορισμένα SQL ερωτήματα και ανακτά τα προϊόντα της εσωτερικής βάσης δεδομένων. Παράλληλα διαβάζει τα δεδομένα εισαγωγής από αρχεία Excel που περιέχουν τις περιγραφές προϊόντων των προμηθευτών. Αποτελεί το σημείο εκκίνησης ολόκληρης της διαδικασίας.

Embedding Agent:

- Ο πράκτορας αυτός μετατρέπει τις περιγραφές προϊόντων σε διανυσματικές αναπαραστάσεις (embeddings) χρησιμοποιώντας το εκπαιδευμένο μοντέλο SentenceTransformer all-MiniLM-L6-v2. Μέσω της διαδικασίας αυτής, οι περιγραφές μετασχηματίζονται από φυσική γλώσσα σε μαθηματικά διανύσματα, τα οποία μπορούν να συγκριθούν αριθμητικά ως προς τη σημασιολογική τους ομοιότητα.

- **Cache Agent:**

Για τη βελτιστοποίηση της απόδοσης, το σύστημα ενσωματώνει μηχανισμό προσωρινής αποθήκευσης (caching). Ο Cache Agent αποθηκεύει τα embeddings που έχουν ήδη υπολογιστεί σε τοπικό αρχείο μέσω της βιβλιοθήκης joblib. Με τον τρόπο αυτό αποφεύγονται επαναλαμβανόμενοι υπολογισμοί, μειώνεται ο χρόνος εκτέλεσης και περιορίζεται το συνολικό υπολογιστικό κόστος.

- **Similarity Agent:**

Ο Similarity Agent είναι υπεύθυνος για τον υπολογισμό της σημασιολογικής ομοιότητας μεταξύ των προϊόντων του προμηθευτή και των προϊόντων της εσωτερικής βάσης. Χρησιμοποιεί το μέτρο Cosine Similarity για τη σύγκριση των embeddings και επιλέγει το προϊόν με τη μεγαλύτερη βαθμολογία ομοιότητας ως υποψήφιο αποτέλεσμα αντιστοίχισης.

- **Validation Agent:**

Ο πράκτορας επικύρωσης εφαρμόζει πρόσθετους επιχειρησιακούς κανόνες πριν από την τελική απόφαση. Στην παρούσα υλοποίηση πραγματοποιεί έλεγχο αριθμητικών χαρακτηριστικών που εντοπίζονται στις περιγραφές προϊόντων, όπως χωρητικότητα αποθήκευσης, μέγεθος οθόνης ή άλλα τεχνικά χαρακτηριστικά. Με τον τρόπο αυτό αποτρέπονται λανθασμένες αντιστοιχίσεις που ενδέχεται να παρουσιάζουν υψηλή σημασιολογική ομοιότητα αλλά να αφορούν διαφορετικά προϊόντα.

- **LLM Reasoning Agent:**

Ο πράκτορας αυτός αξιοποιεί το μεγάλο γλωσσικό μοντέλο Gemini για σημασιολογική επαλήθευση των αποτελεσμάτων. Σε περιπτώσεις όπου η βαθμολογία ομοιότητας βρίσκεται σε ενδιάμεσες ή αμφίβολες περιοχές, το μοντέλο καλείται να εξετάσει αν οι δύο περιγραφές αναφέρονται στο ίδιο ακριβώς προϊόν. Το αποτέλεσμα επιστρέφεται ως δυαδική απόφαση (Yes/No), λειτουργώντας ως μηχανισμός λογικής επιβεβαίωσης.

- **Decision Agent:**

Ο Decision Agent αποτελεί τον κεντρικό μηχανισμό λήψης αποφάσεων του συστήματος. Συνδυάζει τα αποτελέσματα του Similarity Agent, του Validation Agent και του LLM Reasoning Agent και εφαρμόζει προκαθορισμένα κατώφλια (thresholds) και επιχειρησιακούς κανόνες. Με βάση τα δεδομένα αυτά αποφασίζει αν μία αντιστοίχιση θα γίνει αποδεκτή ή θα απορριφθεί.

- **Reporting Agent:**

Ο τελευταίος πράκτορας είναι υπεύθυνος για την παραγωγή και αποθήκευση των τελικών αποτελεσμάτων. Καταγράφει το προτεινόμενο Product ID, την περιγραφή του προϊόντος, το ποσοστό ομοιότητας, την απόφαση του γλωσσικού μοντέλου και την τελική απόφαση αντιστοίχισης. Τα αποτελέσματα αποθηκεύονται σε αρχείο Excel και είναι διαθέσιμα για περαιτέρω επιχειρησιακή αξιοποίηση.

Συνολικά, οι παραπάνω πράκτορες συνεργάζονται υπό τον συντονισμό ενός κεντρικού Orchestrator, ο οποίος ελέγχει τη σειρά εκτέλεσης των επιμέρους σταδίων και διασφαλίζει την ομαλή ροή των δεδομένων από την ανάκτηση μέχρι την τελική επιχειρησιακή απόφαση. Η συνολική ροή μπορεί να αποδοθεί ως:

Orchestrator → Data Retrieval Agent → Embedding Agent → Cache Agent → Similarity Agent → Validation Agent → LLM Reasoning Agent → Decision Agent → Reporting Agent.

3.4 Εφαρμογή Αντιστοίχισης Προϊόντων με Machine Learning.

Εκτός της σημασιολογικής αντιστοίχισης προϊόντων μέσω embeddings και την επαλήθευση ή όχι αυτών μέσω Μεγάλων Γλωσσικών Μοντέλων (LLMs), στο πλαίσιο της παρούσας εργασίας διερευνήθηκε και η εφαρμογή εποπτευόμενης Μηχανικής Μάθησης (Supervised Machine Learning) για το ίδιο πρόβλημα. Στόχος ήταν να αξιολογηθεί κατά πόσο ένα κλασικό μοντέλο ταξινόμησης μπορεί να μάθει από ιστορικά δεδομένα αντιστοιχίσεων και να προβλέπει εάν δύο περιγραφές προϊόντων αναφέρονται στο ίδιο προϊόν.

Για τον σκοπό αυτό αναπτύχθηκε ένα σύστημα βασισμένο στον αλγόριθμο XGBoost (Extreme Gradient Boosting), το οποίο εκπαιδεύεται σε ζεύγη περιγραφών προϊόντων και αξιοποιεί χαρακτηριστικά που προκύπτουν τόσο από τη σημασιολογική αναπαράσταση των κειμένων όσο και από λεκτικές μετρικές ομοιότητας. Η εκπαίδευση πραγματοποιείται μία φορά σε σύνολο επισημασμένων δεδομένων (training dataset), ενώ το παραγόμενο μοντέλο αποθηκεύεται και χρησιμοποιείται στη συνέχεια για την αντιστοίχιση νέων προϊόντων χωρίς να απαιτείται νέα διαδικασία εκπαίδευσης.

Η προσέγγιση αυτή επιτρέπει την αξιολόγηση της αποτελεσματικότητας ενός καθαρά Machine Learning μοντέλου και τη σύγκρισή του με την υβριδική αρχιτεκτονική που παρουσιάστηκε στα προηγούμενα κεφάλαια, η οποία συνδυάζει embeddings, επιχειρησιακούς κανόνες και LLMs. Μέσα από τα πειραματικά αποτελέσματα αναδεικνύονται τόσο οι δυνατότητες όσο και οι περιορισμοί της εποπτευόμενης μάθησης στο πρόβλημα της αντιστοίχισης προϊόντων, ιδιαίτερα όταν το διαθέσιμο σύνολο εκπαίδευσης είναι περιορισμένο.

Στις επόμενες ενότητες παρουσιάζονται αναλυτικά η διαδικασία εκπαίδευσης του μοντέλου XGBoost, η δημιουργία των χαρακτηριστικών (feature engineering), η χρήση του εκπαιδευμένου μοντέλου για την αντιστοίχιση νέων προϊόντων, καθώς και η αξιολόγηση των αποτελεσμάτων σε πραγματικά παραδείγματα.

3.4.1 Εισαγωγικό Πρακτικό Παράδειγμα Εκτέλεσης XGBoost. Supervised M.L.

Σε αυτό το υπό κεφάλαιο θα δούμε ένα ενδεικτικό παράδειγμα λειτουργίας του συστήματος αντιστοίχισης προϊόντων που βασίζεται στην εποπτευόμενη Μηχανική Μάθηση (Supervised Machine Learning). Για τον σκοπό αυτό χρησιμοποιείται ο αλγόριθμος XGBoost, ο οποίος εκπαιδεύεται ώστε να προβλέπει εάν δύο περιγραφές προϊόντων αντιστοιχούν στο ίδιο προϊόν ή όχι.

Η εκπαίδευση υλοποιείται πάνω σε ζεύγη περιγραφών προϊόντων, από τα οποία εξάγονται χαρακτηριστικά ομοιότητας (features), όπως η σημασιολογική ομοιότητα μέσω cosine similarity, η διαφορά μήκους των περιγραφών (length difference) και το ποσοστό κοινών λέξεων (word overlap). Με βάση τα χαρακτηριστικά αυτά, το μοντέλο μαθαίνει να ταξινομεί κάθε ζεύγος ως θετική (label = 1) ή αρνητική αντιστοίχιση (label = 0).

Μετά την ολοκλήρωση της εκπαίδευσης, το μοντέλο χρησιμοποιείται για την αντιστοίχιση νέων περιγραφών προϊόντων με τις ήδη γνωστές εγγραφές της βάσης αναφοράς. Για κάθε πιθανό ζεύγος υπολογίζεται μία πιθανότητα αντιστοίχισης (ML Probability), η οποία χρησιμοποιείται για τη λήψη της τελικής απόφασης (Yes/No).

Ο παρακάτω πίνακας παρουσιάζει ενδεικτικά αποτελέσματα της όλης διαδικασίας.

Import	Matched ProductID	Matched Description	CosineScore	ML_Probability	Final
Dell - Ryzen 7 6800H - 13.3" FHD - 8GB - 512GB SSD - AMD Radeon Graphics - Win11 Pro	21	Lenovo Aspire 5 - Ryzen 7 6800H - 13.3" FHD - 8GB - 512GB SSD - AMD Radeon Graphics - Win11 Pro	0,843500018	0,181600004	No
Lenovo Aspire 5, Ryzen 7 6800H/13.3" FHD/8GB/512GB SSD/AMD Radeon Graphics/Win11 Pro	10	Lenovo XPS 13, Ryzen 5 5500U/15.6" FHD 144Hz/8GB/512GB SSD/AMD Radeon/Win10 Home	0,905099988	0,219899997	No
Lenovo Aspire 5 / Ryzen 7 6800H / 13.3" FHD / 8GB / 512GB SSD / AMD Radeon Graphics / Win11 Pro	21	Lenovo Aspire 5 - Ryzen 7 6800H - 13.3" FHD - 8GB - 512GB SSD - AMD Radeon Graphics - Win11 Pro	1	0,989600003	Yes

Από τα αποτελέσματα παρατηρείται ότι το μοντέλο μπορεί να αναγνωρίσει με υψηλή πιθανότητα τις περιπτώσεις όπου δύο περιγραφές αναφέρονται στο ίδιο προϊόν, ακόμη και όταν υπάρχουν μικρές διαφοροποιήσεις στη μορφοποίηση. Παράλληλα, απορρίπτει περιγραφές που παρουσιάζουν σημαντικές διαφορές στα τεχνικά χαρακτηριστικά ή αφορούν διαφορετικά προϊόντα. Ωστόσο, εμφανίζονται και περιπτώσεις λανθασμένης πρόβλεψης, γεγονός που καταδεικνύει ότι η απόδοση ενός μοντέλου εποπτευόμενης μάθησης εξαρτάται σε μεγάλο βαθμό από την ποιότητα και το μέγεθος του συνόλου εκπαίδευσης.

Η συνολική διαδικασία που ακολουθεί το σύστημα συνοπτικά είναι:

Περιγραφή Προϊόντος → Embeddings embedding_cache_ml.pkl → Feature Engineering
read LLMEmbPromptLearn.xlsx → XGBoost product_matcher_xgb.pkl → Probability →
ProductID LLMEmbMatch_ML_matched.xlsx

Στις επόμενες ενότητες παρουσιάζονται αναλυτικά η διαδικασία εκπαίδευσης του μοντέλου, η δημιουργία των χαρακτηριστικών, η αποθήκευση του εκπαιδευμένου μοντέλου και η εφαρμογή του στην αντιστοίχιση νέων προϊόντων.

3.4.2 Στάδιο Προεκπαίδευσης Μοντέλου Μηχανικής Μάθησης.

Σε αντίθεση με το Train_ProductID.py, το οποίο βασίζεται στη σημασιολογική ομοιότητα των embeddings σε χειροκίνητο αρχείο.xlsx και στην επαλήθευση μέσω LLM χωρίς να πραγματοποιεί εκπαίδευση μοντέλου, το Train_productid_ML.py υλοποιεί διαδικασία εποπτευόμενης μάθησης (Supervised Machine Learning). Σκοπός του είναι η εκπαίδευση ενός μοντέλου XGBoost, το οποίο θα μπορεί να προβλέπει εάν δύο περιγραφές προϊόντων αντιστοιχούν στο ίδιο προϊόν.

Αρχικά, το σύστημα διαβάζει το αρχείο 0LLMEmbPrompt3Learn.xlsx, το οποίο χρησιμοποιείται ως σύνολο εκπαίδευσης (training dataset). Για κάθε εγγραφή ανακτάται η περιγραφή του προϊόντος (Import) και το αντίστοιχο ID προϊόντος (Correct Product ID). Οι περιγραφές μετατρέπονται σε διανυσματικές αναπαραστάσεις (embeddings) μέσω του μοντέλου all-MiniLM-L6-v2, ενώ εφαρμόζεται μηχανισμός προσωρινής αποθήκευσης (embedding cache), ώστε να αποφεύγεται ο επανυπολογισμός embeddings για περιγραφές που έχουν ήδη επεξεργαστεί.

Στη συνέχεια δημιουργούνται αυτόματα όλα τα δυνατά ζεύγη περιγραφών προϊόντων (product pairs). Για κάθε ζεύγος υπολογίζονται τρία χαρακτηριστικά (features), τα οποία αποτελούν το διάνυσμα εισόδου του μοντέλου:

1. Cosine Similarity: Εκφράζει τη σημασιολογική ομοιότητα μεταξύ των embeddings.
2. Length Difference: Υπολογίζει τη διαφορά μήκους μεταξύ των δύο περιγραφών.
3. Word Overlap Ratio (Jaccard Similarity): Εκφράζει το ποσοστό κοινών λέξεων μεταξύ των δύο περιγραφών.

Παράλληλα δημιουργείται και η ετικέτα εκπαίδευσης (label), η οποία χρησιμοποιείται κατά την εποπτευόμενη μάθηση:

- Label = 1: Όταν οι δύο περιγραφές έχουν το ίδιο Product ID.
- Label = 0: Όταν αντιστοιχούν σε διαφορετικά προϊόντα.

Με τον τρόπο αυτό δημιουργείται το εποπτευόμενο σύνολο δεδομένων (X, y), όπου το X περιέχει τα χαρακτηριστικά κάθε ζεύγους προϊόντων και το y την έξοδο όπου μαθαίνει το μοντέλο.

Στη συνέχεια το σύνολο δεδομένων χωρίζεται σε σύνολο εκπαίδευσης (Training Set) και σύνολο ελέγχου (Test Set) μέσω της συνάρτησης `train_test_split()`, ώστε να αξιολογηθεί η ικανότητα γενίκευσης του μοντέλου σε δεδομένα που δεν χρησιμοποιήθηκαν κατά την εκπαίδευση.

Για την εκπαίδευση χρησιμοποιήθηκε ο αλγόριθμος XGBoost (Extreme Gradient Boosting), ένας από τους πλέον διαδεδομένους αλγορίθμους εποπτευόμενης μάθησης για προβλήματα ταξινόμησης. Το μοντέλο εκπαιδεύεται στα παραγόμενα χαρακτηριστικά και μαθαίνει ποιοι συνδυασμοί τιμών αντιστοιχούν σε ίδια ή διαφορετικά προϊόντα. Μετά την ολοκλήρωση της εκπαίδευσης αξιολογείται μέσω του Classification Report, ενώ στη συνέχεια αποθηκεύεται στο αρχείο `product_matcher_xgb.pkl`.

Το αρχείο `.pkl` (Pickle File) αποτελεί τη μόνιμη αποθήκευση του εκπαιδευμένου μοντέλου και περιέχει όλες τις παραμέτρους που υπολογίστηκαν κατά την εκπαίδευση. Η αποθήκευση αυτή επιτρέπει την επαναχρησιμοποίηση του μοντέλου χωρίς να απαιτείται νέα διαδικασία εκπαίδευσης.

Η διαδικασία χωρίζεται ουσιαστικά σε δύο διακριτές φάσεις:

Training Phase:

- Δημιουργία embeddings.
- Δημιουργία χαρακτηριστικών (Feature Engineering).
- Δημιουργία labels.
- Εκπαίδευση του μοντέλου XGBoost.
- Αξιολόγηση του μοντέλου.
- Αποθήκευση του μοντέλου στο αρχείο `product_matcher_xgb.pkl`.

Inference Phase:

- Φόρτωση του αποθηκευμένου μοντέλου.
- Υπολογισμός χαρακτηριστικών για νέα προϊόντα.
- Πρόβλεψη πιθανότητας αντιστοίχισης χωρίς νέα εκπαίδευση.

Η βασική διαφορά του συγκεκριμένου script σε σχέση με τα προηγούμενα στάδια της εργασίας είναι ότι δεν πραγματοποιεί άμεσα αντιστοίχιση προϊόντων. Αντίθετα, δημιουργεί ένα εκπαιδευμένο μοντέλο Μηχανικής Μάθησης, το οποίο χρησιμοποιείται στη συνέχεια για την πρόβλεψη αντιστοιχίσεων σε νέα δεδομένα. Ενώ τα προηγούμενα scripts βασίζονταν κυρίως στη σημασιολογική ομοιότητα των embeddings και στην επαλήθευση μέσω LLM, στο στάδιο αυτό το σύστημα μαθαίνει από ιστορικά παραδείγματα και αποκτά τη δυνατότητα να γενικεύει τη γνώση του σε νέες περιπτώσεις αντιστοίχισης προϊόντων.

Ουσιαστικά το μοντέλο εκπαιδεύεται αποκλειστικά από το αρχείο excel χωρίς SQL και χωρίς API. Δημιουργεί όλους τους δυνατούς συνδυασμούς ζευγών ($i \times j$) και μαθαίνει να διακρίνει ίδια από διαφορετικά προϊόντα βάσει τριών χαρακτηριστικών.

Παράδειγμα:

POCO C75 6+128GB vs POCO C75 6+128GB → label 1 (ίδιο ID)

POCO C75 6+128GB vs POCO X6 Pro 12+512 → label 0 (διαφορετικό ID)

POCO X6 Pro 12+512 vs POCO C75 6+128GB → label 0

POCO X6 Pro 12+512 vs POCO X6 Pro 12+512 → label 1 ...

```
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
Loading weights: 100%|██████████| 103/103 [00:00<00:00, 3116.55it/s]
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key | Status |
-----+-----+
embeddings.position_ids | UNEXPECTED |

Notes:
- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.
Loaded rows: 1101
Creating pairs: 100%|██████████| 1101/1101 [09:05<00:00, 2.02it/s]
Dataset size: (1211100, 3)
Training model...

Classification Report:
      precision    recall  f1-score   support
0               1.00        1.00        1.00      240821
1               0.98        0.24        0.38       1399

 accuracy               1.00      242220
 macro avg              0.99        0.62        0.69      242220
weighted avg              1.00        1.00        0.99      242220

Model saved: product_matcher_xgb.pkl
Process finished with exit code 0
```

Κατά την εκτέλεση του προγράμματος αρχικά φορτώνεται το προεκπαιδευμένο γλωσσικό μοντέλο SentenceTransformer (all-MiniLM-L6-v2), το οποίο χρησιμοποιείται για τη μετατροπή των περιγραφών προϊόντων σε διανυσματικές αναπαραστάσεις (embeddings). Τα προειδοποιητικά μηνύματα (warnings) που εμφανίζονται κατά τη φόρτωση του μοντέλου αφορούν εσωτερικές διαφορές της αρχιτεκτονικής του μοντέλου και δεν επηρεάζουν τη σωστή λειτουργία του συστήματος.

Στη συνέχεια διαβάζονται συνολικά 1.101 εγγραφές από το αρχείο εκπαίδευσης. Για κάθε πιθανό ζεύγος περιγραφών δημιουργούνται αυτόματα τα αντίστοιχα χαρακτηριστικά (cosine similarity, length difference και word overlap), με αποτέλεσμα να παραχθεί ένα σύνολο εκπαίδευσης αποτελούμενο από $1.101 \times 1.101 = 1.211.100$ ζεύγη προϊόντων.

Έπειτα εκτελείται η διαδικασία εκπαίδευσης του ταξινομητή XGBoost, ο οποίος μαθαίνει να διακρίνει εάν δύο περιγραφές αντιστοιχούν στο ίδιο προϊόν ή σε διαφορετικά προϊόντα. Μετά την ολοκλήρωση της εκπαίδευσης πραγματοποιείται αξιολόγηση του μοντέλου μέσω του Classification Report, στο οποίο παρουσιάζονται οι δείκτες Precision, Recall, F1-score και Accuracy.

Στο συγκεκριμένο πείραμα το μοντέλο παρουσίασε πολύ υψηλή συνολική ακρίβεια (Accuracy $\approx 100\%$) για την κλάση των μη αντιστοιχίσεων (label = 0). Αντίθετα, για την κλάση των πραγματικών αντιστοιχίσεων (label = 1) το Recall παρέμεινε σχετικά χαμηλό 24%, γεγονός που υποδηλώνει ότι αρκετές πιθανές πραγματικές αντιστοιχίσεις δεν αναγνωρίστηκαν επιτυχώς. Η συμπεριφορά αυτή οφείλεται κυρίως στη σημαντική ανισορροπία του συνόλου εκπαίδευσης, όπου τα αρνητικά παραδείγματα ήταν πολύ περισσότερα από τα θετικά. Το εκπαιδευμένο μοντέλο αποθηκεύεται στο αρχείο product_matcher_xgb.pkl, ώστε να μπορεί να χρησιμοποιηθεί στα επόμενα στάδια της διαδικασίας για την αντιστοίχιση νέων προϊόντων χωρίς να απαιτείται επανεκπαίδευση.

Μετά την παραπάνω διαπίστωση εφαρμόστηκε επιπλέον ο μηχανισμός `scale_pos_weight=neg/pos` του XGBoost, με στόχο την αντιμετώπιση της ανισορροπίας των δεδομένων εκπαίδευσης (class imbalance). Στο σύνολο εκπαίδευσης ο αριθμός των αρνητικών ζευγών προϊόντων (διαφορετικά προϊόντα) ήταν σημαντικά μεγαλύτερος από τον αριθμό των θετικών ζευγών (ίδια προϊόντα). Για τον λόγο αυτό, η παράμετρος `scale_pos_weight` υπολογίστηκε ως ο λόγος των αρνητικών προς τα θετικά δείγματα (`neg/pos`), αυξάνοντας το βάρος των θετικών παραδειγμάτων κατά τη διαδικασία μάθησης. Με τον τρόπο αυτό το μοντέλο δεν ευνοεί υπερβολικά την πλειοψηφική κλάση, αλλά εκπαιδεύεται ώστε να αναγνωρίζει αποτελεσματικότερα τις πραγματικές αντιστοιχίσεις προϊόντων. Η τεχνική αυτή αποτελεί καθιερωμένη πρακτική σε προβλήματα δυαδικής ταξινόμησης με έντονα μη ισορροπημένα δεδομένα και συμβάλλει στη βελτίωση της ικανότητας γενίκευσης του μοντέλου, ιδιαίτερα ως προς την αναγνώριση της θετικής κλάσης (Product Match).

```
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
Loading weights: 100%|██████████| 103/103 [00:00<00:00, 8217.87it/s]
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key | Status | |
-----+-----+
embeddings.position_ids | UNEXPECTED | |

Notes:
- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.
Creating pairs: 0%|██████████| 0/1101 [00:00<?, ?it/s]Loaded rows: 1101
Creating pairs: 100%|██████████| 1101/1101 [05:47<00:00, 3.17it/s]
Dataset size: (1211100, 3)
Class balance -> neg: 963577, pos: 5303, ratio: 181.7
Training model...

Classification Report:
      precision    recall  f1-score   support

     0       1.00      0.93      0.97     240821
     1       0.08      0.96      0.15       1399

 accuracy          0.93     242220
 macro avg          0.54      0.95      0.56     242220
 weighted avg          0.99      0.93      0.96     242220

Model saved: product_matcher_xgb.pkl

Process finished with exit code 0
```

Κατά την εκτέλεση του προγράμματος φορτώνεται αρχικά το προεκπαιδευμένο μοντέλο SentenceTransformer (all-MiniLM-L6-v2), το οποίο βασίζεται στην αρχιτεκτονική BERT και χρησιμοποιείται για τη μετατροπή των περιγραφών προϊόντων σε σημασιολογικά διανύσματα (embeddings).

Διαβάζονται 1.101 εγγραφές από το σύνολο εκπαίδευσης και δημιουργούνται αυτόματα 1.211.100 ζεύγη προϊόντων, από τα οποία 963.577 αντιστοιχούν σε αρνητικά παραδείγματα (διαφορετικά προϊόντα) και μόλις 5.303 σε θετικά παραδείγματα (ίδια προϊόντα). Η αναλογία περίπου 182:1 καταδεικνύει την έντονη ανισορροπία (class imbalance) του συνόλου εκπαίδευσης.

Για την αντιμετώπιση του προβλήματος αυτού χρησιμοποιήθηκε η παράμετρος `scale_pos_weight` του XGBoost, η οποία αυξάνει το βάρος της θετικής κλάσης κατά την εκπαίδευση. Σε σύγκριση με το προηγούμενο πείραμα, όπου το μοντέλο εμφάνιζε Recall = 0,24 για τις πραγματικές αντιστοιχίσεις προϊόντων, η νέα εκπαίδευση αύξησε σημαντικά το Recall σε 0,96, γεγονός που δείχνει ότι το μοντέλο αναγνώρισε πλέον σχεδόν όλες τις πραγματικές αντιστοιχίσεις.

Ωστόσο, η σημαντική αυτή βελτίωση συνοδεύτηκε από αισθητή μείωση του Precision της θετικής κλάσης ($0,98 \rightarrow 0,08$), καθώς το μοντέλο χαρακτήρισε ως ίδιες πολλές περιγραφές που στην πραγματικότητα αντιστοιχούσαν σε διαφορετικά προϊόντα. Παράλληλα, η συνολική ακρίβεια (Accuracy) μειώθηκε από περίπου 100% σε 93%. Το αποτέλεσμα αυτό δείχνει ότι η υπερβολική αντιστάθμιση της ανισορροπίας οδηγεί το μοντέλο να ευνοεί υπέρμετρα την αναγνώριση της θετικής κλάσης, αυξάνοντας σημαντικά τα ψευδώς θετικά αποτελέσματα (False Positives).

Συνεπώς, η χρήση του `scale_pos_weight` βελτιώνει σημαντικά την ικανότητα εντοπισμού πραγματικών αντιστοιχίσεων, απαιτεί όμως προσεκτική ρύθμιση των παραμέτρων και του ορίου απόφασης (decision threshold), ώστε να επιτευχθεί η βέλτιστη ισορροπία μεταξύ Precision και Recall ανάλογα με τις απαιτήσεις της εφαρμογής.

Αρα με δεδομένα τα παραπάνω τελικά το σύστημα εκπαίδευσης του XGBoost υλοποιήθηκε με δύο εναλλακτικές λειτουργικές ρυθμίσεις, επιτρέποντας την επιλογή κατά την εκτέλεση. Συγκεκριμένα, παρέχεται η δυνατότητα επιλογής μεταξύ δύο modes:

1. Χρήσης της παραμέτρου `scale_pos_weight`, η οποία αντιμετωπίζει την ανισορροπία των κλάσεων ενισχύοντας τη βαρύτητα της θετικής κλάσης.
2. Βασική διαμόρφωση του μοντέλου χωρίς αντιστάθμιση ανισορροπίας.

Με αυτόν τον τρόπο το σύστημα επιτρέπει πειραματική σύγκριση των δύο προσεγγίσεων και προσαρμογή της εκπαίδευσης ανάλογα με τα χαρακτηριστικά του εκάστοτε dataset και τις απαιτήσεις ακρίβειας ή ανάκλησης του προβλήματος αντιστοίχισης προϊόντων.

```
Loading weights: 100%|██████████| 103/103 [00:00<00:00, 6133.16it/s]
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key | Status | |
-----+-----+---
embeddings.position_ids | UNEXPECTED | |

Notes:
- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.
Loaded rows: 1113
Creating pairs: 100%|██████████| 1113/1113 [05:49<00:00, 3.19it/s]
Dataset size: (1237656, 3)

Choose training mode:
  1 → scale_pos_weight (fixes class imbalance - recommended)
  2 → use_label_encoder=False (original mode)
Enter 1 or 2: 1
Class balance → neg: 984757, pos: 5367, ratio: 183.5
Training model...

Classification Report:
      precision    recall  f1-score   support

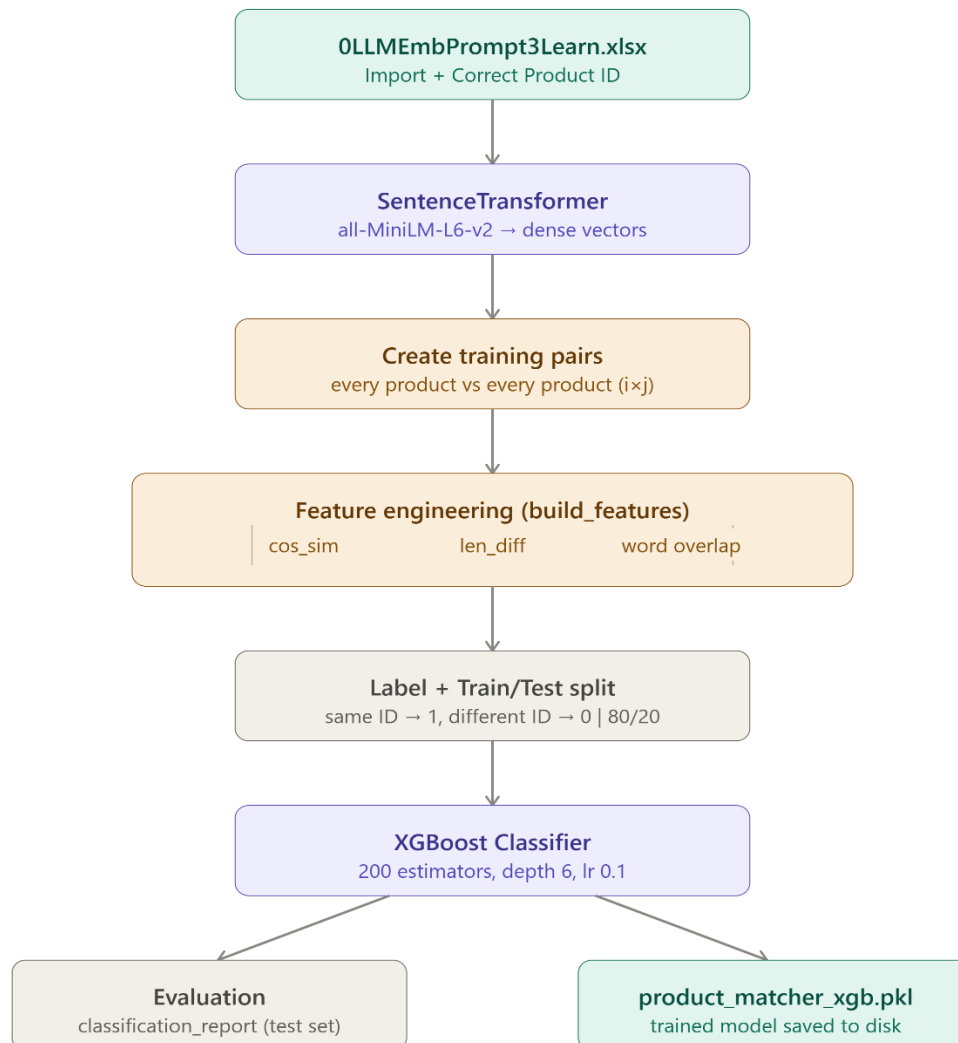
     0       1.00      0.93      0.97     246153
     1       0.08      0.97      0.14       1379

   accuracy      0.93      0.93     247532
  macro avg       0.54      0.95      0.55     247532
 weighted avg       0.99      0.93      0.96     247532

Model saved: product_matcher_xgb.pkl

Process finished with exit code 0
```

Flowchart Train productid ML.py:



Ο κάτωθι πίνακας παρουσιάζει τα αποτελέσματα του συστήματος όταν το μοντέλο XGBoost έχει εκπαιδευτεί με τη χρήση της 1^{ης} επιλογής - παραμέτρου `scale_pos_weight`, δηλαδή με αντιστάθμιση της ανισορροπίας των κλάσεων κατά τη δημιουργία του αρχείου `product_matcher_xgb.pkl`.

Import	Matched ProductID	Matched Description	CosineScore	ML_Probability	Final
15.6" FHD 144Hz/16GB ram/256GB SSD/Win10 Home/RTX3050/ MSI Pavilion 15, Ryzen 7 6800H	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	0,855000019	0,939199984	Yes
Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 64GB ram - 512GB SSD - NVIDIA RTX3060 - Win11 Home	25	Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB - 512GB SSD - NVIDIA RTX3060 - Win11 Home	0,986800015	0,993300021	Yes
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	0,993099988	0,999800026	Yes
galaxy samsung S22 1028GB 6.1" OLED Tensor G2 Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	0,984600008	0,980400026	Yes
galaxy samsung S22 512GB 6.1" OLED Tensor G2 // Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	0,991699994	0,999300003	Yes
galaxy samsung S22 512GB 6.1" OLED Tensor G2 // Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	0,925499976	0,907800019	Yes

Ο επόμενος πίνακας παρουσιάζει αποτελέσματα της 2^{ης} επιλογής όταν το μοντέλο εκπαιδεύεται με την αρχική διαμόρφωση (original mode), χωρίς χρήση της παραμέτρου scale_pos_weight.

Import	Matching ID	Matching Name	Embedding Score	Gemini Verification	Final Match
15.6" FHD 144Hz/16GB ram/256GB SSD/Win10 Home/RTX3050/ MSI Pavilion 15, Ryzen 7 6800H	1	MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB/256GB SSD/RTX3050/Win10 Home	95,05000305	Yes	Yes
Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 64GB ram - 512GB SSD - NVIDIA RTX3060 - Win11 Home	25	Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB - 512GB SSD - NVIDIA RTX3060 - Win11 Home	98,68000031	No	No
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	99,30999756	Yes	Yes
galaxy samsung S22 1028GB 6.1" OLED Tensor G2 Android	28	Samsung Galaxy S22 128GB 6.1" OLED Tensor G2 Android	98,75	No	No
galaxy samsung S22 512GB 6.1" OLED Tensor G2 // Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	99,16999817	Yes	Yes
galaxy samsung S12 522GB 6.1" OLED Tensor G2 // Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	92,55000305	No	No

Αποτελέσματα που εξάγονται από το επόμενο script που θα εξετάσουμε παρακάτω αλλά που έχει προηγηθεί εδώ η 2^η επιλογή original mode.

Import	Matched ProductID	Matched Description	CosineScore	ML_Probability	Final
15.6" FHD 144Hz/16GB ram/256GB SSD/Win10 Home/RTX3050/ MSI Pavilion 15, Ryzen 7 6800H	25	Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB - 512GB SSD - NVIDIA RTX3060 - Win11 Home	0,864700019	0,141100004	No
Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 64GB ram - 512GB SSD - NVIDIA RTX3060 - Win11 Home	25	Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB - 512GB SSD - NVIDIA RTX3060 - Win11 Home	0,986800015	0,690400004	Yes
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	0,993099988	0,99059999	Yes
galaxy samsung S22 1028GB 6.1" OLED Tensor G2 Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	0,984600008	0,223499998	No
galaxy samsung S22 512GB 6.1" OLED Tensor G2 // Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	0,991699994	0,981599987	Yes
galaxy samsung S12 522GB 6.1" OLED Tensor G2 // Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	0,925499976	0,052099999	No

Η 2^η επιλογή (original mode) παρουσίασε καλύτερη συμπεριφορά στο συγκεκριμένο σύνολο δεδομένων, καθώς συμφώνησε περισσότερο με τις αντιστοιχίσεις που είχαν προηγουμένως επαληθευθεί μέσω του υβριδικού συστήματος Embeddings + LLM. Αντίθετα, η πρώτη επιλογή (scale_pos_weight) παρήγαγε δύο ψευδώς θετικές αντιστοιχίσεις (False Positives), αποδίδοντας υψηλή πιθανότητα ταύτισης σε προϊόντα που δεν αντιστοιχούσαν στο ίδιο αντικείμενο.

Η διαφορά αυτή αποδίδεται στον διαφορετικό τρόπο εκπαίδευσης του μοντέλου XGBoost. Η χρήση της παραμέτρου scale_pos_weight αυξάνει τη βαρύτητα της θετικής κλάσης κατά την εκπαίδευση, με αποτέλεσμα το μοντέλο να ευνοεί περισσότερο τις θετικές προβλέψεις. Στο συγκεκριμένο σύνολο δεδομένων η συμπεριφορά αυτή οδήγησε σε αυξημένο αριθμό ψευδώς θετικών αντιστοιχίσεων, ενώ η αρχική διαμόρφωση του μοντέλου (original mode) παρείχε πιο συντηρητικές και τελικά ακριβέστερες προβλέψεις. Θα προχωρήσουμε επιλέγοντας την 2^η επιλογή. use_label_encoder=False (original mode).

```
# Train_productid_ML.py: only train ML and creates product_matcher_xgb.pkl.
import warnings
warnings.filterwarnings("ignore")
import os
import pandas as pd
import numpy as np
import joblib
from tqdm import tqdm
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sentence_transformers import SentenceTransformer
from xgboost import XGBClassifier
# ===== CONFIG =====
MODEL_PATH = "product_matcher_xgb.pkl"
EMB_CACHE_PATH = "embedding_cache_ml.pkl"
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
# ===== CACHE =====
def load_cache(path):
    return joblib.load(path) if os.path.exists(path) else {}
def save_cache(cache, path):
    joblib.dump(cache, path)
# ===== EMBEDDINGS =====
def get_embedding(text, cache):
    text = str(text)
    if text in cache:
        return cache[text]
    emb = embedding_model.encode(text)
    cache[text] = emb
    return emb
# ===== FEATURE ENGINEERING =====
def build_features(text1, text2, emb1, emb2):
    # cosine similarity
    cos_sim = cosine_similarity([emb1], [emb2])[0][0]
    # length difference
    len_diff = abs(len(text1) - len(text2))
    # common words ratio
    set1 = set(text1.lower().split())
    set2 = set(text2.lower().split())
    if len(set1.union(set2)) == 0:
        overlap = 0
    else:
        overlap = len(set1.intersection(set2)) / len(set1.union(set2))
    return [cos_sim, len_diff, overlap]
# ===== CREATE TRAIN DATA =====
def create_training_pairs(df, cache):
    pairs = []
    labels = []
    texts = df["Import"].astype(str).tolist()
    pids = df["Correct Product ID"].tolist()
    # embeddings
    embeddings = [get_embedding(t, cache) for t in texts]
    for i in tqdm(range(len(texts)), desc="Creating pairs"):
        for j in range(len(texts)):
            if i == j:
                continue
            f = build_features(texts[i], texts[j], embeddings[i], embeddings[j])
            # label
            if pids[i] == pids[j]:
                label = 1 # same product
            else:
                label = 0 # different
            pairs.append(f)
            labels.append(label)
    return np.array(pairs), np.array(labels)
# ===== MAIN =====
if __name__ == "__main__":
    excel_path = "C:\Users\lgit\Desktop\lg\Edu\Thesis\0code\0LLMEmbPrompt3Learn.xlsx"
    sheet = "SIGPT"
    cache = load_cache(EMB_CACHE_PATH)
    df = pd.read_excel(excel_path, sheet_name=sheet)
    df = df.dropna(subset=["Correct Product ID"])
    print(f"Loaded rows: {len(df)}")
    # create dataset
    X, y = create_training_pairs(df, cache)
    save_cache(cache, EMB_CACHE_PATH)
    print(f"Dataset size: {X.shape}")
    # split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )
    # ===== TRAIN MODEL =====
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )
    neg = np.sum(y_train == 0)
    pos = np.sum(y_train == 1) # for scale_pos_weight
    # add choose for pkl file
    print("\nChoose training mode:")
    print("  1 -> scale_pos_weight (fixes class imbalance - recommended)")
    print("  2 -> use_label_encoder=False (original mode)")
    choice = input("Enter 1 or 2: ").strip()
    if choice == "1":
        print(f"Class balance -> neg: {neg}, pos: {pos}, ratio: {neg/pos:.1f}")
        model = XGBClassifier(
            n_estimators=200,
            max_depth=6,
            learning_rate=0.1,
            scale_pos_weight=neg/pos,
            eval_metric='logloss'
        )
    else:
        model = XGBClassifier(
            n_estimators=200,
            max_depth=6,
            learning_rate=0.1,
            use_label_encoder=False,
            eval_metric='logloss'
        )
    # model = XGBClassifier(
    #     n_estimators=200,
    #     max_depth=6,
    #     learning_rate=0.1,
    #     scale_pos_weight=neg/pos, # ← dynamic calculate. or scale_pos_weight=240821/1399, before use_label_encoder=False, replace
    #     eval_metric='logloss'
    # )
    # print(f"Class balance -> neg: {neg}, pos: {pos}, ratio: {neg / pos:.1f}") #scale_pos_weight
    print(f"Training model...")
    model.fit(X_train, y_train)
    # ===== EVALUATION =====
    y_pred = model.predict(X_test)
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
    # ===== SAVE MODEL =====
    joblib.dump(model, MODEL_PATH)
    print(f"\n Model saved: {MODEL_PATH}")
```

3.4.3 Αντιστοίχιση προϊόντων με το Μοντέλο Μηχανικής Μάθησης.

Το στάδιο αυτό αποτελεί τη φάση παραγωγικής λειτουργίας του προτεινόμενου συστήματος Μηχανικής Μάθησης. Σε αντίθεση με το προηγούμενο στάδιο (Train_ProductID_ML.py), όπου πραγματοποιείται η εκπαίδευση του μοντέλου XGBoost, στο παρόν στάδιο χρησιμοποιείται το ήδη εκπαιδευμένο μοντέλο (product_matcher_xgb.pkl) για την πρόβλεψη και αντιστοίχιση νέων περιγραφών προϊόντων με τα προϊόντα της βάσης αναφοράς. Κατά συνέπεια, δεν πραγματοποιείται νέα εκπαίδευση του μοντέλου, αλλά αποκλειστικά διαδικασία πρόβλεψης (prediction/inference).

Συγκεκριμένα, το σύστημα φορτώνει το μοντέλο XGBoost που έχει δημιουργηθεί κατά τη διαδικασία εκπαίδευσης, καθώς και το σύνολο γνωστών προϊόντων από το αρχείο LLMEmbPromptLearn.xlsx, το οποίο λειτουργεί ως βάση αναφοράς (Reference Dataset). Παράλληλα, διαβάζονται οι νέες περιγραφές προϊόντων από το αρχείο LLMEmbMatch.xlsx, οι οποίες αποτελούν τα προϊόντα προς αντιστοίχιση.

Δημιουργία χαρακτηριστικών (Feature Engineering):

Για κάθε νέο προϊόν υπολογίζονται τρία βασικά χαρακτηριστικά έναντι κάθε προϊόντος της βάσης αναφοράς:

1. Cosine Similarity: Υπολογίζει σημασιολογική ομοιότητα embed. 2 περιγραφών.
2. Length Difference: Υπολογίζει τη διαφορά μήκους μεταξύ των 2 περιγραφών.
3. Word Overlap: Υπολογίζει το ποσοστό κοινών λέξεων μεταξύ των 2 περιγραφών χρησιμοποιώντας μέτρο Jaccard Similarity.

Τα χαρακτηριστικά αυτά αποτελούν το διάνυσμα εισόδου του μοντέλου XGBoost.

Το μοντέλο XGBoost έχει εκπαιδευτεί να προβλέπει αν δύο περιγραφές αναφέρονται στο ίδιο προϊόν ή όχι.

Για κάθε πιθανό ζεύγος:

δημιουργούνται τα τρία χαρακτηριστικά, εκτελείται πρόβλεψη μέσω της μεθόδου predict_proba(), υπολογίζεται η πιθανότητα οι δύο περιγραφές να αντιστοιχούν στο ίδιο προϊόν.

Η πιθανότητα αυτή αποθηκεύεται στη μεταβλητή:

```
prob = model.predict_proba(features)[0][1]
```

όπου:

- Τιμές κοντά στο 0 υποδηλώνουν διαφορετικά προϊόντα,
- Τιμές κοντά στο 1 υποδηλώνουν ταύτιση προϊόντων.

Επιλογή βέλτιστης αντιστοίχισης:

Για κάθε νέο προϊόν το σύστημα συγκρίνει διαδοχικά όλες τις εγγραφές της βάσης αναφοράς και διατηρεί:

- Το Product ID με τη μεγαλύτερη πιθανότητα.
- Την αντίστοιχη περιγραφή.
- Το cosine similarity score.
- Την τελική πιθανότητα που υπολόγισε το XGBoost.

Ουσιαστικά εφαρμόζεται μία διαδικασία τύπου:

Best Match Search, όπου επιλέγεται η εγγραφή με το μεγαλύτερο ML probability.

Κανόνας λήψης απόφασης: Η τελική απόφαση βασίζεται σε ένα κατώφλι πιθανότητας:

final = "Yes" if best_prob $\geq$ 0.50 else "No"

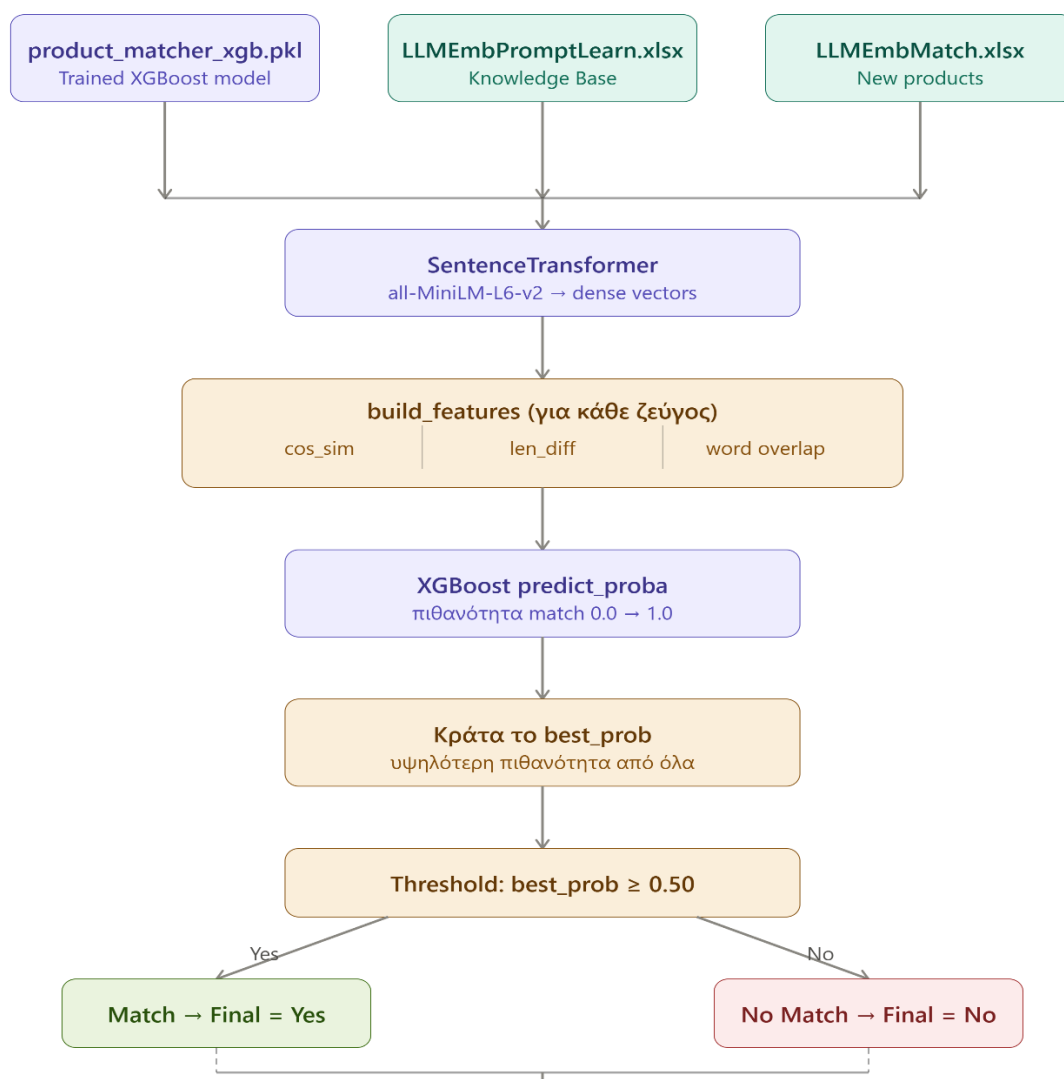
Επομένως:

- Probability $\geq$ 50% $\rightarrow$ Match (Yes)
- Probability $<$ 50% $\rightarrow$ No Match (No)

Η πιθανότητα δεν προέρχεται από το Cosine Similarity αλλά από το μοντέλο XGBoost, το οποίο συνδυάζει και τα τρία χαρακτηριστικά ταυτόχρονα.

Συμπαραστατικά σε αντίθεση με τα προηγούμενα στάδια που βασίζονται σε κανόνες ή σε επιβεβαίωση μέσω LLM, εδώ η τελική απόφαση λαμβάνεται αποκλειστικά από το μοντέλο Μηχανικής Μάθησης, το οποίο έχει εκπαιδευτεί σε ιστορικά δεδομένα αντιστοίχισης προϊόντων και επιστρέφει πιθανότητα ταύτισης για κάθε υποψήφιο προϊόν.

Flowchart Train 2 predict_ml_product matcher.py:



Πρακτικά παραδείγματα:

1.

Import	Matched ProductID	Matched Description	CosineScore	ML_Probability	Final
MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB ram/256GB SSD/RTX3050/Win10 Home	2	MSI XPS 13, Ryzen 5 5500U/13.3" FHD/16GB/1TB SSD/Intel Iris Xe/Win11 Pro	0,823800027	0,200800002	No
Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB ram - 512GB SSD - NVIDIA RTX3060 - Win11 Home	25	Asus Aspire 5 - Ryzen 7 5800H - 15.6" FHD 144Hz - 32GB - 512GB SSD - NVIDIA RTX3060 - Win11 Home	0,986800015	0,969900012	Yes
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	0,993099988	0,989600003	Yes
galaxy samsung S22 128GB 6.1" OLED Tensor G2 Android	28	Samsung Galaxy S22 128GB 6.1" OLED Tensor G2 Android	0,99879998	0,989600003	Yes
galaxy samsung S22 512GB 6.1" OLED Tensor G2 Android	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	0,998700023	0,989600003	Yes

Ο παραπάνω πίνακας απεικονίζει την τελική έξοδο του συστήματος Machine Learning αντιστοίχισης προϊόντων, όπου κάθε νέα περιγραφή (Import) συγκρίνεται με όλα τα προϊόντα της βάσης αναφοράς και καταλήγει σε μία πρόβλεψη αντιστοίχισης ή μη αντιστοίχισης. Για κάθε πιθανό ζεύγος προϊόντων υπολογίζονται τα χαρακτηριστικά εισόδου του μοντέλου (Cosine Similarity, Length Difference και Word Overlap), ενώ το XGBoost εκτιμά την πιθανότητα οι δύο περιγραφές να αντιστοιχούν στο ίδιο προϊόν. Ως τελικό αποτέλεσμα επιλέγεται η αντιστοίχιση με τη μεγαλύτερη πιθανότητα (ML Probability), η οποία χαρακτηρίζεται ως «Yes» ή «No» με βάση το επιλεγμένο κατώφλι πιθανότητας. Τα αποτελέσματα του συστήματος δείχνουν τη συνδυαστική αξιοποίηση της σημασιολογικής ομοιότητας (Cosine Similarity) και του εκπαιδευμένου μοντέλου XGBoost. Σε αντίθεση με τα προηγούμενα στάδια, το σύστημα δεν βασίζεται αποκλειστικά στη διανυσματική ομοιότητα των embeddings, αλλά χρησιμοποιεί ταυτόχρονα τρία χαρακτηριστικά (Cosine Similarity, Length Difference και Word Overlap), τα οποία αποτελούν την είσοδο του classifier. Η τελική απόφαση προκύπτει από την πιθανότητα που επιστρέφει το μοντέλο μέσω της συνάρτησης predict_proba(), με αρχικό κατώφλι απόφασης (threshold) ίσο με 0.50.

Ενδεικτικά, στην περίπτωση του «MSI Pavilion 15» το Cosine Similarity είναι σχετικά χαμηλό (≈ 0.82), γεγονός που υποδηλώνει περιορισμένη σημασιολογική ομοιότητα με το προϊόν που τελικά επιλέχθηκε. Αντίστοιχα, το μοντέλο XGBoost εκτιμά πιθανότητα περίπου 0.20 και επιστρέφει τελικό αποτέλεσμα «No», απορρίπτοντας την αντιστοίχιση. Αντίθετα, στις περιπτώσεις όπου το Cosine Similarity προσεγγίζει το 1 ($\approx 0.98-0.99$), το μοντέλο αποδίδει πολύ υψηλές πιθανότητες ($\approx 0.96-0.99$) και τελικό αποτέλεσμα «Yes», επιβεβαιώνοντας ότι οι δύο περιγραφές αναφέρονται στο ίδιο προϊόν.

Ωστόσο, θα πρέπει να επισημανθεί ότι στην περίπτωση του «MSI Pavilion 15» το μοντέλο δεν επέλεξε το πραγματικά σωστό προϊόν, παρότι αυτό υπήρχε στη βάση αναφοράς (Product ID = 1). Η συμπεριφορά αυτή πιθανότατα οφείλεται στο περιορισμένο μέγεθος και στη μικρή ποικιλία του εκπαιδευτικού συνόλου, καθώς και στον περιορισμένο αριθμό χαρακτηριστικών που χρησιμοποιήθηκαν κατά την εκπαίδευση. Τα εποπτευόμενα μοντέλα Μηχανικής Μάθησης απαιτούν συνήθως μεγάλο αριθμό αντιπροσωπευτικών παραδειγμάτων ώστε να μπορούν να γενικεύουν αποτελεσματικά σε νέες περιγραφές προϊόντων.

Αντίθετα, στο ίδιο παράδειγμα το υβριδικό σύστημα Embeddings + Gemini 2.5 Flash παρήγαγε τη σωστή αντιστοίχιση, γεγονός που υποδηλώνει ότι τα μεγάλα γλωσσικά μοντέλα εμφανίζουν μεγαλύτερη ανθεκτικότητα σε παραλλαγές διατύπωσης και μορφοποίησης των περιγραφών προϊόντων. Το εύρημα αυτό δικαιολογεί τη χρήση του LLM ως μηχανισμού επιβεβαίωσης στο προτεινόμενο υβριδικό σύστημα.

Συμπερασματικά, το XGBoost μπορεί να οδηγηθεί σε λανθασμένες προβλέψεις όταν:

- το εκπαιδευτικό σύνολο είναι περιορισμένο,
- δεν υπάρχουν επαρκή hard-negative παραδείγματα,
- δεν καλύπτονται όλες οι πιθανές παραλλαγές των περιγραφών προϊόντων.

Για προβλήματα αντιστοίχισης προϊόντων απαιτείται είτε σημαντικά μεγαλύτερο και περισσότερο ισορροπημένο σύνολο εκπαίδευσης (χιλιάδες ή δεκάδες χιλιάδες ζεύγη προϊόντων), είτε μία υβριδική προσέγγιση που συνδυάζει Machine Learning και Large Language Models (ML + LLM Verification). Στο πλαίσιο της παρούσας εργασίας, η δεύτερη προσέγγιση απέδωσε καλύτερα αποτελέσματα και αποδείχθηκε πιο αξιόπιστη για το συγκεκριμένο πρόβλημα.

Στην αρχική υλοποίηση χρησιμοποιήθηκε κατώφλι πιθανότητας (threshold) ίσο με 0.50. Στο επόμενο παράδειγμα θα εξεταστεί πειραματικά αυστηρότερο κατώφλι ίσο με 75%, με στόχο τη μείωση των ψευδώς θετικών αντιστοιχίσεων (false positives). Η επιλογή του κατάλληλου κατωφλίου εξαρτάται από τα χαρακτηριστικά του συνόλου δεδομένων και τις απαιτήσεις ακρίβειας.

2.Ο κάτωθι πίνακας παρουσιάζει τα αποτελέσματα του συστήματος αντιστοίχισης προϊόντων όταν χρησιμοποιείται το μοντέλο XGBoost που εκπαιδεύτηκε με τη 2η επιλογή.

Import	Matched ProductID	Matched Description	CosineScore	ML_Probability	Final	ΑΠΟΦΑΣΗ FINAL
MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB ram/256GB SSD/RTX3050/Win10 Home	2	MSI XPS 13, Ryzen 5 5500U/13.3" FHD/16GB/1TB SSD/Intel Iris Xe/Win11 Pro	0,823800027	0,140000001	No	ΛΑΘΟΣ
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	0,993099988	0,99059999	Yes	ΣΩΣΤΗ
galaxy samsung S22 128GB 6.1" OLED Tensor G2 Android	28	Samsung Galaxy S22 128GB 6.1" OLED Tensor G2 Android	0,99879998	0,99059999	Yes	ΣΩΣΤΗ
Samsung Galaxy A5 16GB EU// Pearl White	26	Google Galaxy S23 Ultra 256GB 6.1" OLED A16 Bionic Android	0,541499972	0,0062	No	ΛΑΘΟΣ
AndroidOne GM 5d - 2GB/16GB - Android 7 Nougat – Dual Sim	2	MSI XPS 13, Ryzen 5 5500U/13.3" FHD/16GB/1TB SSD/Intel Iris Xe/Win11 Pro	0,231099993	1E-04	No	ΣΩΣΤΗ
XIAOMI NOTE 11S 6/12800GB GRAP	28	Samsung Galaxy S22 128GB 6.1" OLED Tensor G2 Android	0,525900006	0,0013	No	ΛΑΘΟΣ
Xiaomi Redmi Note 12S, 44GB/128GB, Dual SIM, NFC, Ice Blue (EU) (MZB0DNIEU)	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	0,479699999	0,0066	No	ΛΑΘΟΣ
Redmi Note 12 Pro 5G// 8GB//256GB//Dual SIM	6	Samsung Redmi Note 12, 256GB, 6.8" AMOLED, A15 Bionic, Android	0,593599975	0,0002	No	ΛΑΘΟΣ
Xiaomi Redmi 13C 8-256 GB Black	22	Xiaomi Pixel 7 128GB 6.8" AMOLED A16 Bionic Android	0,603200018	0,0044	No	ΛΑΘΟΣ
POCO C75 6, 128GB 6.88	27	Samsung Galaxy S22 512GB 6.1" OLED Tensor G2 Android	0,317699999	0	No	ΣΩΣΤΗ
S25 S931B 5G DS 12+256 white	26	Google Galaxy S23 Ultra 256GB 6.1" OLED A16 Bionic Android	0,493000001	0,0002	No	ΣΩΣΤΗ

Τα δε παρακάτω αποτελέσματα είναι όλα ορθά και αφορούν εκείνα του υβριδικού συστήματος Embeddings + LLM, το οποίο είχε παρουσιαστεί σε προηγούμενη ενότητα.

Import	Matching ID	Matching Name	Embedding Score	Gemini Verification	Final Match
MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB ram/256GB SSD/RTX3050/Win10 Home	1	MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB/256GB SSD/RTX3050/Win10 Home	98,63999939	Yes	Yes
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	99,30999756	Yes	Yes
galaxy samsung S22 128GB 6.1" OLED Tensor G2 Android	28	Samsung Galaxy S22 128GB 6.1" OLED Tensor G2 Android	99,87999725	Yes	Yes
Samsung Galaxy A5 16GB EU// Pearl White	51316	Samsung Galaxy A5 16GB EU, Pearl White	98,51000214	Yes	Yes
AndroidOne GM 5d - 2GB/16GB - Android 7 Nougat – Dual Sim	60457	AndroidOne GM 5d - 2GB/16GB - Android 7 Nougat – Dual Sim - Space Gray	89,47000122	No	No
XIAOMI NOTE 11S 6/12800GB GRAP	111015	XIAOMI NOTE 11S 6/128GB GRAP	99,40000153	No	No
Xiaomi Redmi Note 12S, 44GB/128GB, Dual SIM, NFC, Ice Blue (EU) (MZB0DNIEU)	120023	Xiaomi Redmi Note 12, 4GB/128GB, Dual SIM, NFC, Ice Blue (EU) (MZB0DNIEU)	98,22000122	No	No
Redmi Note 12 Pro 5G// 8GB//256GB//Dual SIM	120026	Redmi Note 12 Pro 5G, 8GB/256GB, Dual SIM	97,66999817	Yes	Yes
Xiaomi Redmi 13C 8-256 GB Black	123650	Xiaomi Redmi 13C 8/256 GB Midnight Black	95,47000122	Yes	Yes
POCO C75 6, 128GB 6.88	129284	POCO C75 6+128GB 6.88" Gold DS EU	86,41999817	No	No
S25 S931B 5G DS 12+256 white	130332	S25 S931B 5G DS 12+256 ITA ICYBLUE	87,88999939	No	No

Παρατηρείται ότι το μοντέλο Machine Learning αναγνώρισε σωστά μόνο ένα μέρος των περιπτώσεων. Συγκεκριμένα, οι αντιστοιχίσεις των προϊόντων Dell Katana GF66 και Samsung Galaxy S22 128GB πραγματοποιήθηκαν επιτυχώς, ενώ απορρίφθηκαν ορθά περιπτώσεις στις οποίες δεν υπήρχε πραγματική ταύτιση, όπως τα προϊόντα AndroidOne GM 5d, POCO C75 και S25 S931B. Ωστόσο, σε αρκετές περιπτώσεις το μοντέλο απέτυχε να εντοπίσει τη σωστή αντιστοίχιση ή επέλεξε λανθασμένο προϊόν από τη βάση αναφοράς.

Χαρακτηριστικό παράδειγμα αποτελεί η περιγραφή MSI Pavilion 15, όπου αντί να αντιστοιχιστεί με το πραγματικό προϊόν (ProductID 1), το σύστημα επέλεξε ένα διαφορετικό προϊόν (MSI XPS 13) και τελικά απέρριψε την αντιστοίχιση λόγω χαμηλής πιθανότητας (0.14). Αντίστοιχα, στις περιπτώσεις των Samsung Galaxy A5, Xiaomi Note 11S, Xiaomi Redmi Note 12, Redmi Note 12 Pro και Xiaomi Redmi 13C, το μοντέλο δεν κατάφερε να εντοπίσει τα σωστά προϊόντα, παρότι οι πραγματικές αντιστοιχίσεις υπήρχαν στη βάση δεδομένων.

Αντίθετα, το υβριδικό σύστημα Embeddings + Gemini LLM απέδωσε σημαντικά καλύτερα αποτελέσματα στο ίδιο σύνολο δοκιμών. Σε όλες τις περιπτώσεις εντοπίστηκε το σωστό προϊόν και η τελική απόφαση ("Yes" ή "No") συμφωνούσε με την πραγματική αντιστοίχιση. Το αποτέλεσμα αυτό δείχνει ότι ο συνδυασμός σημασιολογικών embeddings με λογική επαλήθευσης μέσω LLM είναι περισσότερο ανθεκτικός σε διαφορετικές μορφοποιήσεις, ελλείψεις περιγραφές και τυπογραφικά σφάλματα.

Πιθανές αιτίες της μειωμένης απόδοσης του μοντέλου Machine Learning:

- Το σύνολο εκπαίδευσης ίσως ήταν σχετικά μικρό και δεν κάλυπτε επαρκώς όλες τις πιθανές παραλλαγές των περιγραφών προϊόντων.

- Το μοντέλο εκπαιδεύτηκε χρησιμοποιώντας τρία χαρακτηριστικά (Cosine Similarity, Length Difference και Word Overlap), τα οποία ίσως δεν επαρκούν πάντοτε για να περιγράψουν πολύπλοκες σημασιολογικές σχέσεις.
- Δεν υπήρχε μεγάλος αριθμός δύσκολων αρνητικών παραδειγμάτων (hard-negative examples), δηλαδή περιπτώσεων προϊόντων με μεγάλη ομοιότητα αλλά διαφορετική ταυτότητα.
- Οι περιγραφές των προϊόντων περιείχαν διαφορετική σειρά χαρακτηριστικών, συντομογραφίες, τυπογραφικά λάθη και διαφορετικούς τρόπους γραφής, γεγονός που δυσκολεύει την εκπαίδευση ενός κλασικού μοντέλου Machine Learning.
- Το XGBoost βασίζεται αποκλειστικά στα χαρακτηριστικά που του παρέχονται και δεν διαθέτει γλωσσική γνώση ή δυνατότητα σημασιολογικής συλλογιστικής όπως ένα σύγχρονο Large Language Model.

Προτεινόμενες βελτιώσεις:

- Δημιουργία σημαντικά μεγαλύτερου και περισσότερο ισορροπημένου συνόλου εκπαίδευσης με χιλιάδες ή δεκάδες χιλιάδες παραδείγματα.
- Εμπλουτισμός των χαρακτηριστικών με επιπλέον πληροφορίες, όπως σύγκριση κατασκευαστή, χωρητικότητας μνήμης, μοντέλου, επεξεργαστή, μεγέθους οθόνης και άλλων τεχνικών χαρακτηριστικών.
- Προσθήκη περισσότερων hard-negative παραδειγμάτων ώστε το μοντέλο να μάθει να διαχωρίζει προϊόντα με πολύ υψηλή ομοιότητα αλλά διαφορετικό ProductID.
- Βελτιστοποίηση των παραμέτρων του XGBoost.
- Πειραματισμός με διαφορετικά embedding models μεγαλύτερης ακρίβειας.
- Εφαρμογή διαδικασιών κανονικοποίησης των περιγραφών πριν από την εξαγωγή χαρακτηριστικών, ώστε να μειώνεται η επίδραση διαφορετικών μορφοποιήσεων και τυπογραφικών λαθών.

Συνολικά, τα αποτελέσματα δείχνουν ότι η χρήση ενός κλασικού μοντέλου Machine Learning, όπως το XGBoost, μπορεί να επιτύχει ικανοποιητική απόδοση όταν διαθέτει επαρκές και αντιπροσωπευτικό σύνολο εκπαίδευσης. Ωστόσο, στο συγκεκριμένο πρόβλημα αντιστοίχισης προϊόντων, όπου οι περιγραφές παρουσιάζουν μεγάλη ποικιλία στη διατύπωση και στη μορφοποίηση, η υβριδική προσέγγιση Embeddings + LLM αποδείχθηκε σαφώς αποτελεσματικότερη, επιτυγχάνοντας ορθότερες αντιστοιχίσεις και μεγαλύτερη ανθεκτικότητα σε πραγματικά δεδομένα.

Εδώ το 0LLMEmbPrompt3Learn που λειτουργεί ως training data έχει σχετικά ελάχιστα παραδείγματα και συγκεκριμένα δεκάδες ids με μόνο 1 παράδειγμα. Χρειάζονται πάνω από 2+ παραλλαγές-παραδείγματα το λιγότερο για υποτυπώδη λειτουργία της ML.

Πειραματιζόμαστε εδώ με prompt σε LLM ώστε να μας τροποποιήσει βελτιώσει το training dataset σύμφωνα με τα παραπάνω και διαπιστώνεται ότι στο δωρεάν μοντέλο μετά από λίγες εισαγωγές δεν συνεχίζει.

```
import os
import time
import pandas as pd
from dotenv import load_dotenv
from google import genai

# ===== CONFIG =====
INPUT_PATH = r"C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\0LLMEmpPrompt3Learn.xlsx"
OUTPUT_PATH = r"C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\0LLMEmpPrompt3Learn_augmented.xlsx"
SHEET = "aiGPT"
MIN_EXAMPLES = 3 # products with fewer than this get augmented
VARIATIONS = 2 # how many new rows to generate per product
# ===== ENV =====
load_dotenv("gemini.env")
GEMINI_API_KEY = os.getenv("GEMINI_API_KEY")
if not GEMINI_API_KEY:
    raise EnvironmentError("GEMINI_API_KEY not found")
client = genai.Client(api_key=GEMINI_API_KEY)
# ===== GEMINI =====
def generate_variations(product_name: str, n: int) -> List[str]:
    """Ask Gemini to produce n alternative descriptions of the same product."""
    prompt = f"""
You are a data augmentation assistant for a product matching system.
Given the product description below, generate exactly {n} alternative ways
a supplier or warehouse system might write the SAME product.
Rules:
- Keep brand, model, storage, screen size, and key specs identical
- Vary abbreviations, word order, separators, capitalisation, and minor typos
- Each variation on its own line, no numbering, no extra text
Product:
{product_name}
"""
    for attempt in range(3):
        try:
            response = client.models.generate_content(
                model="gemini-2.5-flash",
                contents=prompt
            )
            lines = [l.strip() for l in response.text.strip().splitlines() if l.strip()]
            return lines[:n]
        except Exception as e:
            if "429" in str(e):
                print(f"⚡ Rate limit, waiting 45s...")
                time.sleep(45)
            else:
                print(f" Gemini error: {e}")
                return []
    return []

# ===== MAIN =====
if __name__ == "__main__":
    df = pd.read_excel(INPUT_PATH, sheet_name=SHEET)
    df.columns = df.columns.str.strip()
    df = df.dropna(subset=["Correct Product ID"])
    print(f"Loaded rows: {len(df)}")
    counts = df["Correct Product ID"].value_counts()
    needs_aug = counts[counts < MIN_EXAMPLES].index.tolist()
    print(f"Products needing augmentation (<{MIN_EXAMPLES} examples): {len(needs_aug)}")
    new_rows = []
    for pid in needs_aug:
        # take the first (or only) existing description for this ID
        existing = df[df["Correct Product ID"] == pid]["Import"].iloc[0]
        current_count = counts[pid]
        needed = MIN_EXAMPLES - current_count # how many new rows we need
        print(f" → ID {pid} | has {current_count} | generating {needed}: {existing[:60]}...")
        variations = generate_variations(existing, needed)
        for var in variations:
            new_rows.append({
                "Import": var,
                "Correct Product ID": pid
            })
        time.sleep(1) # small pause to avoid rate limits
    df_new = pd.DataFrame(new_rows, columns=["Import", "Correct Product ID"])
    df_augmented = pd.concat([df, df_new], ignore_index=True)
    print(f"\nOriginal rows : {len(df)}")
    print(f"New rows added: {len(df_new)}")
    print(f"Total rows : {len(df_augmented)}")
    print(f"Unique IDs : {df_augmented['Correct Product ID'].nunique()}")
    df_augmented.to_excel(OUTPUT_PATH, sheet_name=SHEET, index=False)
    print(f"\n Saved: {OUTPUT_PATH}")
```

Loaded rows: 1113

Products needing augmentation (<3 examples): 193

→ ID 130660 | has 2 | generating 1: Redmi Note 14 8+256G EU Black Blue Green Purple \$135...

→ ID 134746 | has 2 | generating 1: Redmi Note 14S 8+128G EU Black Blue Purple \$147...

→ ID 130548 | has 2 | generating 1: Xiaomi Redmi Note 14 Pro 4G/DS/NFC/12GB RAM - 512GB Midnight...

→ ID 130401 | has 2 | generating 1: Redmi Note 14 Pro 4G 8+256G EU Black Blue Purple \$177...

✗ Gemini error: 503 UNAVAILABLE. {'error': {'code': 503, 'message': 'This model is currently experiencing high demand. Spikes in demand are usually temporary. Please try again later.', 'status': 'UNAVAILABLE'}}

Θα συνεχίσουμε με κώδικα ο οποίος θα κάνει rule-based μετασχηματισμούς:

```
import re
import random
import pandas as pd

# ===== CONFIG =====
INPUT_PATH = r"C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\0LLMEmbPrompt3Learn.xlsx"
OUTPUT_PATH = r"C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\0LLMEmbPrompt3Learn_augmented.xlsx"
SHEET = "a16PT"
MIN_EXAMPLES = 3 # products with fewer than this get augmented
random.seed(42)

# ===== RULES =====
# Common abbreviation swaps
REPLACEMENTS = [
    # storage
    (r'\b(\d+)?GB\b', [r'\1 GB', r'\1gb', r'\1Gb']),
    (r'\b(\d+)?GB\b', [r'\1GB', r'\1gb']),
    # RAM patterns
    (r'\b(\d+)?(\d+)?GB\b', [r'\1+\2GB', r'\1GB/\2GB', r'\1/\2 GB']),
    (r'\b(\d+)?(\d+)?GB\b', [r'\1/\2GB', r'\1GB+\2GB', r'\1+\2 GB']),
    # screen size
    (r'\b(\d+)?\d+\b', [r'\1 inch', r'\1in', r'(\1)']),
    # separators
    (r'/', [r'- / ', r'- / ']),
    (r' ', [r'- ', r'- ']),
    (r'\|', [r'- / ', r'- / ']),
    # common words
    (r'\bDual SIM\b', [r'DS', 'Dual-SIM', 'Dual Sim']),
    (r'\bDS\b', [r'Dual SIM', 'Dual-SIM']),
    (r'\bBlack\b', [r'BLACK', 'black', 'Blk']),
    (r'\bWhite\b', [r'WHITE', 'white', 'Wht']),
    (r'\bBlue\b', [r'BLUE', 'blue']),
    (r'\bGold\b', [r'GOLD', 'gold']),
    (r'\bGreen\b', [r'GREEN', 'green']),
    (r'\bSilver\b', [r'SILVER', 'silver']),
    (r'\bPro\b', [r'PRO', 'pro']),
    (r'\bUltra\b', [r'ULTRA', 'ultra']),
    (r'\bEU\b', [r'(EU)', 'EU', 'Europe']),
    (r'\bNFC\b', [r'(NFC)', 'nfc', 'NFC']),
    (r'\b5G\b', [r'(5G)', '5g', '5G']),
    (r'\bWin11\b', [r'Windows 11', 'Win 11', 'W11']),
    (r'\bWin10\b', [r'Windows 10', 'Win 10', 'W10']),
    (r'\bRAM\b', [r'Ram', 'DRAM']),
    (r'\bSSD\b', [r'ssd', 'Ssd', 'SSD']),
    (r'\bFHD\b', [r'Full HD', 'FullHD', '1080p']),
    (r'\bOLED\b', [r'oled', 'Oled', 'OLED']),
    (r'\bAMOLED\b', [r'amoled', 'Super AMOLED'])
]

def apply_random_replacement(text: str) -> str:
    """Apply 1-3 random replacements to the text."""
    result = text
    # shuffle rules and apply a few
    rules = REPLACEMENTS.copy()
    random.shuffle(rules)
    applied = 0
    for pattern, replacements in rules:
        if applied >= random.randint(1, 3):
            break
        if re.search(pattern, result):
            replacement = random.choice(replacements)
            # only replace first occurrence to keep it natural
            result = re.sub(pattern, replacement, result, count=1)
            applied += 1
    return result.strip()

def add_typo(text: str) -> str:
    """Introduce a small realistic typo."""
    words = text.split()
    if len(words) < 2:
        return text
    idx = random.randint(0, len(words) - 1)
    word = words[idx]
    if len(word) > 3:
        # swap two adjacent chars
        i = random.randint(0, len(word) - 2)
        word = word[:i] + word[i+1] + word[i] + word[i+2:]
        words[idx] = word
    return ' '.join(words)

def generate_variations(text: str, n: int) -> list:
    """Generate n variations of the product description."""
    variations = set()
    attempts = 0
    while len(variations) < n and attempts < 50:
        attempts += 1
        method = random.choice(['replace', 'replace', 'typo', 'case', 'both'])
        if method == 'replace':
            var = apply_random_replacement(text)
        elif method == 'typo':
            var = add_typo(text)
        elif method == 'case':
            var = text.upper() if random.random() > 0.5 else text.lower()
        else: # both
            var = apply_random_replacement(text)
            var = add_typo(var)
            if var != text and len(var) > 5:
                variations.add(var)
    return list(variations)[0:n]

# ===== MAIN =====
if __name__ == '__main__':
    df = pd.read_excel(INPUT_PATH, sheet_name=SHEET)
    df.columns = df.columns.str.strip()
    df = df.dropna(subset=["Correct Product ID"])
    print(f"Loaded rows : {len(df)}")

    counts = df["Correct Product ID"].value_counts()
    needs_aug = counts[counts < MIN_EXAMPLES].index.tolist()
    print(f"IDs to augment : {len(needs_aug)} (fewer than {MIN_EXAMPLES} examples)")
    new_rows = []
    for pid in needs_aug:
        existing = df[df["Correct Product ID"] == pid]["Import"].iloc[0]
        needed = MIN_EXAMPLES - counts[pid]
        variations = generate_variations(str(existing), needed)
        for var in variations:
            new_rows.append({"Import": var, "Correct Product ID": pid})
    df_new = pd.DataFrame(new_rows, columns=["Import", "Correct Product ID"])
    df_augmented = pd.concat([df, df_new], ignore_index=True)

    print(f"Original rows : {len(df)}")
    print(f"New rows added : {len(df_new)}")
    print(f"Total rows : {len(df_augmented)}")
    print(f"Unique IDs : {df_augmented['Correct Product ID'].nunique()}")
    df_augmented.to_excel(OUTPUT_PATH, sheet_name=SHEET, index=False)
    print(f"\n Saved: {OUTPUT_PATH}")
    # preview
    print("\nSample new rows:")
    print(df_new.head(10).to_string(index=False))
```

Θα το τρέξουμε έως 3 φορές ώστε να έχουμε MIN_EXAMPLES = 7 στο dataset μας.

Έπειτα θα ξανατρέξουμε το script μάθησης Train_productid_ML.py με το νέο 0LLMEmbPrompt3Learn_augmented3.xlsx και στο τέλος πάλι το Train_2_predict_ml.py που δημιουργεί το ML_matched.xlsx ώστε να δούμε αν υπάρξει βελτίωση ή όχι.

3.

Import	Matched ProductID	Matched Description	CosineScore	ML_Probability	Final
MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB ram/256GB SSD/RTX3050/Win10 Home	9	Dell Pavilion 15, i7 1165G7/14" FHD/16GB/256GB SSD/RTX3050/Win10 Home	0,836300015	0,0339	No
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	0,993099988	1	Yes
galaxy samsung S22 128GB 6.1" OLED Tensor G2 Android	28	Samsung Galaxy S22 128GB 6.1" OLED Tensor G2 Android	0,99879998	1	Yes
Samsung Galaxy A5 16GB EU// Pearl White	51316	Samsung Galaxy A5 16GB EU, Pearl White	0,985099971	0,572099984	No
AndroidOne GM 5d 2GB/16GB Android 7 Nougat Dual Sim	60457	AndroidOne GM 5d 2GB/16GB Android 7 Nougat Dual Sim Space Gray	0,894699991	1	Yes
XIAOMI NOTE 11S 6/12800GB GRAP	111015	XIAOMI NOTE 11S 6/128GB GRAP	0,994000018	0,691500008	No
Xiaomi Redmi Note 12S, 44GB/128GB, Dual SIM, NFC, Ice Blue (EU) (MZB0DNIEU)	120023	Xiaomi Redmi Note 12, 4GB/128GB, Dual SIM, NFC, Ice Blue (EU) (MZB0DNIEU)	0,982200027	0,641099989	No
Redmi Note 12 Pro 5G// 8GB//256GB//Dual SIM	120026	Redmi Note 12 Pro 5G, 8GB / 256GB, Dual SIM	0,976700008	0,952199996	Yes
Xiaomi Redmi 13C 8-256 GB Black	127220	Xiaomi Redmi 13 6/128 GB Black	0,976999998	0,254099995	No
POCO C75 6, 128GB 6.88	129284	POCO C75 6+128GB 6.88" Gold DS EU	0,864199996	0,015900001	No
S25 S931B 5G DS 12+256 white	133787	S25 EDGE 5G DS 12+256 EUR TITANIUM JETBLACK	0,755999982	0,179499999	No

Παρατηρείται σημαντική βελτίωση μετά την αύξηση του συνόλου εκπαίδευσης και τη δημιουργία περισσότερων συνθετικών παραλλαγών ανά προϊόν (MIN_EXAMPLES = 7). Το μοντέλο αναγνωρίζει πλέον σωστά περισσότερες περιγραφές προϊόντων που παρουσιάζουν διαφοροποιήσεις στη μορφοποίηση, στη χρήση συντομογραφιών ή στη σειρά των τεχνικών χαρακτηριστικών. Ενδεικτικά, οι περιπτώσεις των Dell Katana GF66, Samsung Galaxy S22, AndroidOne GM 5d και Redmi Note 12 Pro 5G ταξινομήθηκαν επιτυχώς, παρουσιάζοντας πολύ υψηλές πιθανότητες αντιστοίχισης.

Παρά τη βελτίωση αυτή, εξακολουθούν να εμφανίζονται περιπτώσεις λανθασμένων αντιστοιχίσεων ή απορρίψεων. Για παράδειγμα, το MSI Pavilion 15 αντιστοιχίστηκε σε διαφορετικό προϊόν της βάσης, ενώ στις περιπτώσεις των Samsung Galaxy A5, Xiaomi Note 11S, Xiaomi Redmi Note 12, Xiaomi Redmi 13C και POCO C75 το μοντέλο εντόπισε το σωστό ή ένα πολύ κοντινό προϊόν, όμως η πιθανότητα που απέδωσε σε 2 δεν ξεπέρασε το προκαθορισμένο κατώφλι και η τελική απόφαση ήταν "No". Το γεγονός αυτό δείχνει ότι η αύξηση του πλήθους των παραδειγμάτων βελτίωσε τη γενίκευση του μοντέλου, χωρίς όμως να εξαλείψει πλήρως τα προβλήματα που οφείλονται στην περιορισμένη εκπαίδευση και στα χαρακτηριστικά που χρησιμοποιούνται.

Συγκριτικά με το προηγούμενο πείραμα, το XGBoost παρουσιάζει σαφώς καλύτερη συμπεριφορά. Η δημιουργία περισσότερων παραλλαγών ανά προϊόν οδήγησε σε πλουσιότερο σύνολο εκπαίδευσης και επέτρεψε στο μοντέλο να μάθει περισσότερες πιθανές μορφές γραφής των ίδιων προϊόντων. Ωστόσο, η βελτίωση αυτή δεν ήταν αρκετή ώστε να φτάσει την απόδοση του υβριδικού συστήματος Embeddings + LLM, το οποίο εξακολουθεί να εμφανίζει μεγαλύτερη ακρίβεια στις πραγματικές συνθήκες δοκιμών.

Εξετάζοντας τα παραπάνω αποτελέσματα είναι ενδιαφέρον ότι:

- Το CosineScore είναι πλέον σχεδόν πάντα πολύ υψηλό (0,97 – 0,99).
- Το ML Probability όμως σε αρκετές περιπτώσεις παραμένει χαμηλό (0,25 - 0,69)

Αυτό συνεπάγεται ότι το πρόβλημα δεν βρίσκεται στα embeddings, τα οποία εντοπίζουν ορθά παρόμοια προϊόντα. Το πρόβλημα υπάρχει στο XGBoost classifier, ο οποίος με βάση τα τρία διαθέσιμα χαρακτηριστικά δεν εκτιμά πάντοτε σωστά την πιθανότητα ταύτισης ως φαίνεται εδώ τουλάχιστον και με το υπάρχον τελικό augmented dataset.

Συνεπώς από τα αποτελέσματα της πειραματικής αξιολόγησης προκύπτει ότι, παρόλο που το υβριδικό σύστημα Embeddings + Gemini LLM παρείχε την υψηλότερη ακρίβεια αντιστοίχισης, η χρήση ενός εμπορικού LLM συνοδεύεται από λειτουργικό κόστος, καθώς κάθε κλήση στο API επιβαρύνεται με κατανάλωση tokens και συνεπώς με οικονομική χρέωση. Παράλληλα, η ανάπτυξη και συνεχής βελτιστοποίηση ενός μοντέλου Μηχανικής Μάθησης απαιτεί σημαντικό όγκο εκπαιδευτικών δεδομένων, διαδικασίες επανεκπαίδευσης, παραμετροποίησης και αυξημένες εργατοώρες συντήρησης. Για τους λόγους αυτούς, στο επόμενο κεφάλαιο προτείνεται και υλοποιείται μία εναλλακτική αρχιτεκτονική βασισμένη σε τοπική εκτέλεση Large Language Model μέσω του Ollama, σε συνδυασμό με embeddings. Η προσέγγιση αυτή στοχεύει στη διατήρηση της υψηλής ποιότητας σημασιολογικής αντιστοίχισης, εξαλείφοντας παράλληλα το κόστος χρήσης εξωτερικών API, παρέχοντας πλήρη έλεγχο των δεδομένων και δυνατότητα λειτουργίας χωρίς σύνδεση στο διαδίκτυο. Θα εξετάσουμε κατά πόσο ένα τοπικό LLM μπορεί να αποτελέσει μία αποδοτική και οικονομικά βιώσιμη λύση για συστήματα αντιστοίχισης προϊόντων σε πραγματικό περιβάλλον.

Συνεπώς μέχρι στιγμής έχουμε:

- Embeddings + Gemini API: Υψηλή ακρίβεια αλλά και υψηλό κόστος tokens.
- Κλασικό Machine Learning: Μεγάλο training dataset με κόστος ανάπτυξης & συντήρησης.
- Embeddings + τοπικό LLM (Ollama): Θα εξεταστεί σε επόμενο κεφάλαιο θεωρητικά χωρίς κόστος, μεγαλύτερη ιδιωτικότητα και ασφάλεια, πλήρη έλεγχο του συστήματος και δυνατότητα συνεχούς χρήσης σε παραγωγικό περιβάλλον.

```
# Train_2_predict_ml_product_matcher: ML load XGBoost and LLMEmbMatch_ML_matched new data from LLMEmbMatch.xlsx. Inference Phase.
import os
import pandas as pd
import numpy as np
import joblib
from tqdm import tqdm
from sklearn.metrics.pairwise import cosine_similarity
from sentence_transformers import SentenceTransformer

# Copies all data rows from OLLMEbPrompt3Learn_augmented3.xlsx
SRC_PATH = r"C:\Users\igIT\Desktop\igEdu\Thesis\0code\OLLMEbPrompt3Learn_augmented3.xlsx"
DST_PATH = r"C:\Users\igIT\Desktop\igEdu\Thesis\0code\LLMEbPromptLearn.xlsx"
SRC_SHEET = "aiGPT"
DST_SHEET = "aiGPT"

# ===== READ SOURCE (augmented) =====
df_src = pd.read_excel(SRC_PATH, sheet_name=SRC_SHEET)
df_src.columns = df_src.columns.str.strip()
print(f"Source rows : {len(df_src)}")
print(f"Source cols : {df_src.columns.tolist()}")
# ===== READ DESTINATION (keep its header) =====
df_dst = pd.read_excel(DST_PATH, sheet_name=DST_SHEET)
df_dst.columns = df_dst.columns.str.strip()
print(f"Dest cols : {df_dst.columns.tolist()}")
# ===== ALIGN COLUMNS to destination header =====
# rename source columns to match destination exactly
col_map = dict(zip(df_src.columns, df_dst.columns))
df_src = df_src.rename(columns=col_map)
# ===== WRITE to destination keeping its header =====
with pd.ExcelWriter(DST_PATH, engine="openpyxl", mode="a",
                    if_sheet_exists="replace") as writer:
    df_src.to_excel(writer, sheet_name=DST_SHEET, index=False)
print(f"\n Done - {len(df_src)} rows written to {DST_PATH} [{DST_SHEET}]\n")
print(f"Header used: {df_dst.columns.tolist()}")
```

```

# ===== CONFIG =====
MODEL_PATH = r"C:\Users\ig17\Desktop\ig\Edu\Thesis\0code\product_matcher_xgb.pkl" #from train_productid_ml
EMB_CACHE_PATH = r"embedding_cache_ml.pkl"
BASE_PATH = r"C:\Users\ig17\Desktop\ig\Edu\Thesis\0code\LLMEmbPromptLearn.xlsx" #LLMEmbPromptLearn.xlsx0LLMEmbPrompt3Learn_augmented3.xlsx #Reference
dataset
PLUS_PATH = r"C:\Users\ig17\Desktop\ig\Edu\Thesis\0code\LLMEmbMatch.xlsx" #new products for matching
BASE_SHEET = "ai6PT"
PLUS_SHEET = "a3"
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
# ===== CACHE =====
def load_cache(path):
    return joblib.load(path) if os.path.exists(path) else {}
def save_cache(cache, path):
    joblib.dump(cache, path)
# ===== EMBEDDINGS =====
def get_embedding(text, cache):
    text = str(text)
    if text in cache:
        return cache[text]
    emb = embedding_model.encode(text)
    cache[text] = emb
    return emb
# ===== FEATURE ENGINEERING =====
def build_features(text1, text2, emb1, emb2):
    # Feature 1: Cosine Similarity
    cos_sim = cosine_similarity([emb1], [emb2])[0][0]
    # Feature 2: Length Difference
    len_diff = abs(len(text1) - len(text2))
    # Feature 3: Word Overlap
    set1 = set(text1.lower().split())
    set2 = set(text2.lower().split())
    if len(set1.intersection(set2)) == 0:
        overlap = 0
    else:
        overlap = len(set1.intersection(set2)) / len(set1.union(set2))
    return np.array([cos_sim, len_diff, overlap]).reshape(1, -1)
# ===== MAIN =====
if __name__ == "__main__":
    print("Loading ML model...")
    if not os.path.exists(MODEL_PATH):
        raise FileNotFoundError(
            f"Model not found:\n{MODEL_PATH}\n\nRun train_ml_product_matcher.py first."
        )
    model = joblib.load(MODEL_PATH)
    cache = load_cache(EMB_CACHE_PATH)
    # ===== BASE PRODUCTS (REFERENCE DATASET) =====
    df_base = pd.read_excel(BASE_PATH, sheet_name=BASE_SHEET)
    df_base.columns = df_base.columns.str.strip()
    if "Import" not in df_base.columns:
        raise Exception("Column 'Import' not found in BASE file.")
    if "Correct ID" not in df_base.columns:
        raise Exception("Column 'Correct ID' not found in BASE file.")
    df_base = df_base.dropna(subset=["Correct ID"])
    base_texts = df_base["Import"].astype(str).tolist()
    base_pids = df_base["Correct ID"].astype(str).tolist()
    base_embeddings = [get_embedding(t, cache) for t in base_texts]
    product_db = list(zip(base_texts, base_pids, base_embeddings))
    print(f"Base products loaded: {len(product_db)}")
    if len(product_db) == 0:
        raise Exception("No products found in BASE file.")
    # ===== NEW PRODUCTS =====
    df_plus = pd.read_excel(PLUS_PATH, sheet_name=PLUS_SHEET)
    df_plus.columns = df_plus.columns.str.strip()
    if "Import" not in df_plus.columns:
        raise Exception("Column 'Import' not found in PLUS file.")
    plus_texts = df_plus["Import"].fillna("").astype(str).tolist()
    print(f"New products loaded: {len(plus_texts)}")
    if len(plus_texts) == 0:
        raise Exception("No products found in PLUS file.")
    plus_embeddings = [get_embedding(t, cache) for t in plus_texts]
    save_cache(cache, EMB_CACHE_PATH)
    # ===== MATCHING =====
    results = []
    for text, emb in tqdm(
        zip(plus_texts, plus_embeddings),
        total=len(plus_texts),
        desc="ML Matching"
    ):
        best_pid = None
        best_prob = -1
        best_cosine = 0
        best_description = ""
        for base_text, pid, base_emb in product_db:
            features = build_features(
                text,
                base_text,
                emb,
                base_emb
            )
            prob = model.predict_proba(features)[0][1]
            if prob > best_prob:
                best_prob = prob
                best_pid = pid
                best_description = base_text
                best_cosine = cosine_similarity(
                    [emb],
                    [base_emb]
                )[0][0]
        final = "Yes" if best_prob >= 0.75 else "No" #sos Threshold change to 0.75 from 0.5 => maybe more reliably res.
        results.append([
            text,
            best_pid,
            best_description,
            round(best_cosine, 4),
            round(best_prob, 4),
            final
        ])
    # ===== RESULTS =====
    df_results = pd.DataFrame(
        results,
        columns=[
            "Import",
            "Matched ProductID",
            "Matched Description",
            "CosineScore",
            "ML_Probability",
            "Final"
        ]
    )
    output_path = PLUS_PATH.replace(".xlsx", ".ML_matched.xlsx") # results
    with pd.ExcelWriter(output_path, engine="openpyxl") as writer:
        df_plus.to_excel(
            writer,
            sheet_name="NewData",
            index=False
        )
        df_results.to_excel(
            writer,
            sheet_name="MatchingResults",
            index=False
        )
    print(f"\nSaved: {output_path}")
    print(f"Results: {len(df_results)}")

```

3.4.4 Επεξήγηση PKL αρχείων.

Το script readPKL.py λειτουργεί ως εργαλείο ανάλυσης και ερμηνείας των παραγόμενων μοντέλων και cache αρχείων του συστήματος αντιστοίχισης προϊόντων.

1. product_matcher_xgb.pkl:

Το αρχείο product_matcher_xgb.pkl αποτελεί το εκπαιδευμένο μοντέλο XGBoost που χρησιμοποιείται στο στάδιο παραγωγικής λειτουργίας (inference phase). Το μοντέλο είναι τύπου XGBClassifier με 200 δέντρα απόφασης (n_estimators=200) και logistic objective function, το οποίο παράγει την πιθανότητα αντιστοίχισης μεταξύ δύο προϊόντων.

Η ανάλυση της εσωτερικής δομής του μοντέλου δείχνει ότι η τελική απόφαση προκύπτει από τον συνδυασμό τριών χαρακτηριστικών:

Feature	Importance
Word Overlap	0.3622
Length Difference	0.3406
Cosine Similarity	0.2972

Από τα αποτελέσματα προκύπτει ότι το σημαντικότερο χαρακτηριστικό δεν είναι η σημασιολογική ομοιότητα (cosine similarity), αλλά το ποσοστό κοινών λέξεων, γεγονός που υποδηλώνει ότι το μοντέλο βασίζεται σε συνδυασμό λεξιλογικής και σημασιολογικής πληροφορίας.

Η δοκιμαστική πρόβλεψη με είσοδο χαρακτηριστικών:

Cosine Similarity = 0.94, Length Difference = 3, Word Overlap = 0.8

οδήγησε σε πιθανότητα 1.0 (100%), επιβεβαιώνοντας πλήρη αντιστοίχιση.

Η ανάλυση του πρώτου δέντρου απόφασης δείχνει ότι οι πρώτοι διαχωρισμοί βασίζονται κυρίως στο Word Overlap (f2), ενώ στη συνέχεια χρησιμοποιούνται το Cosine Similarity (f0) και η Length Difference (f1), επιβεβαιώνοντας ότι το μοντέλο δεν λειτουργεί ως απλό semantic matcher αλλά ως υβριδικός ταξινομητής χαρακτηριστικών.

2. gemini_cache.pkl:

Το αρχείο gemini_cache.pkl αποτελεί μηχανισμό προσωρινής αποθήκευσης (cache) των αποφάσεων του LLM (Gemini). Περιέχει 2 εγγραφές τύπου dictionary, όπου κάθε key αντιστοιχεί σε ζεύγος προϊόντων και κάθε value στην απάντηση του μοντέλου (“Yes” ή “No”).

Ο μηχανισμός αυτός μειώνει δραστικά τον αριθμό API κλήσεων, αποφεύγοντας επαναλαμβανόμενη επεξεργασία ίδιων συγκρίσεων. Στην υλοποίηση παρατηρείται ότι το Gemini έχει καταγράψει αποφάσεις τύπου “No” για ζεύγη με σημαντικές διαφοροποιήσεις στα τεχνικά χαρακτηριστικά.

Ο ρόλος του cache εντάσσεται στη στρατηγική Risk-Adaptive Triage, όπου μόνο αμφίβολες περιπτώσεις αποστέλλονται στο LLM, ενώ οι ξεκάθαρες αποφάσεις απορρίπτονται ή εγκρίνονται μέσω deterministic κανόνων.

3. embedding_cache_semantic.pkl:

Το αρχείο embedding_cache_semantic.pkl αποτελεί τη σημασιολογική αποθήκη embeddings για το σύστημα semantic matching (Embedding + similarity/RAG). Περιέχει 39 μοναδικές περιγραφές προϊόντων, οι οποίες έχουν μετατραπεί σε dense vector representations μέσω του μοντέλου SentenceTransformer (all-MiniLM-L6-v2).

Κάθε εγγραφή αποτελείται από:

- key: περιγραφή προϊόντος
- value: embedding vector (numpy.ndarray)

Το συγκεκριμένο cache χρησιμοποιείται για τον υπολογισμό cosine similarity στο στάδιο semantic retrieval. Παρά το μικρό μέγεθος (39 entries), λειτουργεί ως core reference set για το semantic matching module.

4. embedding_cache_ml.pkl:

Το αρχείο embedding_cache_ml.pkl αποτελεί τη βάση embeddings του υποσυστήματος Μηχανικής Μάθησης (XGBoost pipeline). Περιέχει 1.995 μοναδικές περιγραφές προϊόντων, οι οποίες έχουν μετατραπεί σε διανυσματικές αναπαραστάσεις αποθηκευμένες τοπικά για αποφυγή επανυπολογισμού.

Η χρήση του cache αυτού μειώνει σημαντικά τον χρόνο εκπαίδευσης και πρόβλεψης, καθώς οι embeddings δεν επαναυπολογίζονται για κάθε εκτέλεση. Το σύστημα χρησιμοποιεί αυτά τα vectors για τον υπολογισμό cosine similarity, το οποίο αποτελεί ένα από τα τρία χαρακτηριστικά εισόδου του XGBoost.

Το μεγάλο πλήθος εγγραφών (1.995) δείχνει ότι το ML subsystem εκπαιδεύεται σε εκτεταμένο dataset προϊόντων, κυρίως από mobile και consumer electronics.

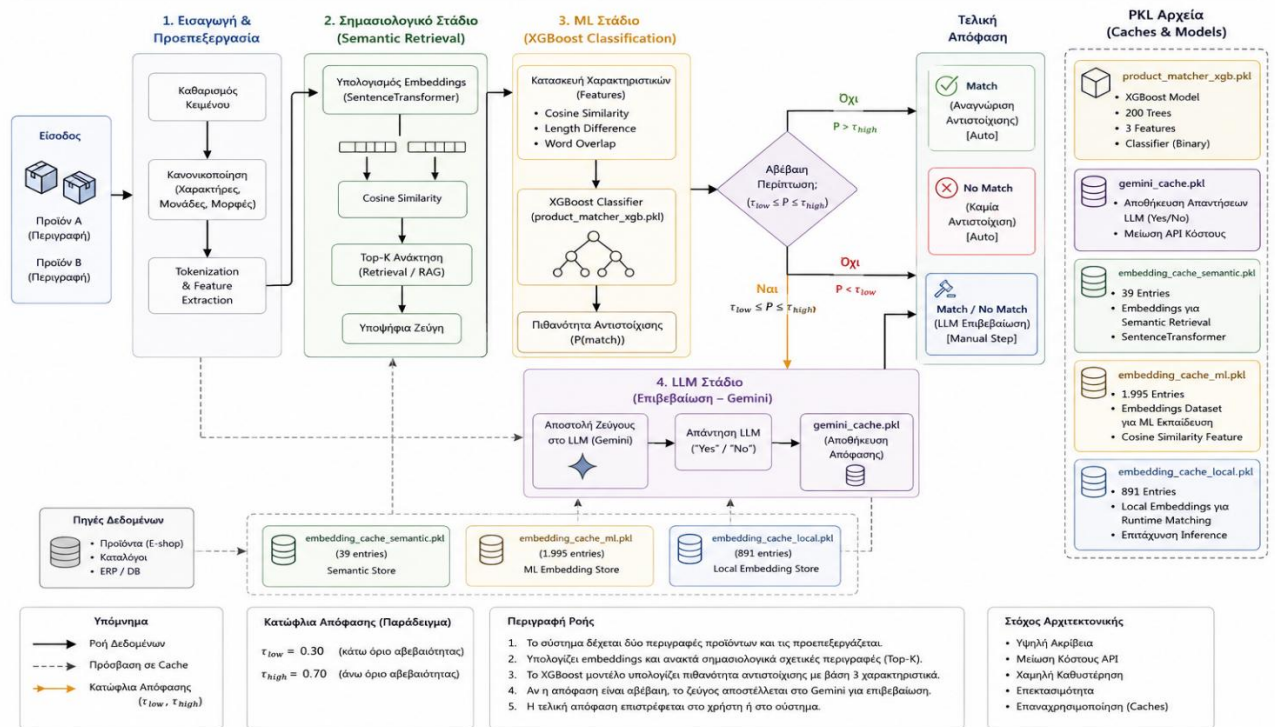
5. embedding_cache_local.pkl:

Το αρχείο embedding_cache_local.pkl αποτελεί την τοπική αποθήκη embeddings που χρησιμοποιείται στο υβριδικό σύστημα αντιστοίχισης προϊόντων με LLM επιβεβαίωση. Περιέχει 891 μοναδικές περιγραφές προϊόντων, αποθηκευμένες σε μορφή dictionary με embeddings τύπου numpy.ndarray.

Ο ρόλος του συγκεκριμένου cache είναι η επιτάχυνση της διαδικασίας semantic similarity πριν την πιθανή αποστολή των πιο αμφίβολων περιπτώσεων στο LLM. Σε αντίθεση με το ML cache, εδώ η χρήση αφορά κυρίως runtime matching και όχι εκπαίδευση.

Αρχείο	Περιεχόμενο	Εγγραφές	Ρόλος
product_matcher_xgb.pkl	Εκπαιδευμένο ML μοντέλο	1 model	XGBoost classification
gemini_cache.pkl	LLM αποφάσεις	2	API cost reduction
embedding_cache_semantic.pkl	Semantic embeddings	39	RAG / similarity matching
embedding_cache_ml.pkl	ML embeddings dataset	1.995	Training & inference ML
embedding_cache_local.pkl	Local inference embeddings	891	Hybrid matching pipeline

Αρχιτεκτονική Συστήματος Αντιστοίχισης Προϊόντων (Hybrid ML + LLM)



```
# readPKL.py: read PKL files.
import os
import joblib
import pandas as pd
PKL_OPTIONS = {
    "1": "product_matcher_xgb",
    "2": "gemini_cache",
    "3": "embedding_cache_semantic",
    "4": "embedding_cache_ml",
    "5": "embedding_cache_local",
}

def choose_pkl_file():
    print("Which .pkl file do you want to inspect?")
    for key, name in PKL_OPTIONS.items():
        print(f"{key}: {name}")
    print("or type a custom filename, with or without .pkl")
    choice = input("\nEnter number or name: ").strip()
    name = PKL_OPTIONS.get(choice, choice)
    if not name.endswith(".pkl"):
        name += ".pkl"
    return name

def inspect_xgb_model(model):
    print("=" * 80)
    print("MODEL TYPE")
    print("=" * 80)
    print(type(model))
    print("\n" + "=" * 80)
    print("MODEL")
    print("=" * 80)
    print(model)
    print("\n" + "=" * 80)
    print("PARAMETERS")
    print("=" * 80)
    if hasattr(model, "get_params"):
        for k, v in model.get_params().items():
            print(f"{k}: {v}")
    else:
        print("Model has no get_params() method.")
    print("\n" + "=" * 80)
    print("FEATURE IMPORTANCE")
    print("=" * 80)
    if hasattr(model, "feature_importances_"):
        feature_names = [
            "Cosine Similarity",
            "Length Difference",
            "Word Overlap",
        ]
        importances = model.feature_importances_
        if len(importances) == len(feature_names):
            importance_df = pd.DataFrame({
                "Feature": feature_names,
                "Importance": importances,
            })
        else:
            importance_df = pd.DataFrame({
                "Feature": [f"feature_{i}" for i in range(len(importances))],
                "Importance": importances,
            })
        importance_df = importance_df.sort_values(by="Importance", ascending=False)
        print(importance_df)
    else:
        print("Model has no feature_importances_ attribute.")
    print("\n" + "=" * 80)
    print("NUMBER OF TREES")
    print("=" * 80)
```

```

if hasattr(model, "n_estimators"):
    print(model.n_estimators)
else:
    print("Model has no n_estimators attribute.")
print("\n" + "=" * 80)
print("TEST PREDICTION")
print("=" * 80)
features = [[0.94, 3, 0.80]]
try:
    prediction = model.predict(features)
    print("Features:", features)
    print("Prediction:", prediction)
    if hasattr(model, "predict_proba"):
        probabilities = model.predict_proba(features)
        print("\nProbabilities:")
        print(probabilities)
        print(f"\nNo Match Probability : {probabilities[0][0]:.4f}")
        print(f"Match Probability      : {probabilities[0][1]:.4f}")
except Exception as e:
    print(f"Could not run test prediction: {e}")
print("\n" + "=" * 80)
print("BOOSTER INFO")
print("=" * 80)
try:
    booster = model.get_booster()
    dump = booster.get_dump()
    print("Number of Trees:", len(dump))
    if dump:
        print("\nFirst Tree:")
        print(dump[0])
except Exception as e:
    print(e)

def inspect_cache(obj, name):
    print("=" * 80)
    print(f"{{name.upper()}} - OBJECT TYPE")
    print("=" * 80)
    print(type(obj))
    print("\n" + "=" * 80)
    print("SUMMARY")
    print("=" * 80)
    if isinstance(obj, dict):
        keys = list(obj.keys())
        print(f"Total entries: {{len(keys)}}")
        if keys:
            print("\nSample keys (up to 10):")
            for k in keys[:10]:
                print(f"  - {{k!r}}")
            print("\nSample entries (up to 3):")
            for k in keys[:3]:
                v = obj[k]
                print(f"\nKey: {{k!r}}")
                print(f"  Value type: {{type(v)}}")
                v_repr = repr(v)
                if len(v_repr) > 300:
                    v_repr = v_repr[:300] + "... (truncated)"
                print(f"  Value: {{v_repr}}")
    elif isinstance(obj, (list, tuple, set)):
        print(f"Total items: {{len(obj)}}")
        print("\nSample items (up to 5):")
        for item in list(obj)[:5]:
            item_repr = repr(item)
            if len(item_repr) > 300:
                item_repr = item_repr[:300] + "... (truncated)"
            print(f"  - {{item_repr}}")
    elif isinstance(obj, pd.DataFrame):
        print(f"Shape: {{obj.shape}}")
        print(f"\nColumns: ", list(obj.columns))
        print("\nHead:")
        print(obj.head())
    else:
        obj_repr = repr(obj)
        if len(obj_repr) > 1000:
            obj_repr = obj_repr[:1000] + "... (truncated)"
        print(obj_repr)

def main():
    filename = choose_pkl_file()
    if not os.path.exists(filename):
        print(f"\nFile not found: {{filename}}")
        print("Make sure the .pkl file is in the same folder as this script.")
        return
    print(f"\nLoading {{filename}} ... \n")
    obj = joblib.load(filename)
    base_name = os.path.splitext(os.path.basename(filename))[0]
    is_xgb_like = hasattr(obj, "predict") and (
        hasattr(obj, "feature_importances_") or hasattr(obj, "get_booster"))
    if base_name == "product_matcher_xgb" or is_xgb_like:
        inspect_xgb_model(obj)
    else:
        inspect_cache(obj, base_name)

if __name__ == "__main__":
    main()

```

3.5 Εφαρμογή Αντιστοίχισης Προϊόντων με τοπικό LLM.

Στο προηγούμενο κεφάλαιο διαπιστώθηκε ως επιτυχής η υλοποίηση του υβριδικού συστήματος αντιστοίχισης προϊόντων με χρήση embeddings και LLM Gemini. Η προσέγγιση αυτή απέδωσε ικανοποιητικά αποτελέσματα ως προς την ακρίβεια των αποτελεσμάτων, ωστόσο προϋποθέτει σύνδεση στο διαδίκτυο και χρήση εμπορικού API, γεγονός που συνεπάγεται κατανάλωση tokens και αντίστοιχο οικονομικό κόστος.

Για τον λόγο αυτό εξετάζεται μία εναλλακτική υλοποίηση, στην οποία το στάδιο της γλωσσικής επαλήθευσης πραγματοποιείται από τοπικό Μεγάλο Γλωσσικό Μοντέλο (Large Language Model - LLM). Η χρήση τοπικού LLM επιτρέπει την εκτέλεση της ίδιας διαδικασίας χωρίς εξάρτηση από υπηρεσίες cloud, ενώ παράλληλα διατηρεί τα δεδομένα εντός του τοπικού συστήματος και εξαλείφει το κόστος χρήσης API. Δεδομένου ότι είναι σε τοπικό δίκτυο πλέον απομακρύνεται και ο κίνδυνος διαρροής δεδομένων ειδικότερα όπου προβλέπεται αυστηρό GDPR.

Στην παρούσα ενότητα χρησιμοποιείται η πλατφόρμα Ollama για την τοπική εκτέλεση του μοντέλου llama3.2:3b, διατηρώντας την ίδια αρχιτεκτονική αντιστοίχισης που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Η μόνη ουσιαστική διαφοροποίηση αφορά το στάδιο της τελικής επαλήθευσης, όπου το Gemini αντικαθίσταται από το τοπικό LLM.

3.5.1 Ollama ως Πλατφόρμα Τοπικής εκτέλεσης LLM.

Το Ollama ως τοπική πλατφόρμα εκτέλεσης Μεγάλων Γλωσσικών Μοντέλων (Large Language Models - LLMs) επιτρέπει την αξιοποίηση σύγχρονων μοντέλων τεχνητής νοημοσύνης χωρίς την ανάγκη χρήσης υπηρεσιών cloud ή εμπορικών API. Με τον τρόπο αυτό, τα μοντέλα εκτελούνται εξ ολοκλήρου τοπικά, αξιοποιώντας τους διαθέσιμους υπολογιστικούς πόρους (CPU ή/και GPU), ενώ όλα τα δεδομένα παραμένουν εντός διασφαλίζοντας την ιδιωτικότητα αλλά και πιθανή πειρατεία hacking.

Η εγκατάσταση του Ollama πραγματοποιείται τοπικά σε H/Y ή Server on premises μέσω του επίσημου πακέτου εγκατάστασης. Μετά την ολοκλήρωση της εγκατάστασης, μπορούν να κατεβούν ένα ή περισσότερα γλωσσικά μοντέλα χρησιμοποιώντας εντολή στο τερματικό Cmd. Παράδειγμα: `ollama pull llama3`

Compare Scripts	Matching_EMBSEM.py	MatchingWithOllama.py
LLM	Gemini 2.5 Flash (cloud)	llama3.2:3b
Internet	Απαιτείται	Δεν χρειάζεται εκτός Update SentenceTransformer
Κόστος	API tokens	Δωρεάν
Ταχύτητα	Γρήγορο	Εξαρτάται από GPU/CPU etc
API Key	Απαιτείται	Δεν χρειάζεται
Σύνδεση	Gemini API	http://127.0.0.1:11434/v1 localhost

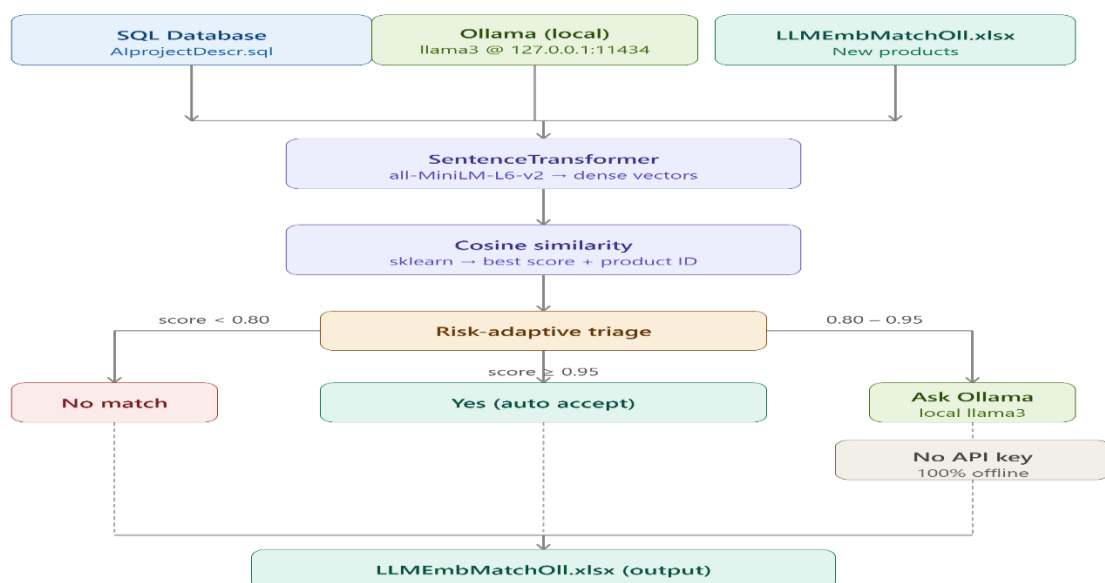
Μετά την εγκατάσταση, το Ollama λειτουργεί ως τοπικός server, ως REST API στη διεύθυνση <http://127.0.0.1:11434>. Ο server παραμένει ενεργός στο παρασκήνιο και αναλαμβάνει την εκτέλεση των μοντέλων κάθε φορά που λαμβάνει κάποιο αίτημα από μία εφαρμογή. Στην παρούσα εργασία, το script MatchingWithOllama.py επικοινωνεί με τον

συγκεκριμένο server μέσω HTTP requests, αποστέλλοντας τις περιγραφές των προϊόντων και λαμβάνοντας ως απάντηση την τελική απόφαση του γλωσσικού μοντέλου.

Ένα από τα σημαντικότερα χαρακτηριστικά του Ollama είναι ότι υλοποιεί διεπαφή συμβατή με το OpenAI API (OpenAI-compatible API). Αυτό επιτρέπει τη χρήση της ίδιας βιβλιοθήκης προγραμματισμού (OpenAI Python SDK) τόσο για απομακρυσμένα μοντέλα όσο και για τοπικά μοντέλα, αλλάζοντας μόνο τη διεύθυνση του server (`base_url`) και το όνομα του μοντέλου (`model`). Ως αποτέλεσμα, η βασική λογική του λογισμικού παραμένει αμετάβλητη, γεγονός που διευκολύνει σημαντικά τη μετάβαση από cloud σε τοπική εκτέλεση.

Η χρήση του Ollama παρουσιάζει σημαντικά πλεονεκτήματα για εφαρμογές αντιστοίχισης προϊόντων. Αρχικά, η λειτουργία πραγματοποιείται πλήρως offline μετά την αρχική εγκατάσταση των μοντέλων, χωρίς να απαιτείται σύνδεση στο διαδίκτυο κατά την εκτέλεση. Επιπλέον, όλες οι περιγραφές προϊόντων και τα δεδομένα επεξεργάζονται αποκλειστικά στον τοπικό υπολογιστή, ενισχύοντας την προστασία των δεδομένων και την ιδιωτικότητα (privacy), καθώς δεν αποστέλλονται πληροφορίες σε εξωτερικούς διακομιστές. Παράλληλα, εξαλείφεται το κόστος χρήσης API, αφού δεν υπάρχει χρέωση ανά αίτημα ή κατανάλωση tokens, όπως συμβαίνει στις cloud-based υπηρεσίες LLM. Τα χαρακτηριστικά αυτά καθιστούν το Ollama ιδιαίτερα κατάλληλο για επιχειρησιακά περιβάλλοντα όπου απαιτείται μεγάλος αριθμός συγκρίσεων προϊόντων, αυξημένη προστασία δεδομένων και δυνατότητα συνεχούς λειτουργίας χωρίς πρόσθετο λειτουργικό κόστος.

Στην παρούσα εργασία το Ollama χρησιμοποιήθηκε ως τοπικός μηχανισμός επαλήθευσης των αποτελεσμάτων του συστήματος αντιστοίχισης προϊόντων, αντικαθιστώντας το Gemini API χωρίς να μεταβληθεί η συνολική αρχιτεκτονική της εφαρμογής. Με τον τρόπο αυτό κατέστη δυνατή η αξιολόγηση ενός πλήρως τοπικού υβριδικού συστήματος, το οποίο συνδυάζει σημασιολογική αναζήτηση μέσω embeddings και λογική επαλήθευση από LLM χωρίς εξάρτηση από εξωτερικές υπηρεσίες.



3.5.2 Εφαρμογή και Παραδείγματα Τοπικού LLM.

Το script MatchingWithOllama.py αποτελεί την offline εκδοχή του συστήματος αντιστοίχισης προϊόντων που παρουσιάστηκε στην προηγούμενη ενότητα. Η βασική αρχιτεκτονική του συστήματος παραμένει ίδια, καθώς χρησιμοποιούνται τα ίδια embeddings, η ίδια διαδικασία υπολογισμού της σημασιολογικής ομοιότητας (Cosine Similarity) και ο ίδιος μηχανισμός Risk-Adaptive Triage. Η βασική διαφορά είναι ότι η τελική επιβεβαίωση των αμφίβολων περιπτώσεων δεν πραγματοποιείται μέσω του Gemini API, αλλά μέσω του τοπικού μοντέλου Llama3, το οποίο εκτελείται από το Ollama.

Επιπλέον, στο συγκεκριμένο script χρησιμοποιήθηκε και η τεχνική εξαγωγής αριθμητικών χαρακτηριστικών (extract_numbers), η οποία είχε ήδη παρουσιαστεί στο προηγούμενο σύστημα αντιστοίχισης. Ο έλεγχος αυτός χρησιμοποιείται ώστε να εντοπίζονται διαφορές σε αριθμητικές πληροφορίες, όπως χωρητικότητα μνήμης, διαστάσεις οθόνης ή αριθμοί μοντέλων. Όταν εντοπίζονται τέτοιες διαφορές, ακόμη και σε περιπτώσεις με πολύ υψηλό Cosine Similarity, η τελική απόφαση ανατίθεται στο Llama3 ώστε να μειωθούν πιθανές λανθασμένες αντιστοιχίσεις που οφείλονται αποκλειστικά στη σημασιολογική ομοιότητα.

Η λειτουργία του συστήματος παραμένει ίδια με εκείνη του Matching_EMBSEM.py:

- για Cosine Similarity < 0,80 η αντιστοίχιση απορρίπτεται αυτόματα,
- για Cosine Similarity $\geq 0,95$ η αντιστοίχιση εγκρίνεται αυτόματα, εκτός εάν εντοπιστούν διαφορετικά αριθμητικά χαρακτηριστικά, οπότε πραγματοποιείται επιπλέον έλεγχος από το Llama3,
- για τις ενδιάμεσες περιπτώσεις (0,80–0,95) η τελική απόφαση λαμβάνεται από το τοπικό LLM όταν θεωρείται ότι υπάρχει αβεβαιότητα.

Παραδείγματα:

1.

Import	Matching ID	Matching Name	Embedding Score	Ollama Verification	Final Match
MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB ram/256GB SSD/RTX3050/Win10 Home	1	MSI Pavilion 15, Ryzen 7 6800H/15.6" FHD 144Hz/16GB/256GB SSD/RTX3050/Win10 Home	99,82	Yes	Yes
katana dell GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	7	Dell Katana GF66, Ryzen 5 5500U/15.6" FHD 144Hz/16GB/256GB SSD/RTX3060/Win11 Home	99,07	Yes	Yes
galaxy samsung S22 128GB 6.1" OLED Tensor G2 Android	28	Samsung Galaxy S22 128GB 6.1" OLED Tensor G2 Android	99,71	Yes	Yes
Samsung Galaxy A5 16GB EU// Pearl White	51316	Samsung Galaxy A5 16GB EU, Pearl White	99,47	Yes	Yes
AndroidOne GM 5d - 2GB/16GB - Android 7 Nougat – Dual Sim	60457	AndroidOne GM 5d - 2GB/16GB - Android 7 Nougat – Dual Sim - Space Gray	93,22	No	No
XIAOMI NOTE 11S 6/12800GB GRAP	111015	XIAOMI NOTE 11S 6/128GB GRAP	98,93	No	No
Xiaomi Redmi Note 12S, 44GB/128GB, Dual SIM, NFC, Ice Blue (EU) (MZB0DNIEU)	120023	Xiaomi Redmi Note 12, 4GB/128GB, Dual SIM, NFC, Ice Blue (EU) (MZB0DNIEU)	95,27	No	No
Redmi Note 12 Pro 5G// 8GB//256GB//Dual SIM	120026	Redmi Note 12 Pro 5G, 8GB/256GB, Dual SIM	99,3	Yes	Yes
Xiaomi Redmi 13C 8-256 GB Black	123650	Xiaomi Redmi 13C 8/256 GB Midnight Black	96,34	Yes	Yes
POCO C75 6, 128GB 6.88	129284	POCO C75 6+128GB 6.88" Gold DS EU	88,72	No	No
S25 S931B 5G DS 12+256 white	130332	S25 S931B 5G DS 12+256 ITA ICYBLUE	84,22	No	No

Από τα αποτελέσματα παρατηρείται ότι το τοπικό μοντέλο Llama κατέληξε στις ίδιες τελικές αποφάσεις με το αντίστοιχο σύστημα που χρησιμοποιούσε το Gemini. Οι περιπτώσεις με μικρές διαφορές στη μορφοποίηση της περιγραφής, όπως διαφορετικοί διαχωριστές χαρακτήρων ή συντομογραφίες, αναγνωρίστηκαν επιτυχώς ως ίδιες, ενώ προϊόντα με διαφορετικά τεχνικά χαρακτηριστικά ή ακόμα και χρώματα απορρίφθηκαν παρά την υψηλή σημασιολογική ομοιότητα των embeddings.

Ιδιαίτερο ενδιαφέρον παρουσιάζουν οι περιπτώσεις Xiaomi Note 11S και Redmi Note 12S, όπου το Cosine Similarity ήταν ιδιαίτερα υψηλό (98,93% και 95,27% αντίστοιχα), όμως το πρόσθετο στάδιο ελέγχου των αριθμητικών χαρακτηριστικών ενεργοποίησε το Llama3, το οποίο αναγνώρισε ότι οι περιγραφές δεν αντιστοιχούσαν στο ίδιο προϊόν και επέστρεψε την απόφαση No. Το εύρημα αυτό επιβεβαιώνει ότι η συνδυαστική χρήση embeddings και τοπικού LLM περιορίζει λανθασμένες αντιστοιχίσεις που θα μπορούσαν να προκύψουν εάν η απόφαση βασιζόταν αποκλειστικά στο Cosine Similarity.

Συνολικά, τα πειραματικά αποτελέσματα δείχνουν ότι η αντικατάσταση του cloud LLM από ένα τοπικό μοντέλο μέσω του Ollama εδώ δεν επηρέασε την ορθότητα των τελικών αποφάσεων. Η offline υλοποίηση παρήγαγε αντίστοιχης ποιότητας αποτελέσματα, διατηρώντας παράλληλα τα πλεονεκτήματα της τοπικής εκτέλεσης, όπως μηδενικό κόστος χρήσης API, προστασία των δεδομένων και δυνατότητα λειτουργίας χωρίς σύνδεση στο διαδίκτυο.

2. Παράδειγμα με μεγαλύτερο data 3.071 γραμμών και σύγκρισης 357 προϊόντων.

```
Loading weights: 100%|██████████| 103/103 [00:00<00:00, 3880.02it/s]
Executing SQL...
Rows fetched: 3071
Loaded products: 3071
Matching: 100%|██████████| 357/357 [04:41<00:00, 1.27it/s]
Done

Process finished with exit code 0
```

Ανταποκρίθηκε με επιτυχία καμία χρέωση τοπικά στον H/Y.

Καταμέτρηση από Matching	
Ετικέτες γραμμής	ID
No	51
Yes	306
Γενικό Άθροισμα	357

Όμως εντοπίστηκαν και εσφαλμένες απορρίψεις π.χ.:

Import	Matching ID	Matching Name	Embedding Score	Ollama Verification	Final Match
macbook apple Air - M2/15.3 Retina/8GB ram/256GB SSD/10-Core GPU/Webcam/MacOS - Midnight - GR (2023) (MQKW3GR/A)	128584	Apple Macbook Air, M2/15.3 Retina/8GB/256GB SSD/10-Core GPU/Webcam/MacOS, Midnight, GR (2023) (MQKW3GR/A)	94,55999756	No	No
iphone apple 16 Plus 5G - 8GB/128GB - NFC - Ultramarine (EU) (195949722714)(195949722776)	130362	Apple iPhone 16 Plus 5G, 8GB/128GB, NFC, Ultramarine (EU) (195949722714)(195949722776)	94,66000366	No	No
sm-f731 samsung Galaxy Z Flip5 5G / Dual SIM / NFC / Mint (EU) (SM-F731BLGHEUE)(8806095012803) / 8GB/512GB	124398	Samsung SM-F731 Galaxy Z Flip5 5G, 8GB/512GB, Dual SIM, NFC, Mint (EU) (SM-F731BLGHEUE)(8806095012803)	93,95999908	No	No

Συνεπώς μετά από αμερόληπτο δειγματοληπτικό έλεγχο των αποτελεσμάτων διαπιστώθηκε ότι, παρότι η σημασιολογική ομοιότητα μεταξύ των περιγραφών είναι υψηλή, το σύστημα μπορούσε να απορρίπτει λανθασμένα ορισμένες σωστές αντιστοιχίσεις. Για τον λόγο αυτό έγιναν τεχνικές βελτιώσεις στη λογική matching, ώστε να λαμβάνονται υπόψη και κρίσιμα αριθμητικά χαρακτηριστικά, όπως η RAM, η χωρητικότητα αποθήκευσης και το μέγεθος της οθόνης. Επίσης, προστέθηκε έλεγχος στο κείμενο ώστε μικρές αλλαγές στη μορφοποίηση, όπως κόμματα, κάθετοι ή παύλες, να μην επηρεάζουν άδικα το αποτέλεσμα. Έτσι, η τελική αντιστοίχιση έγινε πιο αξιόπιστη όπως παρατηρείται παρακάτω.

Τεχνικά καταλήξαμε σε υβριδικό pipeline αντιστοίχισης προϊόντων, όπου πρώτα γίνεται κανονικοποίηση του κειμένου, token sorting και έλεγχος αριθμητικών χαρακτηριστικών, ώστε να μειώνεται η επίδραση από διαφορές σε στίξη, κεφαλαία/πεζά και σειρά όρων. Στη συνέχεια, τα προϊόντα μετατρέπονται σε dense embeddings με τοπικό μοντέλο, και η αρχική ανάκτηση του πιο κοντινού υποψηφίου γίνεται με cosine similarity. Για τις αμφίβολες περιπτώσεις εφαρμόζεται τοπικό LLM μέσω Ollama για τελική επαλήθευση, ώστε η απόφαση να βασίζεται όχι μόνο σε σημασιολογική ομοιότητα αλλά και σε αυστηρούς κανόνες συμβατότητας τεχνικών προδιαγραφών.

Πράγματι υπήρξε σημαντική βελτίωση.

Ετικέτες γραμμής	Καταμέτρηση από Matching ID
No	12
Yes	345
Γενικό Άθροισμα	357

Με την παραπάνω αλλαγή ανταποκρίθηκε ορθά όπως και φαίνεται στο παρακάτω παράδειγμα:

3.

Import	Matching ID	Matching Name	Embedding Score	Ollama Verification	Final Match
macbook apple Air - M2/15.3 Retina/8GB ram/256GB SSD/10-Core GPU/Webcam/MacOS - Midnight - GR (2023) (MQKW3GR/A)	128584	Apple Macbook Air, M2/15.3 Retina/8GB/256GB SSD/10-Core GPU/Webcam/MacOS, Midnight, GR (2023) (MQKW3GR/A)	99,36000061	Yes	Yes
iphone apple 16 Plus 5G - 8GB/128GB - NFC - Ultramarine (EU) (195949722714)(195949722776)	130362	Apple iPhone 16 Plus 5G, 8GB/128GB, NFC, Ultramarine (EU) (195949722714)(195949722776)	99,77999878	Yes	Yes
sm-f731 samsung Galaxy Z Flip5 5G / Dual SIM / NFC / Mint (EU) (SM-F731BLGHEUE)(8806095012803) / 8GB/512GB	124398	Samsung SM-F731 Galaxy Z Flip5 5G, 8GB/512GB, Dual SIM, NFC, Mint (EU) (SM-F731BLGHEUE)(8806095012803)	98,88999939	Yes	Yes

4.

Import	Matching ID	Matching Name	Embedding Score	Ollama Verification	Final Match
refurbished dell Inspiron 7610, Blue, Grade_A (Εκθεσιακό) (i7610-7333BLU-PUSr) (i7610-7248BLU-PUSr), i7-11800H/16 3K/16GB/1TB SSD/Webcam/Win11 Home	126797	LG Gram Refurbished 16T90R-K.ADB9U1 2in1, i7-1360P/16 WQXGA IPS Touch/32GB/1TB SSD/Webcam/Win11 Home, Black, Grade_A (Εκθεσιακό)	90,07	No	No
Samsung Galaxy X82000 Tab S10+, Gray (8806095597294), 12.4-Inch/12GB ram/512GB/Android	129792	Samsung Galaxy X820 Tab S10+, 12.4-Inch/12GB/256GB/Android, Moonstone Gray (8806095597287)	89,260002	No	No
galaxy samsung SM-X716 Tab S9 / Beige (8806095066394) / 11-Inch/8GB/128GB/5G/Android	129926	Samsung Galaxy SM-X716 Tab S9, 11-Inch/12GB/256GB/5G/Android, Beige (8806095066370)	91,410004	No	No

```
# MatchingWithOllama.py: RAG cosine sim + Local Ollama verification
# BBDBetterMatchingWithOllama/ MatchingWithOllama_Final_StrictNumbers.py runs with udp with the other laptop who have local llama
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn")
import os
import re
import pyodbc
import pandas as pd
import numpy as np
import joblib
from tqdm import tqdm
from sklearn.metrics.pairwise import cosine_similarity
from dotenv import load_dotenv
from sentence_transformers import SentenceTransformer
from openai import OpenAI

# ===== OLLAMA =====
client = OpenAI(
    base_url="http://127.0.0.1:11434/v1",
    api_key="ollama"
)

LLM_MODEL = "llama3.2:3b"
# ===== ENV =====
#load_dotenv("gemini.env")
# ===== EMBEDDING =====
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
CACHE_PATH = "embedding_cache_semantic.pkl"
SQL_FILE = r"C:\Users\User\Desktop\PY\SQLQuerybybigData.sql"
EXCEL_PATH = r"C:\Users\User\Desktop\PY\LLMEmbMatchOll.xlsx"
SHEET_NAME = "a1"
# ===== SQL =====
conn_str = (
    "DRIVER={SQL Server};"
    "SERVER=DESKTOP-HP;"
    "DATABASE=X;"
    "Trusted_Connection=yes;"
)

def get_db_connection():
    return pyodbc.connect(conn_str)

def fetch_product_data():
    if not os.path.exists(SQL_FILE):
        raise FileNotFoundError(f"SQL file not found: {SQL_FILE}")
    with open(SQL_FILE, "r", encoding="utf-8") as f:
        sql_query = f.read()
    conn = get_db_connection()
    cursor = conn.cursor()
    print("Executing SQL...")
    cursor.execute(sql_query)
    while cursor.description is not None:
        if not cursor.nextset():
            raise ValueError("SQL query did not return SELECT results")
    columns = [col[i] for col in cursor.description]
    rows = cursor.fetchall()
    conn.close()
    print(f"Rows fetched: {len(rows)}")
    return [dict(zip(columns, row)) for row in rows]

# ===== CACHE =====
def load_embedding_cache():
    return joblib.load(CACHE_PATH) if os.path.exists(CACHE_PATH) else {}

def save_embedding_cache(cache):
    joblib.dump(cache, CACHE_PATH)

# ===== STRICT TOKENIZER =====
def normalize_text(text):
    text = str(text).lower()
    text = text.replace("&", " ")
    text = text.replace("/", " ")
    text = text.replace("-", " ")
    text = text.replace(".", " ")
    text = text.replace("(", " ").replace(")", " ")
    text = text.replace(":", " ")
    text = re.sub(r"^\s+", " ", text).strip()
    return text

def split_alnum_tokens(text):
    text = normalize_text(text)
    return re.findall(r"^[^\W\d_]+\d+", text, flags=re.UNICODE)

def token_sort_strict(text):
    tokens = split_alnum_tokens(text)
    num_tokens = []
    word_tokens = []
    for tok in tokens:
        if tok.isdigit():
            num_tokens.append(tok)
        else:
            word_tokens.append(tok)
    word_tokens = sorted(word_tokens)
    return " ".join(word_tokens + num_tokens)

def extract_numbers(text):
    return set(re.findall(r"\d+", normalize_text(text)))

def numeric_mismatch(a, b):
    return extract_numbers(a) != extract_numbers(b)

# ===== EMBEDDINGS =====
def get_embeddings_batch(texts, cache):
    embeddings = []
    for text in texts:
        norm = token_sort_strict(text)
        if norm in cache:
            embeddings.append(cache[norm])
        else:
            emb = embedding_model.encode(norm)
            cache[norm] = emb
            embeddings.append(emb)
    return embeddings
```

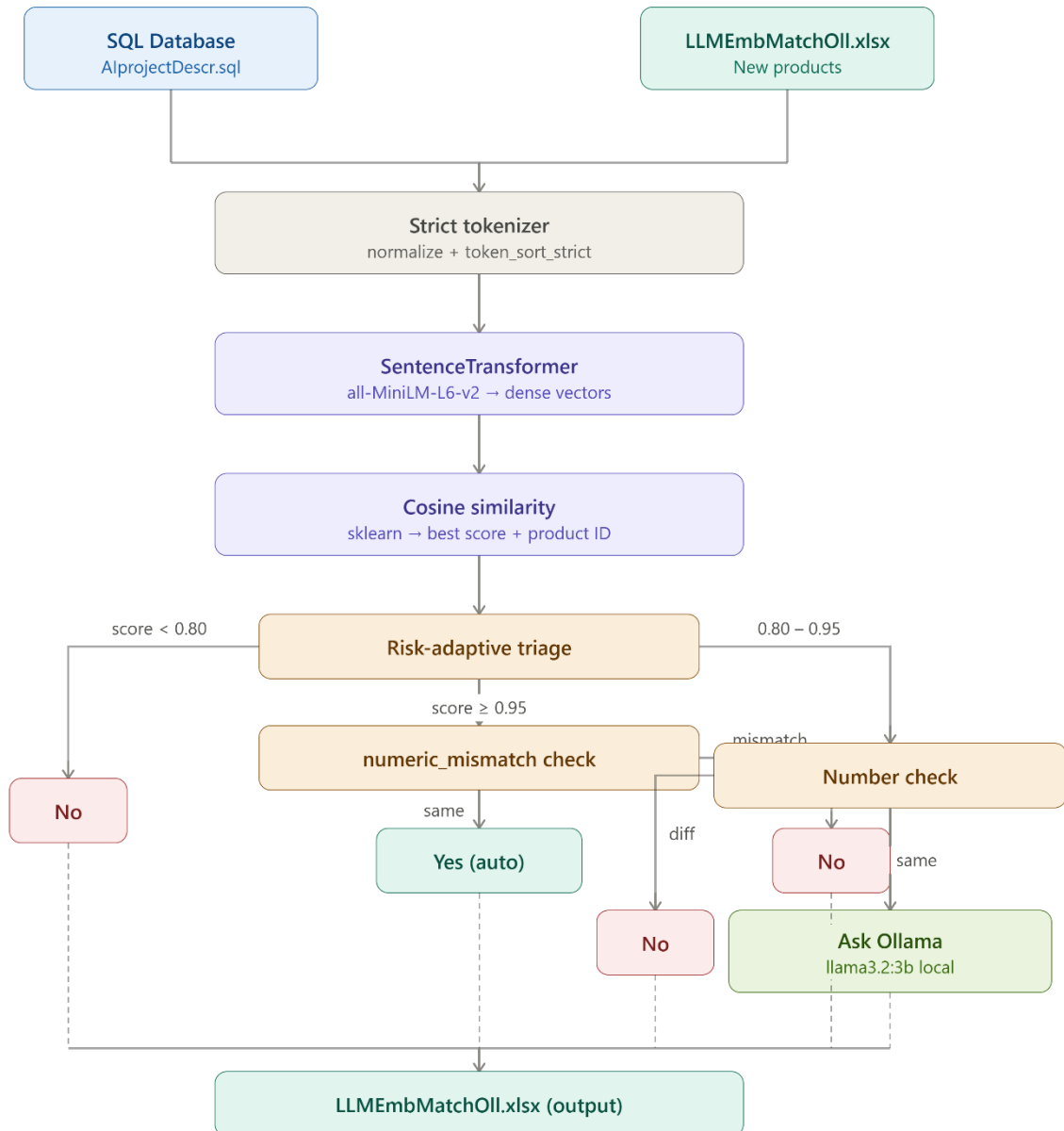
```
# ===== OLLAMA =====
def ask_ollama_yes_no(import_name, matched_name):
    prompt = f"""
You are a strict product matching system.
Decide if these two items are the same exact product.
Rules:
- Same brand and same model are required.
- Same RAM is required.
- Same storage is required.
- Same screen size is required.
- Same CPU/GPU is required when present.
- Different category means No.
- Different numbers usually mean No.
- Different punctuation, commas, slashes, dashes, capitalization, or word order do not matter.
- If the items are the same product with only formatting differences, answer Yes.
Answer only:
Yes
or
No
Import:
{import_name}
Matched:
{matched_name}
"""
    try:
        response = client.chat.completions.create(
            model=LLM_MODEL,
            messages=[{"role": "user", "content": prompt}],
            temperature=0.0,
            max_completion_tokens=10
        )
        reply = response.choices[0].message.content.strip().lower()
        return "Yes" if reply.startswith("yes") else "No"
    except Exception as e:
        print(f"Ollama local API error: {e}")
        return "Error"

# ===== MATCHING =====
def find_best_match(import_embedding, product_dbs, threshold=0.75):
    db_embeddings = [p["embedding"] for p in product_dbs]
    sims = cosine_similarity([import_embedding], db_embeddings)[0]
    best_idx = np.argmax(sims)
    best_score = sims[best_idx]
    best_id = product_dbs[best_idx]["ProductID"]
    if best_score < threshold:
        return None, best_score
    return best_id, best_score

# ===== MAIN =====
def process_supplier_matching():
    df = pd.read_excel(EXCEL_PATH, sheet_name=SHEET_NAME, engine="openpyxl")
    if "Import" not in df.columns:
        raise Exception("Import column not found")
    embedding_cache = load_embedding_cache()

    product_data = fetch_product_data()
    name_by_id = {p["ProductID"]: p["Descr"] for p in product_data}
    product_texts = [p["Descr"] for p in product_data]
    product_embeddings = get_embeddings_batch(product_texts, embedding_cache)
    product_db = [
        {"ProductID": p["ProductID"], "name": p["Descr"], "embedding": emb}
        for p, emb in zip(product_data, product_embeddings)
    ]
    print(f"Loaded products: {len(product_db)}")
    import_texts = df["Import"].fillna("").astype(str).tolist()
    import_embeddings = get_embeddings_batch(import_texts, embedding_cache)
    save_embedding_cache(embedding_cache)
    results = []
    for i, import_text in enumerate(tqdm(import_texts, desc="Matching")):
        emb = import_embeddings[i]
        best_id, score = find_best_match(emb, product_db)
        if best_id is None:
            results.append((None, "No Match", round(score * 100, 2), "No", "No"))
            continue
        matched_name = name_by_id[best_id]
        if score < 0.80:
            verdict = "No"
            final = "No"
        elif score ≥ 0.95:
            if numeric_mismatch(import_text, matched_name):
                verdict = "No"
                final = "No"
            else:
                verdict = "Yes"
                final = "Yes"
        else:
            nums_import = extract_numbers(import_text)
            nums_matched = extract_numbers(matched_name)
            if nums_import ≠ nums_matched:
                verdict = "No"
                final = "No"
            else:
                verdict = ask_ollama_yes_no(import_text, matched_name)
                final = "Yes" if verdict == "Yes" else "No"
        results.append((
            best_id,
            matched_name,
            round(score * 100, 2),
            verdict,
            final
        ))
    result_df = pd.DataFrame(results, columns=[
        "Matching ID",
        "Matching Name",
        "Embedding Score",
        "Ollama Verification",
        "Final Match"
    ])
    result_df.insert(0, "Import", df["Import"])
    with pd.ExcelWriter(
        EXCEL_PATH,
        engine="openpyxl",
        mode="a",
        if_sheet_exists="replace"
    ) as writer:
        result_df.to_excel(writer, sheet_name=SHEET_NAME, index=False)
    save_embedding_cache(embedding_cache)
    print("Done")
if __name__ == "__main__":
    process_supplier_matching()
```

Flowchart του MatchingWithOllama:



Βασική διαφορά από το προηγούμενο flowchart είναι τα 2 νέα στάδια στην αρχή και στη μέση: Strict tokenizer που κανονικοποιεί το κείμενο πριν το embedding και ο διπλός αριθμητικός έλεγχος — ένας στο $\text{score} \geq 0.95$ (numeric_mismatch) και ένας στο $0.80-0.95$ (extract_numbers) — πριν φτάσει στο Ollama.

3.6 Ανατροφοδότηση και Συνεχής Βελτίωση.

Η ανάπτυξη του προτεινόμενου συστήματος πραγματοποιήθηκε με επαναληπτική διαδικασία συνεχούς αξιολόγησης και βελτίωσης. Μετά από κάθε στάδιο δοκιμών πραγματοποιούνταν δειγματοληπτικός έλεγχος των αποτελεσμάτων, με στόχο τον εντοπισμό λανθασμένων αντιστοιχίσεων και την κατανόηση των αιτιών που τις προκαλούσαν. Η διαδικασία αυτή επέτρεψε τη σταδιακή αναβάθμιση τόσο της λογικής αντιστοίχισης όσο και των επιχειρησιακών κανόνων (business rules) που εφαρμόζει το σύστημα.

Από την αξιολόγηση των αρχικών αποτελεσμάτων διαπιστώθηκε ότι η χρήση μόνο της σημασιολογικής ομοιότητας (Cosine Similarity) δεν ήταν επαρκής, καθώς περιγραφές με υψηλή σημασιολογική ομοιότητα μπορούσαν να αφορούν διαφορετικά προϊόντα. Για τον λόγο αυτό προστέθηκαν επιπλέον μηχανισμοί, όπως η κανονικοποίηση των περιγραφών, ο έλεγχος αριθμητικών χαρακτηριστικών (RAM, χωρητικότητα αποθήκευσης, μέγεθος οθόνης, κωδικοί μοντέλων κ.ά.) και η τελική επαλήθευση από μεγάλο γλωσσικό μοντέλο μόνο στις περιπτώσεις που παρουσίαζαν αβεβαιότητα. Ειδικότερα δε κανένα LLM από όσα εξετάστηκαν εδώ αλλά και πρακτικά που έχω δει σε επιχειρήσεις δεν επαρκεί από μόνο του για σύνθετες διαδικασίες όπως ήταν εδώ η αντιστοίχιση σύνθετων και πολύπλοκων περιγραφών προϊόντων με την τοπική βάση.

Η ίδια διαδικασία εφαρμόστηκε τόσο στην cloud υλοποίηση με το Gemini όσο και στην τοπική υλοποίηση μέσω Ollama, όπου πραγματοποιήθηκαν επαναλαμβανόμενες δοκιμές σε διαφορετικά σύνολα δεδομένων. Οι βελτιώσεις που προέκυψαν ενσωματώνονταν σταδιακά στο σύστημα και επανελέγχονταν σε νέα πειραματικά δεδομένα, μέχρι να επιτευχθεί σταθερή και αξιόπιστη συμπεριφορά.

Η προσέγγιση αυτή οδήγησε στη δημιουργία ενός υβριδικού συστήματος αντιστοίχισης προϊόντων, το οποίο συνδυάζει σημασιολογική αναζήτηση, κανόνες επιχειρησιακής λογικής και επαλήθευση μέσω LLM. Παράλληλα, η αρχιτεκτονική του επιτρέπει την εύκολη προσθήκη νέων κανόνων ή την αντικατάσταση των επιμέρους μοντέλων χωρίς να απαιτούνται σημαντικές αλλαγές στη συνολική λειτουργία του συστήματος, γεγονός που το καθιστά επεκτάσιμο και κατάλληλο για μελλοντικές εφαρμογές σε μεγαλύτερα και πιο ετερογενή σύνολα προϊόντων.

4. Συμπεράσματα Επέκταση και αξιολόγηση συστημάτων:

4.1 Αξιολόγηση των Προτεινόμενων Συστημάτων.

Στο κεφάλαιο αυτό παρουσιάζεται η συνολική αξιολόγηση των διαφορετικών προσεγγίσεων που υλοποιήθηκαν κατά τη διάρκεια της εργασίας. Συγκρίνονται τα αποτελέσματα των μεθόδων που βασίζονται αποκλειστικά σε embeddings, της προσέγγισης Embeddings + Machine Learning (XGBoost), του υβριδικού συστήματος Embeddings + Gemini, καθώς και της αντίστοιχης offline υλοποίησης μέσω Ollama. Αναλύονται η ακρίβεια των αντιστοιχίσεων, η αξιοπιστία των αποφάσεων, ο χρόνος εκτέλεσης και η συμβολή κάθε επιμέρους τεχνικής στη συνολική απόδοση του συστήματος. Ιδιαίτερη έμφαση δίνεται στα πλεονεκτήματα της υβριδικής αρχιτεκτονικής, η οποία συνδυάζει σημασιολογική αναζήτηση, επιχειρησιακούς κανόνες και επαλήθευση μέσω μεγάλων γλωσσικών μοντέλων.

4.2 Συμπεράσματα Επέκταση και Επιχειρησιακή Αξιοποίηση.

Η υλοποίηση απέδειξε ότι η ορθή αντιστοίχιση προϊόντων αποτελεί βασική προϋπόθεση για την αξιοποίηση των πληροφοριών που είναι ήδη αποθηκευμένες στη βάση δεδομένων της επιχείρησης. Μετά την επιτυχή αντιστοίχιση ενός προϊόντος με το αντίστοιχο ProductID, μπορούν να ανακτηθούν αυτόματα στοιχεία όπως οι τιμές αγοράς και πώλησης, η τιμή με ΦΠΑ, τα διαθέσιμα αποθέματα, οι αναμενόμενες παραλαβές, οι παραγγελίες, ο κατασκευαστής και κάθε άλλο χαρακτηριστικό που είναι διαθέσιμο στο πληροφοριακό σύστημα.

Η δυνατότητα αυτή επιτρέπει στο εμπορικό τμήμα να συγκρίνει άμεσα προσφορές διαφορετικών προμηθευτών για το ίδιο προϊόν, να αξιολογεί τη διαθεσιμότητα και να λαμβάνει ταχύτερες και πιο αξιόπιστες αποφάσεις σχετικά με τις αγορές και την τιμολογιακή πολιτική. Παράλληλα, το σύστημα μπορεί να επεκταθεί εύκολα με επιπλέον αναφορές (reports) και dashboards που θα υποστηρίζουν τη λήψη επιχειρησιακών αποφάσεων, όπως συγκρίσεις τιμών, υπολογισμό περιθωρίου κέρδους, ιστορική παρακολούθηση μεταβολών τιμών και αξιολόγηση προμηθευτών.

4.2.1 Παράδειγμα Επέκτασης και Αξιοποίησης του Συστήματος.

Μετά την επιτυχή εφαρμογή και αξιολόγηση του υβριδικού συστήματος αντιστοίχισης σε μεγάλο σύνολο δεδομένων δωρεάν μέσω συμμετοχής και του τοπικού LLM Ollama χιλιάδες αντιστοιχήσεις με σύνθετες περιγραφές που αντιστοιχούν σε μεγάλο κόστος token, το επόμενο βήμα θα ήταν η αξιοποίηση των αποτελεσμάτων για την υποστήριξη επιχειρησιακών αποφάσεων. Η ορθή αντιστοίχιση των περιγραφών με τα αντίστοιχα primary keys ID επέτρεψε την αυτόματη σύνδεση των δεδομένων του προμηθευτή με τις πληροφορίες που είναι ήδη αποθηκευμένες στη βάση δεδομένων της επιχείρησης.

Για τον σκοπό αυτό αναπτύχθηκε προσθήκη της εφαρμογής, η οποία διατηρεί αμετάβλητο τον μηχανισμό αντιστοίχισης (κανονικοποίηση κειμένου, token sorting, embeddings, cosine similarity, αριθμητικό έλεγχο και επαλήθευση μέσω τοπικού LLM), ενώ μετά την επιβεβαίωση της αντιστοίχισης προχωρά σε επιχειρησιακή ανάλυση των αποτελεσμάτων. Συγκεκριμένα, η εφαρμογή διαβάζει τις τιμές των προϊόντων από το αρχείο XML του προμηθευτή και τις συγκρίνει με τις αντίστοιχες τιμές πώλησης που ανακτώνται από τη βάση δεδομένων SQL Server.

Με βάση τα δεδομένα αυτά υπολογίζεται αυτόματα η διαφορά τιμής μεταξύ προμηθευτή και επιχείρησης και παράγεται μία αρχική ένδειξη αγοράς (BuyOrNot), η οποία υποδεικνύει εάν η προσφορά του προμηθευτή είναι οικονομικά συμφέρουσα σύμφωνα με τον επιλεγμένο κανόνα αξιολόγησης εδώ απλά ως παράδειγμα αν είναι φθηνότερο το προϊόν του. Με τον τρόπο αυτό η διαδικασία δεν περιορίζεται μόνο στην αναγνώριση του ίδιου προϊόντος, αλλά επεκτείνεται στην αυτόματη αξιολόγηση εμπορικών ευκαιριών.

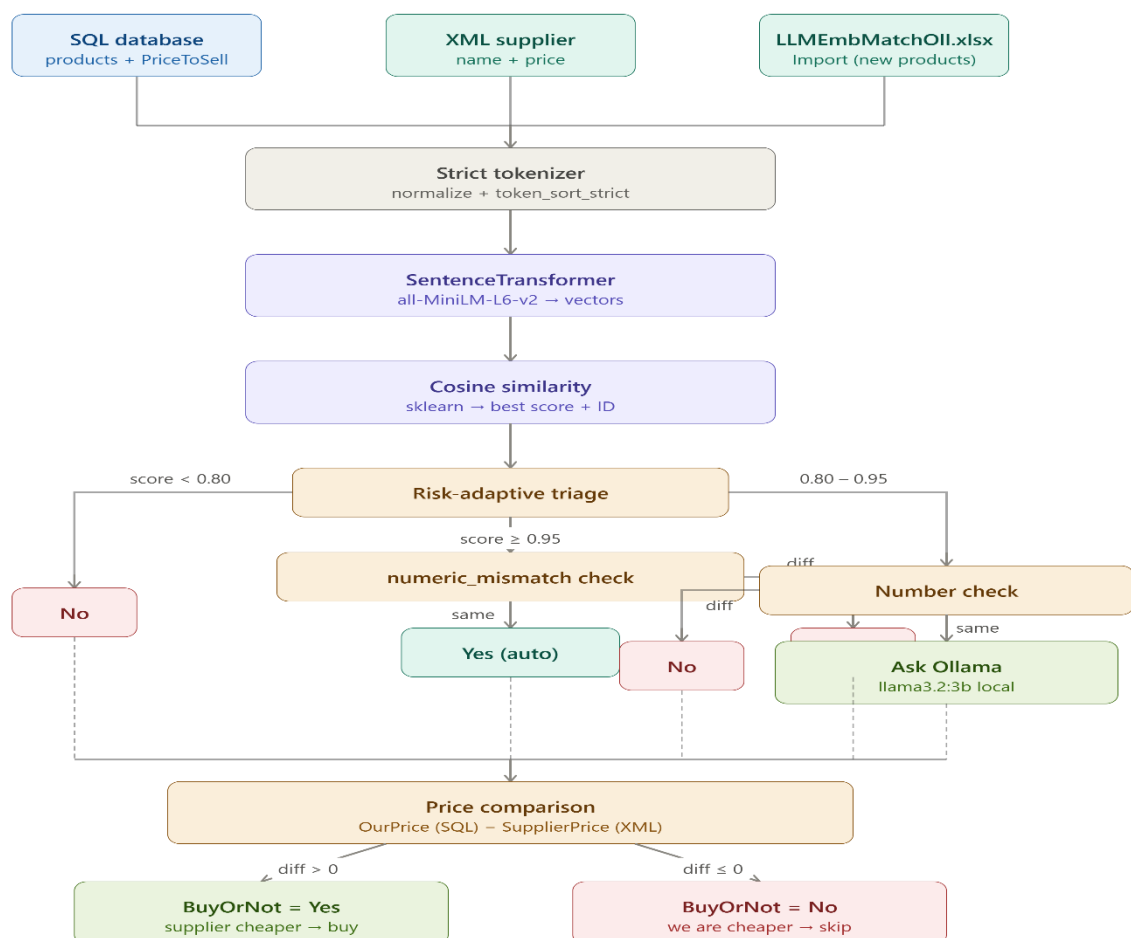
Όπως βέβαια προειπώθηκε το κλειδί είναι να γίνει η σωστή αντιστοιχία των προϊόντων είναι να γίνουν matching οι περιγραφές με το αντίστοιχο id primary key της ΒΔ μετά η όποια ανάλυση γίνεται πολύ πιο εύκολα. Η λεπτομέρεια της περιγραφής είναι καθοριστική καθώς ακόμα και μια μικρή διαφορά π.χ. το χρώμα ίδιας συσκευής μπορεί να κάνει διαφορά στην τιμή και την κερδοφορία ή όχι της επιχείρησης.

Η συνολική αρχιτεκτονική του συστήματος διαμορφώνεται πλέον σε τρία διακριτά επίπεδα:

1. Το πρώτο επίπεδο αφορά την εισαγωγή δεδομένων από ετερογενείς πηγές (XML, SQL Server και Excel).

2. Το δεύτερο επίπεδο περιλαμβάνει τη σημασιολογική αντιστοίχιση των προϊόντων μέσω διαδοχικών σταδίων επεξεργασίας, όπως η κανονικοποίηση του κειμένου, η δημιουργία embeddings, ο υπολογισμός της cosine similarity, ο έλεγχος αριθμητικών χαρακτηριστικών και όπου απαιτείται, η τελική επαλήθευση από το τοπικό μοντέλο Llama μέσω Ollama.
3. Το τρίτο επίπεδο αφορά την επιχειρησιακή αξιοποίηση των αποτελεσμάτων, όπου πραγματοποιούνται συγκρίσεις τιμών και εντοπισμός πιθανών ευκαιριών αγοράς. Εδώ να προσθέσουμε ότι μπορεί σε ένα Data Lake να συλλέγονται δεδομένα και να αναλύονται από πολλαπλούς προμηθευτές και το εμπορικό τμήμα να έχει έτοιμο report ώστε να προβεί στις δικές του ενέργειες αξιοποίησης της πληροφορίας.

Αξίζει να σημειωθεί ότι το τοπικό LLM χρησιμοποιείται αποκλειστικά για τη σημασιολογική επαλήθευση των περιγραφών προϊόντων στις αμφίβολες περιπτώσεις αντιστοίχισης. Οι οικονομικοί υπολογισμοί, οι συγκρίσεις τιμών και οι αποφάσεις που σχετίζονται με την εμπορική αξιολόγηση βασίζονται αποκλειστικά σε αντικειμενικά αριθμητικά δεδομένα και προκαθορισμένους επιχειρησιακούς κανόνες, διασφαλίζοντας ότι το γλωσσικό μοντέλο εδώ δεν επηρεάζει τις οικονομικές αποφάσεις.



```

Loading weights: 100%|██████████| 103/103 [00:00<00:00, 3561.82it/s]
XML prices loaded: 7515
Executing SQL...
Rows fetched: 11611
Loaded products: 11611
Matching: 100%|██████████| 7516/7516 [11:31<00:00, 10.87it/s]
Done
Process finished with exit code 0

```

Παράδειγμα εφαρμογής σε πραγματικά δεδομένα σχετικά μεγάλης κλίμακας:

Μετά την αξιολόγηση του αλγορίθμου ακαδημαϊκά σε μικρότερα σύνολα δεδομένων, πραγματοποιήθηκε δοκιμή σε random πραγματικά δεδομένα αγοράς μεγάλης κλίμακας, ώστε να αξιολογηθεί η συμπεριφορά του συστήματος σε συνθήκες λειτουργίας εμπορικής επιχείρησης.

Συγκεκριμένα, το σύστημα επεξεργάστηκε 7.516 περιγραφές προϊόντων προμηθευτή, οι οποίες συγκρίθηκαν με 11.611 προϊόντα που ήταν αποθηκευμένα στη βάση δεδομένων SQL Server. Κατά την εκτέλεση φορτώθηκαν επιτυχώς οι τιμές από το XML του προμηθευτή 7.515 εγγραφές, ανακτήθηκαν τα προϊόντα από τη βάση δεδομένων και ολοκληρώθηκε η διαδικασία αντιστοίχισης όλων των εγγραφών επιτυχώς χωρίς σφάλματα.

Η συνολική διαδικασία ολοκληρώθηκε σε περίπου 11 λεπτά και 31 δευτερόλεπτα, γεγονός που αποδεικνύει ότι η προτεινόμενη υβριδική αρχιτεκτονική μπορεί να εφαρμοστεί αποτελεσματικά ακόμη και σε μεγάλα σύνολα δεδομένων.

Import	Matching ID	Matching Name	Embedding Score	Ollama Verification	Final Match	SupplierPrice	OurPrice	PriceDiff	BuyOrNot
Lexmark MS823dn, Duplex, USB, 1200x1200dpi, A4 Mono Laser Printer (Print/Scan/Copy), Ethernet (50G0220), 61ppm	118641	Lexmark MS823dn, A4 Mono Laser Printer (Print/Scan/Copy), Duplex, 1200x1200dpi, 61ppm, USB, Ethernet (50G0220)	99,91999817	Yes	Yes	643,11	705.48	62,37	Yes
replacement apc Battery Cartridge 133 (APCRBC133)	65848	APC Replacement Battery Cartridge 133 (APCRBC133)	100	Yes	Yes	400,13	392.24	-7,89	No
Cisco MGB ramSX1 / Gigabit SX Mini-GBIC SFP Transceiver	39939	Cisco MGBSX1, Gigabit SX Mini-GBIC SFP Transceiver	97,05000305	Yes	Yes	126,59	136.49	9,9	Yes
Corsair Dominator Platinum RGB 32GB Kit (2x16GB) 6200MHz UDIMM DDR5 CL36 1.30V / Black (CMT32GX5M2X6200C36)	115878	Corsair Dominator Platinum RGB 32GB Kit (2x16GB) 6200MHz UDIMM DDR5 CL36 1.30V, Black (CMT32GX5M2X6200C36)	100	Yes	Yes	158,76	163.45	4,69	Yes

...

Import	Matching ID	Matching Name	Embedding Score	Ollama Verification	Final Match	SupplierPrice	OurPrice	PriceDiff	BuyOrNot
sm-s931 samsung Galaxy S25 5G - 12GB ram/128GB - NFC - Navy (EU) (8806095852393)	130341	Samsung SM-S931 Galaxy S25 5G, 12GB/256GB, NFC, Navy (EU) (8806095852065)	91,23000336	No	No	692,1	744.74	52,64	Yes
Western Digital Ultrastar 7K6 3.5 4000 GB ram Serial ATA III (0B35950)	127641	Western Digital Ultrastar 7K6 3.5 4000 GB Serial ATA III (0B35950) old	91,58999634	No	No	161,95	162.47	0,52	Yes

Παρατηρείται ότι το σύστημα αναγνώρισε επιτυχώς την πλειονότητα των προϊόντων, ενώ μόλις 28 εγγραφές απορρίφθηκαν ως μη έγκυρες αντιστοιχίσεις, επιβεβαιώνοντας την σχετικά υψηλή αξιοπιστία του μηχανισμού semantic matching και της επαλήθευσης μέσω τοπικού LLM.

Μετά την ολοκλήρωση της αντιστοίχισης, η εφαρμογή προχώρησε αυτόματα στη σύγκριση των τιμών μεταξύ του προμηθευτή και της βάσης δεδομένων της επιχείρησης, παράγοντας ένδειξη πιθανής ευκαιρίας αγοράς (BuyOrNot).

Το αποτέλεσμα αυτό δείχνει ότι σχεδόν οι μισές από τις επιτυχώς αντιστοιχισμένες εγγραφές αντιστοιχούσαν σε περιπτώσεις όπου η προσφορά του προμηθευτή κρίθηκε φθηνότερη και πιθανός συμφέρουσα σύμφωνα με τον κανόνα αξιολόγησης που εφαρμόστηκε. Αντίστοιχα, στις υπόλοιπες περιπτώσεις η υπάρχουσα τιμή της επιχείρησης ήταν ήδη χαμηλότερη ή ίση, επομένως δεν προέκυψε εμπορική ευκαιρία.

Final Match	Καταμέτρηση από Matching ID
No	28
No Buy	13
Yes Buy	15
Yes	7488
No Buy	3767
Yes Buy	3721
Γενικό Άθροισμα	7516

Αξίζει να σημειωθεί ότι ολόκληρη η διαδικασία πραγματοποιήθηκε τοπικά, χωρίς αποστολή δεδομένων σε εξωτερικές υπηρεσίες τεχνητής νοημοσύνης. Το μοντέλο Llama εκτελέστηκε μέσω Ollama στον ίδιο υπολογιστή, ενώ τα embeddings δημιουργήθηκαν επίσης τοπικά. Κατά συνέπεια, δεν χρησιμοποιήθηκαν cloud APIs και δεν υπήρξε καμία χρέωση χρήσης ή κατανάλωση API tokens.

Ενδεικτικά, περιγραφές προϊόντων όπως P300 Toshiba 2TB HDD / 7200rpm (HDWD320UZSVAJ) / 3.5-Inch / SATA3 / 256MB Cache ή Corsair Dominator Titanium RGB 64GB Kit (2x32GBs) 6400MHz UDIMM DDR5 CL32 1.40V, White (CMP64GX5M2B6400C32WT), περιλαμβάνουν μεγάλο πλήθος τεχνικών χαρακτηριστικών και θα απαιτούσαν σημαντικό αριθμό tokens εάν αποστέλλονταν σε κάποιο cloud LLM. Σε πραγματικές συνθήκες, η επεξεργασία χιλιάδων αντίστοιχων περιγραφών θα οδηγούσε σε αισθητό λειτουργικό κόστος από την κατανάλωση εισερχόμενων και εξερχόμενων tokens. Με την τοπική εκτέλεση μέσω Ollama το κόστος αυτό ήταν πρακτικά μηδενικό, ανεξάρτητα από το πλήθος των προϊόντων που επεξεργάστηκε το σύστημα.

```
# MatchingWithOllama Price.py
# Same as MatchingWithOllama_Final_StrictNumbers.py but adds price comparison:
# - SupplierPrice : from XML (SupplierProducts.xml)
# - OurPrice : from SQL (PriceToSell)
# - PriceDiff : OurPrice - SupplierPrice
# - BuyOrNot : Yes if OurPrice > SupplierPrice (supplier cheaper), No otherwise
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn")
import os
import re
import xml.etree.ElementTree as ET
import pyodbc
import pandas as pd
import numpy as np
import joblib
from tqdm import tqdm
from sklearn.metrics.pairwise import cosine_similarity
from sentence_transformers import SentenceTransformer
from openai import OpenAI
# ===== OLLAMA =====
client = OpenAI(
    base_url="http://127.0.0.1:11434/v1",
    api_key="ollama"
)
LLM_MODEL = "llama3.2:3b"
# ===== EMBEDDING =====
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")
CACHE_PATH = "embedding_cache_semantic.pkl"
SQL_FILE = r"C:\Users\User\Desktop\PY\SQLQuerybigData.sql"
EXCEL_PATH = r"C:\Users\User\Desktop\PY\LLMEmbMatchOll.xlsx"
XML_PATH = r"C:\Users\User\Desktop\PY\SupplierProducts.xml"
SHEET_NAME = "sa"
# ===== SQL =====
conn_str = (
    "DRIVER={SQL Server};"
    "SERVER=DESKTOP-M;"
    "DATABASE=ag;"
    "Trusted_Connection=yes;"
)
def get_db_connection():
    return pyodbc.connect(conn_str)
def fetch_product_data():
    if not os.path.exists(SQL_FILE):
        raise FileNotFoundError(f"SQL file not found: {SQL_FILE}")
    with open(SQL_FILE, "r", encoding="utf-8") as f:
        sql_query = f.read()
    conn = get_db_connection()
    cursor = conn.cursor()
    print(f"Executing SQL: {sql_query}")
    cursor.execute(sql_query)
    while cursor.description is None:
        if not cursor.nextset():
            raise ValueError("SQL query did not return SELECT results")
    columns = [col[0] for col in cursor.description]
    rows = cursor.fetchall()
    conn.close()
    print(f"Rows fetched: {len(rows)}")
    return [dict(zip(columns, row)) for row in rows]
# ===== XML PRICE READER =====
def read_xml_prices(xml_path):
    """Returns dict: {product name stripped: price} from XML."""
    if not os.path.exists(xml_path):
        print(f"⚠ XML not found: {xml_path} - SupplierPrice will be empty")
        return {}
    tree = ET.parse(xml_path)
    root = tree.getroot()
    prices = {}
    for product in root.findall(".//product"):
        name = product.findtext("name")
        price = product.findtext("price")
        if name and price:
            try:
                prices[name.strip()] = float(price)
            except ValueError:
                pass
    print(f"XML prices loaded: {len(prices)}")
    return prices
# ===== CACHE =====
def load_embedding_cache():
    return joblib.load(CACHE_PATH) if os.path.exists(CACHE_PATH) else {}
def save_embedding_cache(cache):
    joblib.dump(cache, CACHE_PATH)
# ===== STRICT TOKENIZER =====
def normalize_text(text):
    text = str(text).lower()
    for ch in ["&", "/", "=", ",", "(", ")", "!", ":"]:
        text = text.replace(ch, " ")
    text = re.sub(r"\s+", " ", text).strip()
    return text
def split_alnum_tokens(text):
    text = normalize_text(text)
    return re.findall(r"[^\W\d_]+|\d+", text, flags=re.UNICODE)
def token_sort_strict(text):
    tokens = split_alnum_tokens(text)
    num_tokens = [t for t in tokens if t.isdigit()]
    word_tokens = sorted(t for t in tokens if not t.isdigit())
    return " ".join(word_tokens + num_tokens)
def extract_numbers(text):
    return set(re.findall(r"[^\d+]", normalize_text(text)))
def numeric_mismatch(a, b):
    return extract_numbers(a) != extract_numbers(b)
# ===== EMBEDDINGS =====
def get_embeddings_batch(texts, cache):
    embeddings = []
    for text in texts:
        norm = token_sort_strict(text)
        if norm in cache:
            embeddings.append(cache[norm])
        else:
            emb = embedding_model.encode(norm)
            cache[norm] = emb
            embeddings.append(emb)
    return embeddings
# ===== OLLAMA =====
def ask_ollama_yes_no(import_name, matched_name):
    prompt = """
    You are a strict product matching system.
    Decide if these two items are the same exact product.
    Rules:
    - Same brand and same model are required.
    - Same RAM is required.
    - Same storage is required.
    - Same screen size is required.
    - Same CPU/GPU is required when present.
    - Different category means No.
    - Different numbers usually mean No.
    - Different punctuation, commas, slashes, dashes, capitalization, or word order do not matter.
    - If the items are the same product with only formatting differences, answer Yes.
    Answer only:
    Yes
    or
    No
    """
    import re
    matched = re.match(r"(import_name)", import_name)
    matched = re.match(r"(matched_name)", matched_name)
    return
```

```

try:
    response = client.chat.completions.create(
        model=LLM_MODEL,
        messages=[{"role": "user", "content": prompt}],
        temperature=0.0,
        max_completion_tokens=10
    )
    reply = response.choices[0].message.content.strip().lower()
    return "Yes" if reply.startswith("yes") else "No"
except Exception as e:
    print(f"Ollama local API error: {e}")
    return "Error"

# ===== MATCHING =====
def find_best_match(import_embedding, product_dbs, threshold=0.75):
    db_embeddings = [p["embedding"] for p in product_dbs]
    sims = cosine_similarity([import_embedding], db_embeddings)[0]
    best_idx = np.argmax(sims)
    best_score = sims[best_idx]
    best_id = product_dbs[best_idx]["ProductID"]
    if best_score < threshold:
        return None, best_score
    return best_id, best_score

# ===== MAIN =====
def process_supplier_matching():
    df = pd.read_excel(EXCEL_PATH, sheet_name=SHEET_NAME, engine="openpyxl")
    if "Import" not in df.columns:
        raise Exception("Import column not found")
    # --- supplier prices from XML ---
    xml_prices = read_xml_prices(XML_PATH)
    # map each import row to its XML price (exact name match)
    supplier_price_col = []
    for imp in df["Import"].fillna("").astype(str).tolist():
        supplier_price_col.append(xml_prices.get(imp.strip(), None))
    embedding_cache = load_embedding_cache()
    # --- SQL products ---
    product_data = fetch_product_data()
    name_by_id = {p["ProductID"]: p["Descr"] for p in product_data}
    price_by_id = {p["ProductID"]: p.get("PriceToSell") for p in product_data}
    product_texts = [p["Descr"] for p in product_data]
    product_embeddings = get_embeddings_batch(product_texts, embedding_cache)
    product_db = [
        {"ProductID": p["ProductID"], "name": p["Descr"], "embedding": emb}
        for p, emb in zip(product_data, product_embeddings)
    ]
    print(f"Loaded products: {len(product_db)}")
    import_texts = df["Import"].fillna("").astype(str).tolist()
    import_embeddings = get_embeddings_batch(import_texts, embedding_cache)
    save_embedding_cache(embedding_cache)
    results = []
    for i, import_text in enumerate(tqdm(import_texts, desc="Matching")):
        emb = import_embeddings[i]
        best_id, score = find_best_match(emb, product_db)
        if best_id is None:
            results.append((None, "No Match", round(score * 100, 2),
                           "No", "No", None, None, None, None))
            continue
        matched_name = name_by_id[best_id]
        # --- triage ---
        if score < 0.80:
            verdict = "No"
            final = "No"
        elif score ≥ 0.95:
            if numeric_mismatch(import_text, matched_name):
                verdict = "No"
                final = "No"
            else:
                verdict = "Yes"
                final = "Yes"
        else:
            if extract_numbers(import_text) ≠ extract_numbers(matched_name):
                verdict = "No"
                final = "No"
            else:
                verdict = ask_ollama_yes_no(import_text, matched_name)
                final = "Yes" if verdict == "Yes" else "No"
        # --- price compare ---
        supplier_price = supplier_price_col[i] # from XML
        our_price = price_by_id.get(best_id) # from SQL
        if supplier_price is not None and our_price is not None:
            try:
                sup = float(supplier_price)
                our = float(our_price)
                diff = round(our - sup, 2) # positive = we are more expensive
                buy_or_not = "Yes" if diff > 0 else "No" # Yes = buy from supplier
            except (ValueError, TypeError):
                diff = buy_or_not = None
        else:
            diff = buy_or_not = None
        results.append((
            best_id,
            matched_name,
            round(score * 100, 2),
            verdict,
            final,
            supplier_price, # SupplierPrice (from XML)
            our_price, # OurPrice (from SQL PriceToSell)
            diff, # PriceDiff (OurPrice - SupplierPrice)
            buy_or_not # BuyOrNot (Yes if we are more expensive)
        ))
    result_df = pd.DataFrame(results, columns=[
        "Matching ID",
        "Matching Name",
        "Embedding Score",
        "Ollama Verification",
        "Final Match",
        "SupplierPrice", # XML price
        "OurPrice", # SQL PriceToSell
        "PriceDiff", # OurPrice - SupplierPrice
        "BuyOrNot", # Yes = supplier is cheaper → consider buying
    ])
    result_df.insert(0, "Import", df["Import"])
    with pd.ExcelWriter(
        EXCEL_PATH,
        engine="openpyxl",
        mode="a",
        if_sheet_exists="replace"
    ) as writer:
        result_df.to_excel(writer, sheet_name=SHEET_NAME, index=False)
    save_embedding_cache(embedding_cache)
    print("Done")
if __name__ == "__main__":
    process_supplier_matching()

```

4.2.2 Δειγματοληπτικός Έλεγχος και Ποιοτική Αξιολόγηση των Αποτελεσμάτων.

Μετά την ολοκλήρωση της διαδικασίας αντιστοίχισης και της σύγκρισης τιμών, πραγματοποιήθηκε ένας επιπλέον ποιοτικός έλεγχος των αποτελεσμάτων με σκοπό την αξιολόγηση της ορθότητας των αποφάσεων που παρήγαγε το σύστημα. Για τον σκοπό αυτό αναπτύχθηκε ξεχωριστή εφαρμογή ελέγχου (audit), η οποία αξιοποιεί το τελικό αρχείο αποτελεσμάτων που παράγεται από το σύστημα αντιστοίχισης.

Η εφαρμογή επιλέγει αυτόματα ένα αμερόληπτο τυχαίο στατιστικό δείγμα 33 εγγραφών από το τελικό σύνολο αποτελεσμάτων. Η επιλογή πραγματοποιείται με στρωματοποιημένη δειγματοληψία (stratified sampling), ώστε να εκπροσωπούνται ισορροπημένα τόσο οι επιτυχημένες αντιστοιχίσεις που προτείνονται για αγορά, όσο και οι επιτυχημένες αντιστοιχίσεις που δεν προτείνονται για αγορά, καθώς και οι περιπτώσεις όπου το σύστημα απέρριψε την αντιστοίχιση.

Για κάθε επιλεγμένη εγγραφή αποστέλλονται στο τοπικό μοντέλο Llama μέσω Ollama τα βασικά στοιχεία της αντιστοίχισης, όπως η περιγραφή του προϊόντος του προμηθευτή, η αντίστοιχη περιγραφή της βάσης δεδομένων, το ProductID, η τιμή του προμηθευτή, η τιμή της επιχείρησης, καθώς και οι αποφάσεις Final Match και BuyOrNot που παρήγαγε το σύστημα.

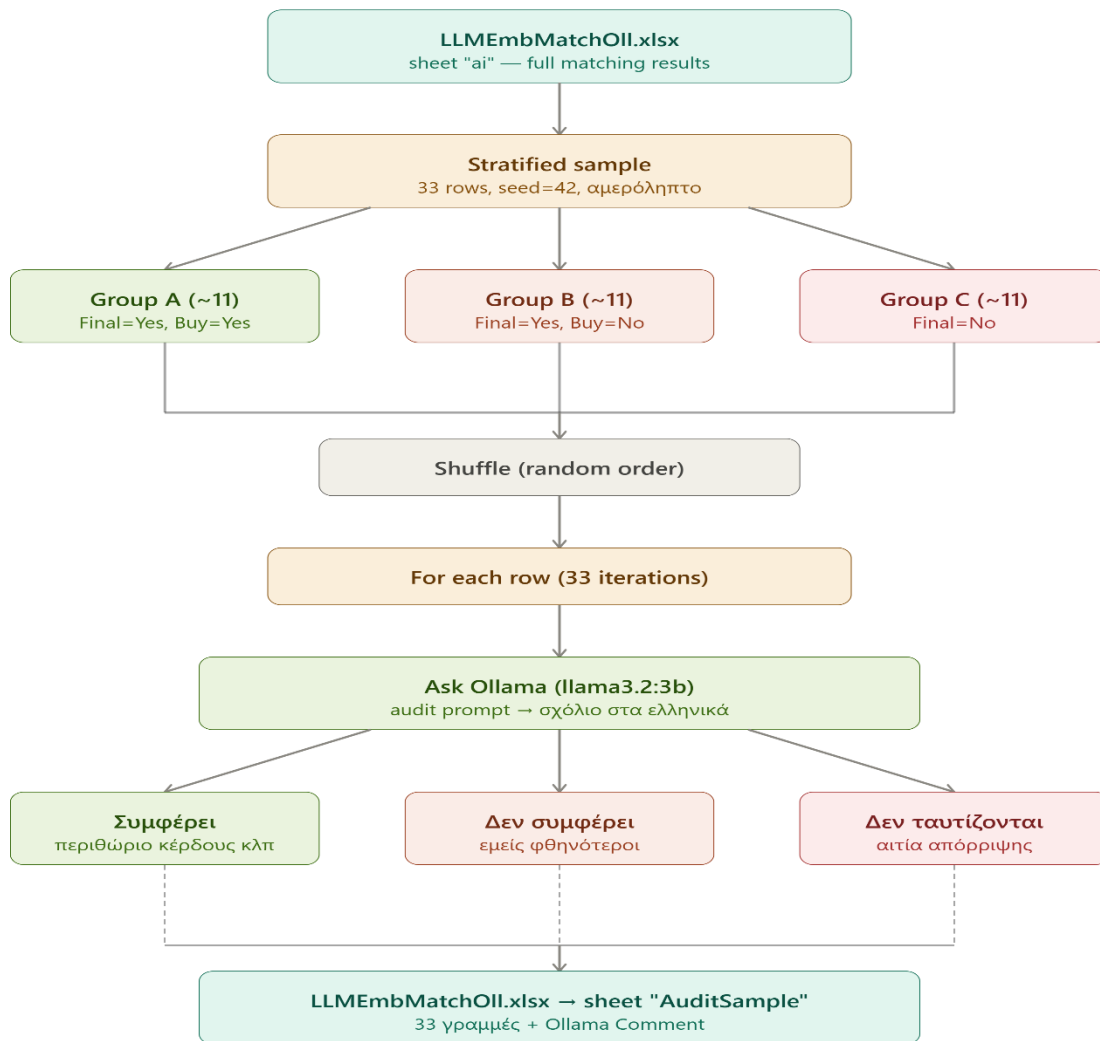
Το LLM δεν εκτελεί νέα αντιστοίχιση ούτε αντικαθιστά τον αλγόριθμο matching. Αντίθετα, λειτουργεί ως μηχανισμός ποιοτικής αξιολόγησης (audit), εξετάζοντας εάν η τελική απόφαση είναι λογικά τεκμηριωμένη σύμφωνα με τα διαθέσιμα στοιχεία. Στις περιπτώσεις όπου η αντιστοίχιση είναι επιτυχής, αξιολογεί εάν τα δύο προϊόντα φαίνεται να αναφέρονται στο ίδιο αντικείμενο και σχολιάζει συνοπτικά κατά πόσο η προτεινόμενη αγορά είναι εμπορικά συμφέρουσα με βάση τη διαφορά τιμής. Αντίστοιχα, όταν η αντιστοίχιση έχει απορριφθεί, αιτιολογεί συνοπτικά τους πιθανότερους λόγους, όπως διαφορετικό μοντέλο, διαφορετικά τεχνικά χαρακτηριστικά ή ασυμφωνία βασικών προδιαγραφών.

Τα αποτελέσματα του ελέγχου αποθηκεύονται αυτόματα σε νέο φύλλο (AuditSample) του ίδιου αρχείου Excel, χωρίς να μεταβάλλονται τα αρχικά αποτελέσματα της εφαρμογής. Για κάθε εγγραφή καταγράφονται η περιγραφή του προϊόντος του προμηθευτή, το αντίστοιχο προϊόν της βάσης δεδομένων, το ProductID, οι δύο τιμές, οι αποφάσεις Final Match και BuyOrNot, καθώς και ένα επεξηγηματικό σχόλιο που παράγεται από το τοπικό LLM.

Η διαδικασία αυτή δεν χρησιμοποιείται για την αυτόματη διόρθωση των αποτελεσμάτων, αλλά ως πρόσθετος μηχανισμός ποιοτικής αξιολόγησης και επαλήθευσης της λειτουργίας του συστήματος. Παράλληλα, μπορεί να αποτελέσει χρήσιμο εργαλείο για τον αναλυτή ή το εμπορικό τμήμα της επιχείρησης, καθώς παρέχει συνοπτική αιτιολόγηση των αποφάσεων και διευκολύνει τον έλεγχο των αποτελεσμάτων σε μεγάλα σύνολα δεδομένων.

Πλέον εδώ προχωρήσαμε σε αρχιτεκτονική 4 επιπέδων:

1. Data Layer (XML – SQL – Excel).
2. Matching Layer (Embeddings + Cosine + Numeric Rules + Ollama).
3. Business Layer (Price Comparison, BuyOrNot).
4. Audit Layer (τυχαία δειγματοληψία και ποιοτικός έλεγχος από το τοπικό LLM).



```

Total rows: 7516
Sample size: 33 rows
[1/33] Auditing: usb3.0 *4-port 4K HDMI Dual Display KVM Switch (CS1844)...
[2/33] Auditing: Asus Prime X870-P WiFi1 / AMD / Socket AM5 / ATX / 4xDDR5 / 2...
[3/33] Auditing: Honor Magic5 Lite 5G, 6GB ram/128GB, Dual SIM, Emerald Green...
[4/33] Auditing: Digitus Modular Console With 17-Inch TFT & Touchpad, Without...
[5/33] Auditing: Dell P2425 / 1500:1 / VGA / Sms / 24-Inch WUXGA IPS Monitor ...
[6/33] Auditing: Gigabyte Z790 Aorus Pro X WiFi7, Intel, Socket 1700, ATX, 4x...
[7/33] Auditing: TP-Link 4G+ Cat6 AC1200 Wireless Dual Band Gigabit Router - ...
[8/33] Auditing: geforce gigabyte RTX 4060 Ti Eagle OC 8GB GDDR6 - 128-Bit - ...
[9/33] Auditing: biz jabra 2300 Duo QD NC Headset (2309-820-104)...
[10/33] Auditing: Asus ProArt B760-Creator D4, Intel, Socket 1700, ATX, 4xDDR4...
[11/33] Auditing: Canon 067H Toner Cartridge - 2350 Pages - Cyan (S105C002AA)...
[12/33] Auditing: Samsung Galaxy Tab A9 LTE 4G LTE-TDD & LTE-FDD 128 GB 22.1 c...
[13/33] Auditing: Corsair VENGEANCE RGB Pro 32GB (2 x 16GB) DDR4 DRAM 3600MHz ...
[14/33] Auditing: Wacom Intuos M Comfort / Black (CTL-6100WLK-N) CTL-6100WLK-S...
[15/33] Auditing: t-force teamgroup Vulcan 32GB Kit (2x16GB) 5600MHz UDIMM DDR...
[16/33] Auditing: Apple iPhone 15 Plus 5G - 6GB ram/128GB - NFC - Blue (EU) (1...
[17/33] Auditing: b650i asrock Lightning WiFi1 - AMD - RAID - Mini-ITX - USB3.2...
[18/33] Auditing: triple asus Display USB-C Dock DC300 - Black (90XB08CN-BDS01...
[19/33] Auditing: radeon sapphire Pulse RX 7900 XT 20GB ram GDDR6 - 320-Bit - ...
[20/33] Auditing: radeon asrock RX 7900 XT Phantom Gaming 24GB GDDR6 OC, 384...
[21/33] Auditing: DDR4 64GB ram KIT 2x32GB PC 3200 Teamgroup T-Create Expert 0...
[22/33] Auditing: viper patriot Venom RGB ram 32GB Kit (2x16GB) 6000MHz UDIMM ...
[23/33] Auditing: CORSAIR iCUE 5000T RGB ram - MDT - extended ATX...
[24/33] Auditing: galaxy samsung SM-X716 Tab S9, 11-Inch/8GB ram/128GB/5G/Andr...
[25/33] Auditing: rog asus Rapture GT-AX6000 Dual-Band WiFi 6 (802.11ax) Gamin...
[26/33] Auditing: G.Skill Ripjaws V 64GB ram Kit (2x32GB) 4000MHz UDIMM DDR4 C...
[27/33] Auditing: thinkcentre lenovo Neo 50a 27 Gen 5 A10 - 17-13620H/27 FHD I...
[28/33] Auditing: Xiaomi Redmi 14C 4G - 4GB ram/128GB - Dual SIM - NFC - Midni...
[29/33] Auditing: ARCTIC Liquid Freezer III 420 A-RGB ram White | A10-Wasserkf...
[30/33] Auditing: Be quiet! Dark Rock Pro 5 CPU Cooling Unit, 2x135mm Fans, BL...
[31/33] Auditing: z790 gigabyte Aorus Master X - Intel - Socket 1700 - E-ATX -...
[32/33] Auditing: imac apple A10 / 15-7360U/21.5 FHD IPS/8GB/256B SSD/Webcam/W...
[33/33] Auditing: TCL T509K1 505 4G / Dual SIM / 4GB ram/64GB / Space Gray (EU...

```

Done - sheet 'AuditSample' written to C:\Users\User\Desktop\PY\LLMEmbMatchOll.xlsx
Rows audited: 33

Παράδειγμα:

Supplier Product	Our Product	Our ID	Our Price	Supplier Price	Final Match	BuyOrNot	Ollama Comment
Asus ProArt B760-Creator D4, Intel, Socket 1700, ATX, 4xDDR4, 4xSATA3, M.2, Raid, 2.5GLAN+GLAN, USB3.2, HDMI, DP (90MB1DU0-M0EAY0)	Asus ProArt B760-Creator D4, Intel, Socket 1700, ATX, 4xDDR4, 4xSATA3, M.2, Raid, 2.5GLAN+GLAN, USB3.2, HDMI, DP (90MB1DU0-M0EAY0)	119473	213,77	201,95	Yes	Yes	The product matching result indicates a perfect match between the supplier's and our product specifications. Since Final Match = Yes and BuyOrNot = Yes, buying from the supplier is beneficial due to a price difference of \$12.82 (our price - supplier price), resulting in a higher profit margin for us.
t-force teamgroup Vulcan 32GB Kit (2x16GB) 5600MHz UDIMM DDR5 CL36 1.2V, Red (FLRD532G5600HC36BDC01)	TeamGroup T-Force Vulcan 32GB Kit (2x16GB) 5600MHz UDIMM DDR5 CL36 1.2V, Red (FLRD532G5600HC36BDC01)	119031	91,07	95,15	Yes	No	The product matching result indicates a close match between the supplier's and our product, but with a price difference of \$4.08. Since Final Match = Yes, we should confirm that these are indeed identical products with no significant differences. The slight price difference may be due to various factors such as shipping costs or vendor fees.
.....							

galaxy samsung SM-X716 Tab S9, 11-Inch/8GB ram/128GB/5G/Android, Beige (8806095066394)	Samsung Galaxy SM-X716 Tab S9, 11- Inch/12GB/256GB/5G/Android, Beige (8806095066370)	129926	672,42	724,35	No	No	The two products do not match due to the difference in RAM specifications (8GB vs 12GB). The system decision is correct as they are not identical products. Further verification is required to confirm if these are indeed the same product variant.
---	---	--------	--------	--------	----	----	---

Στο παράδειγμα μας εμφανίζονται το προϊόν του προμηθευτή, το αντίστοιχο προϊόν της βάσης δεδομένων, το ProductID, οι δύο τιμές, η τελική απόφαση του συστήματος Final Match, η επιχειρησιακή απόφαση αγοράς BuyOrNot και το επεξηγηματικό σχόλιο που παρήγαγε το τοπικό μας μοντέλο LLM: Llama.

Στο πρώτο παράδειγμα οι δύο περιγραφές είναι απολύτως ταυτόσημες, γεγονός που οδήγησε το σύστημα σε επιτυχή αντιστοίχιση (Final Match = Yes). Παράλληλα, η τιμή του προμηθευτή 201,95€ είναι χαμηλότερη από την αντίστοιχη τιμή της επιχείρησης 213,77€, με αποτέλεσμα το σύστημα να προτείνει την αγορά του προϊόντος (BuyOrNot = Yes). Το σχόλιο του LLM επιβεβαιώνει ότι πρόκειται για το ίδιο προϊόν και αιτιολογεί την απόφαση αγοράς βάσει της οικονομικής διαφοράς μεταξύ των δύο τιμών.

Στο δεύτερο παράδειγμα το σύστημα αναγνώρισε επίσης ότι οι δύο περιγραφές αντιστοιχούν στο ίδιο προϊόν (Final Match = Yes). Ωστόσο, η τιμή του προμηθευτή 95,15€ είναι υψηλότερη από την αντίστοιχη τιμή της επιχείρησης 91,07€, με αποτέλεσμα η τελική ένδειξη αγοράς να είναι αρνητική (BuyOrNot = No). Το παραγόμενο σχόλιο του LLM αναγνωρίζει ότι πρόκειται για την ίδια μνήμη RAM και επισημαίνει ότι η μικρή διαφορά τιμής δεν δικαιολογεί την πραγματοποίηση αγοράς, ενώ αναφέρει ενδεικτικά ότι πιθανές αποκλίσεις μπορεί να οφείλονται σε πρόσθετα κόστη, όπως μεταφορικά ή διαφορετική εμπορική πολιτική του προμηθευτή.

Στο τρίτο παράδειγμα παρουσιάζεται μία περίπτωση απόρριψης αντιστοίχισης. Παρότι οι δύο περιγραφές αναφέρονται στην ίδια οικογένεια προϊόντων (Samsung Galaxy Tab S9), υπάρχουν ουσιώδεις διαφορές στα τεχνικά χαρακτηριστικά, καθώς η περιγραφή του προμηθευτή αφορά έκδοση με μνήμη 8 GB και αποθηκευτικό χώρο 128 GB, ενώ το προϊόν της βάσης δεδομένων αντιστοιχεί σε έκδοση με 12 GB μνήμη και 256 GB αποθηκευτικό χώρο. Για τον λόγο αυτό το σύστημα επέστρεψε Final Match = No και δεν προχώρησε σε εμπορική αξιολόγηση της προσφοράς. Το σχόλιο του LLM εντόπισε επίσης τη διαφορά στις τεχνικές προδιαγραφές και επιβεβαίωσε ότι η απόρριψη της αντιστοίχισης ήταν ορθή.

Τα παραπάνω παραδείγματα δείχνουν ότι ο μηχανισμός ποιοτικού ελέγχου δεν περιορίζεται στην επιβεβαίωση της αντιστοίχισης, αλλά εξετάζει και τη συνέπεια της επιχειρησιακής απόφασης που ακολουθεί.

Από τεχνικής πλευράς εδώ αξιοποιούνται πάλι οι βιβλιοθήκες pandas για την επεξεργασία των δεδομένων Excel, openpyxl για την ανάγνωση και ενημέρωση των φύλλων εργασίας, καθώς και το OpenAI Python SDK ως διεπαφή επικοινωνίας με το τοπικό μοντέλο Llama 3.2 μέσω του Ollama. Η εφαρμογή διαβάζει το αρχείο αποτελεσμάτων που παρήχθη από το προηγούμενο στάδιο της αντιστοίχισης, επιλέγει ένα τυχαίο στρωματοποιημένο δείγμα 33 εγγραφών και για κάθε εγγραφή καλεί τη συνάρτηση `ask_ollama_audit()`, η οποία δημιουργεί δυναμικά το κατάλληλο prompt και αποστέλλει τα δεδομένα στο τοπικό LLM. Στη συνέχεια, το παραγόμενο σχόλιο επιστρέφεται από το μοντέλο και αποθηκεύεται αυτόματα σε νέο φύλλο εργασίας AuditSample του ίδιου αρχείου Excel.

Το prompt που χρησιμοποιείται δεν ζητά από το LLM να πραγματοποιήσει νέα αντιστοίχιση προϊόντων αυτό έχει γίνει σε προηγούμενο στάδιο script, αλλά να αξιολογήσει την ήδη παραγόμενη απόφαση του συστήματος. Συγκεκριμένα, λαμβάνει ως είσοδο την περιγραφή του προϊόντος του προμηθευτή, την αντίστοιχη περιγραφή της βάσης δεδομένων, το αναγνωριστικό προϊόντος ProductID, την τιμή του προμηθευτή, την τιμή της επιχείρησης, καθώς και τις αποφάσεις Final Match και BuyOrNot. Με βάση αυτές τις πληροφορίες καλείται να συντάξει ένα σύντομο και αντικειμενικό σχόλιο, στο οποίο αιτιολογεί εάν η αντιστοίχιση θεωρείται ορθή και, όταν αυτή είναι επιτυχής, εάν η προτεινόμενη αγορά είναι εμπορικά συμφέρουσα. Αντίστοιχα, στις περιπτώσεις απόρριψης εξηγεί συνοπτικά τους βασικούς λόγους για τους οποίους τα δύο προϊόντα δεν θεωρούνται ταυτόσημα.

4.2.3 Μελλοντική Δυνατότητα αξιοποίησης Frameworks Ενорχήστρωσης LLM.

Στην παρούσα εργασία δεν χρησιμοποιήθηκαν σύγχρονα εξειδικευμένα frameworks ενорχήστρωσης εφαρμογών Τεχνητής Νοημοσύνης, όπως τα LangChain, LangGraph, AutoGen ή αντίστοιχες πλατφόρμες. Η επιλογή αυτή έγινε σκόπιμα, καθώς στόχος ήταν η ανάπτυξη και κατανόηση όλων των επιμέρους σταδίων της διαδικασίας αντιστοίχισης, από την ανάκτηση των δεδομένων και τη δημιουργία embeddings έως την εφαρμογή των επιχειρησιακών κανόνων και την επαλήθευση μέσω LLM. Πλέον όταν ξεκίνησε η υλοποίηση δεν τα γνώριζα και δεν είχαν ακόμα αναπτυχθεί.

Η υλοποίηση πραγματοποιήθηκε με χρήση Python και των απαραίτητων βιβλιοθηκών για κάθε επιμέρους λειτουργία (SQL Server, XML, Excel, Sentence Transformers, Ollama και OpenAI SDK), επιτρέποντας τον πλήρη έλεγχο της ροής εκτέλεσης, της λογικής λήψης αποφάσεων και της βελτιστοποίησης κάθε σταδίου του αλγορίθμου.

Παρόλα αυτά, σε μελλοντική επέκταση της εφαρμογής τα παραπάνω frameworks θα μπορούσαν να αξιοποιηθούν για την καλύτερη ενорχήστρωση πιο σύνθετων ροών εργασίας. Ενδεικτικά:

- το LangChain θα μπορούσε να οργανώσει τα επιμέρους στάδια του pipeline και τη διαχείριση prompts,
- το LangGraph να υποστηρίζει πολύπλοκες ροές με διαφορετικά μονοπάτια εκτέλεσης και επαναλήψεις,
- ενώ το AutoGen θα μπορούσε να υλοποιήσει συνεργασία μεταξύ πολλαπλών εξειδικευμένων AI agents, όπως agent αντιστοίχισης προϊόντων, agent οικονομικής αξιολόγησης και agent ποιοτικού ελέγχου.

Στην παρούσα εργασία, ωστόσο, η υπάρχουσα αρχιτεκτονική κρίθηκε επαρκής, καθώς καλύπτει πλήρως τις απαιτήσεις της εφαρμογής με απλούστερη σχεδίαση, μικρότερη πολυπλοκότητα, ευκολότερη συντήρηση, ιδιωτικότητα και κυρίως δωρεάν υλοποίηση ώστε να είναι υλοποιήσιμη από όλους, τουλάχιστον μέχρι και σήμερα που γράφηκε η Διπλωματική 3/7/2026.

4.2.4 Μελλοντική Επέκταση με μηχανισμό συνεχούς μάθησης.

Η εφαρμογή που αναπτύχθηκε παράγει αξιόπιστα αποτελέσματα αντιστοίχισης και παρέχει επιπλέον μηχανισμό ελέγχου LLM. Οι αποφάσεις που λαμβάνονται κατά τη διαδικασία του ελέγχου θα μπορούσαν να αξιοποιηθούν περαιτέρω για τη βελτίωση του συστήματος.

Μία μελλοντική επέκταση θα ήταν η υλοποίηση ενός μηχανισμού συνεχούς μάθησης. Ο χρήστης ή το εμπορικό τμήμα θα μπορούσε να επιβεβαιώνει ή να διορθώνει τις αντιστοιχίσεις που παράγει το σύστημα. Οι διορθώσεις αυτές θα αποθηκεύονταν σε ειδική βάση γνώσης ή εδώ απλά θα τροποποιούσε το αρχείο `xlsx` στήλη `final` που παρήχθη από το `Train_2_predict_ml_product_matcher` και θα μπορούσαν να χρησιμοποιηθούν για τη συνεχή βελτίωση των μελλοντικών αντιστοιχίσεων.

Παράδειγμα:

Rows loaded: 5

Rows after cosine filter (≥ 0.75): 5

New feedback file created

Saved: `feedback_dataset.csv`

Total feedback rows : 5

Label=1 (match) : 3

Label=0 (no match) : 2

Run `Train_4_retrain_xgb.py` when you have enough rows.

Output `feedback_dataset.csv`:

```
Import,MatchedProductID,MatchedName,CosineScore,FinalMatch,label,date,source
Apple iPhone 14 Pro / Android / 6.8" AMOLED / 128GB ram / Tensor G2,24,"Apple iPhone 13 | 128GB | 67."" amoled 120Hz |
A15 Bionic | Android",0.7939000129699707,No,0,2026-07-10,ML_prediction
Samsung Galaxy A5 16GB EU, Pearl White,51316,"Samsung Galaxy A5 16GB EU, Pearl White",1.0,Yes,1,2026-07-
10,ML_prediction
galaxy xiaomi S23 Ultra, 128GB, Tensor G2, Android, 6.1" OLED,26,"Google Galaxy S23 Ultra | 256GB | .61"" OLED | A16 Bionic |
Android",0.8215000033378601,No,0,2026-07-10,ML_prediction
galaxy samsung S22 | 512GB | 6.1" OLED | Tensor G2 | Android,27,"Samsung Galaxy S22 | 512GB | 6.1"" OLED | Tensor G2 |
Android",0.9987000226974487,Yes,1,2026-07-10,ML_prediction
Lenovo XPS 13 / Ryzen 5 5500U/15.6" FHD 144Hz/8GB/512GB SSD/AMD Radeon/Win10 Home,10,"Lenovo XPS 13, Ryzen 5
5500U/15.6"" FHD 144Hz/8GB/512GB SSD/AMD Radeon/Win10 Home",0.9911999702453613,Yes,1,2026-07-10,ML_prediction
```

Επιπλέον, τα δεδομένα αυτά θα μπορούσαν να αποτελέσουν νέο σύνολο εκπαίδευσης για μοντέλα μηχανικής μάθησης ή για επαναπροσδιορισμό των ορίων και των κανόνων που χρησιμοποιούνται κατά τη διαδικασία αντιστοίχισης. Με τον τρόπο αυτό το σύστημα θα αποκτούσε τη περαιτέρω δυνατότητες, βελτιώνοντας συνεχώς την ακρίβεια και την αξιοπιστία των αποτελεσμάτων του.

```
# Train_3_feedback_dataset.py
# Reads the output of Train_2_predict_ml_product_matcher.py (LLMEmbMatch_ML_matched.xlsx)
# Extracts confirmed decisions (Final Match = Yes/No) as gold labels
# Appends them to feedback_dataset.csv for future retraining
# Run AFTER Train_2_predict_ml_product_matcher.py
# Output: feedback_dataset.csv

import os
import pandas as pd
from datetime import datetime

# ===== CONFIG =====
INPUT_PATH = r"C:\Users\igIT\Desktop\ig\Edu\Thesis\0code\LLMEmbMatch_ML_matched.xlsx"
INPUT_SHEET = "MatchingResults"
FEEDBACK_PATH = "feedback_dataset.csv"
MIN_COSINE_SCORE = 0.75 # ignore very low confidence rows
# ===== MAIN =====

if __name__ == "__main__":
    # — read prediction output —
    print(f"Reading: {INPUT_PATH}")
    df = pd.read_excel(INPUT_PATH, sheet_name=INPUT_SHEET, engine="openpyxl")
    df.columns = df.columns.str.strip()
    print(f"Rows loaded: {len(df)}")

    # — validate columns —
    required = ["Import", "Matched Description", "CosineScore", "Final"]
    missing = [c for c in required if c not in df.columns]
    if missing:
        raise Exception(f"Missing columns: {missing}\nFound: {df.columns.tolist()}")

    # — filter: keep only rows with enough confidence —
    df = df[df["CosineScore"] ≥ MIN_COSINE_SCORE].copy()
    print(f"Rows after cosine filter (≥ {MIN_COSINE_SCORE}): {len(df)}")

    # — map Final → binary label —
    # Yes → 1 (same product), No → 0 (different product)
    df["label"] = df["Final"].str.strip().str.lower().map({"yes": 1, "no": 0})
    df = df.dropna(subset=["label"])
    df["label"] = df["label"].astype(int)

    # — add metadata —
    df["date"] = datetime.now().strftime("%Y-%m-%d")
    df["source"] = "ML_prediction"

    # — select columns for feedback dataset —
    feedback_cols = {
        "Import": "Import",
        "Matched ProductID": "MatchedProductID",
        "Matched Description": "MatchedName",
        "CosineScore": "CosineScore",
        "Final": "FinalMatch",
        "label": "label",
        "date": "date",
        "source": "source",
    }

    # keep only columns that exist
    available = {k: v for k, v in feedback_cols.items() if k in df.columns}
    df_out = df[list(available.keys())].rename(columns=available)

    # — append to existing feedback_dataset.csv —
    if os.path.exists(FEEDBACK_PATH):
        df_existing = pd.read_csv(FEEDBACK_PATH)
        print(f"Existing feedback rows: {len(df_existing)}")
        # avoid duplicates: same Import + MatchedName already in dataset
        merge_keys = ["Import", "MatchedName"]
        available_keys = [k for k in merge_keys if k in df_existing.columns and k in df_out.columns]
        if available_keys:
            df_new = df_out[~df_out.set_index(available_keys).index.isin(
                df_existing.set_index(available_keys).index
            )]
        else:
            df_new = df_out
        df_final = pd.concat([df_existing, df_new], ignore_index=True)
        print(f"New rows added : {len(df_new)}")
    else:
        df_final = df_out
        print(f"New feedback file created")

    # — save —
    df_final.to_csv(FEEDBACK_PATH, index=False, encoding="utf-8-sig")
    print(f"\n Saved: {FEEDBACK_PATH}")
    print(f" Total feedback rows : {len(df_final)}")
    print(f" Label=1 (match) : {(df_final['label']==1).sum()}")
    print(f" Label=0 (no match) : {(df_final['label']==0).sum()}")
    print(f"\nRun Train_4_retrain_xgb.py when you have enough rows")
```

Το παραπάνω script: `Train_3_feedback_dataset.py` αποτελεί το στάδιο δημιουργίας του `feedback dataset` στο υβριδικό σύστημα αντιστοίχισης προϊόντων. Διαβάζει τα αποτελέσματα πρόβλεψης του `Train_2_predict_ml_product_matcher.py` από το αρχείο `LLMEmbMatch_ML_matched.xlsx`, sheet: `MatchingResults`, ελέγχει ότι υπάρχουν οι απαραίτητες πληροφορίες και φιλτράρει μόνο τις αντιστοιχίσεις με υψηλή σημασιολογική ομοιότητα μέσω του `Cosine Similarity threshold ( $\geq 0.75$ )`, ώστε να αποφεύγονται χαμηλής ποιότητας δεδομένα εκπαίδευσης. Στη συνέχεια μετατρέπει τις τελικές αποφάσεις (Yes/No) σε δυαδικά machine learning labels (1/0), όπου το 1 δηλώνει ότι δύο προϊόντα θεωρούνται ίδια και το 0 ότι είναι διαφορετικά. Προσθέτει επιπλέον metadata όπως ημερομηνία δημιουργίας και πηγή δεδομένων, επιλέγει τις απαραίτητες στήλες και δημιουργεί ένα δομημένο dataset εκπαίδευσης. Πριν την αποθήκευση ελέγχει αν υπάρχουν ήδη καταχωρήσεις στο `feedback_dataset.csv` και αποφεύγει την εισαγωγή διπλότυπων εγγραφών, διατηρώντας έτσι ένα καθαρό ιστορικό δεδομένων. Το τελικό αρχείο χρησιμοποιείται στο επόμενο στάδιο παρακάτω (`Train_4_retrain_xgb.py`) για επανεκπαίδευση του Machine Learning μοντέλου, δημιουργώντας έναν μηχανισμό continuous learning feedback loop, όπου οι επιβεβαιωμένες αντιστοιχίσεις βελτιώνουν σταδιακά την ακρίβεια του συστήματος.

Το `Train_4_retrain_xgb.py` αποτελεί το στάδιο επανεκπαίδευσης του Machine Learning μοντέλου στο σύστημα αντιστοίχισης προϊόντων. Συνδυάζει τα αρχικά δεδομένα εκπαίδευσης με το νέο `feedback_dataset.csv`, το οποίο περιέχει τις επιβεβαιωμένες αντιστοιχίσεις προϊόντων, και δημιουργεί νέα χαρακτηριστικά βασισμένα στα embeddings, στο `Cosine Similarity`, στη διαφορά μήκους κειμένου και στην επικάλυψη κοινών λέξεων. Στη συνέχεια εκπαιδεύει ένα νέο μοντέλο XGBoost, αξιολογεί την απόδοσή του μέσω classification metrics και αποθηκεύει τη νέα έκδοση του μοντέλου σε ξεχωριστό αρχείο .pkl. Τα παλιά μοντέλα δεν τροποποιούνται, καθώς εφαρμόζεται μηχανισμός model versioning, όπου κάθε νέο retraining δημιουργεί νέο αρχείο με timestamp. Έτσι διατηρείται ιστορικό εκδόσεων, υπάρχει δυνατότητα σύγκρισης αποτελεσμάτων και αποφεύγεται η αντικατάσταση ενός λειτουργικού μοντέλου από πιθανώς χειρότερη έκδοση. Για να χρησιμοποιηθεί το νέο μοντέλο, γίνεται χειροκίνητη αλλαγή του `MODEL_PATH` στο στάδιο πρόβλεψης (`Train_2_predict_ml_product_matcher.py`). Με αυτόν τον τρόπο ολοκληρώνεται ο κύκλος continuous learning, όπου το σύστημα βελτιώνεται σταδιακά από τις νέες επιβεβαιωμένες αντιστοιχίσεις.

BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2

embeddings.position_ids | UNEXPECTED | |

Notes: - UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.

Feedback rows : 5

label=1 (match): 3

label=0 (no) : 2

WARNING: Only 5 feedback rows.

Recommended minimum is 1000 for reliable retraining.

Results may not improve significantly.

Continue anyway? (y/n): y

Loading original training data...

Original rows : 2444

Building pairs from original data...

Embedding original: 100% ██████████ 2444/2444 [00:00<00:00, 1907495.16it/s]

```

Train_4_retrain_xgb.py
# Combines original training data + feedback_dataset.csv
# Retrains XGBoost and saves a NEW versioned .pkl - never overwrites old ones.
# To use the new model: change MODEL_PATH in Train_2_predict_ml_product_matcher.py

import os
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import joblib
from datetime import datetime
from tqdm import tqdm
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score
from sentence_transformers import SentenceTransformer
from xgboost import XGBClassifier

# ===== CONFIG =====
ORIGINAL_PATH = r"C:\Users\ig1t\Desktop\ig\edu\Thesis\0code\0LLMEmbPrompt3Learn_augmented3.xlsx"
ORIGINAL_SHEET = "orig"
FEEDBACK_PATH = "feedback_dataset.csv"
EMB_CACHE_PATH = "embedding_cache_ml.pkl"
# - NEVER overwritten - always creates a new file with timestamp
# e.g. product_matcher_xgb_v2_20200119_143052.pkl
TIMESTAMP = datetime.now().strftime("%Y%m%d_%H%M%S")
NEW_MODEL_PATH = f"product_matcher_xgb_v2_{TIMESTAMP}.pkl"
RECOMMENDED_MIN = 1000
ABSOLUTE_MIN = 5
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')

# ===== CACHE =====
def load_cache(path):
    return joblib.load(path) if os.path.exists(path) else {}
def save_cache(cache, path):
    joblib.dump(cache, path)
# ===== EMBEDDINGS =====
def get_embedding(text, cache):
    text = str(text)
    if text in cache:
        return cache[text]
    emb = embedding_model.encode(text)
    cache[text] = emb
    return emb
# ===== FEATURES =====
def build_features(text1, text2, emb1, emb2):
    cos_sim = cosine_similarity([emb1], [emb2])[0][0]
    len_diff = abs(len(text1) - len(text2))
    set1 = set(text1.lower().split())
    set2 = set(text2.lower().split())
    union = len(set1.union(set2))
    overlap = len(set1.intersection(set2)) / union if union > 0 else 0
    return [cos_sim, len_diff, overlap]
# ===== PAIRS FROM EXCEL =====
def build_pairs_from_excel(df, cache):
    pairs, labels = [], []
    texts = df[['import']].astype(str).tolist()
    pids = df[['product_id']].astype(int).tolist()
    embeddings = [get_embedding(t, cache) for t in tqdm(texts, desc="Embedding original")]
    for i in range(len(texts)):
        for j in range(len(texts)):
            if i == j:
                continue
            f = build_features(texts[i], texts[j], embeddings[i], embeddings[j])
            pairs.append(f)
            labels.append(1 if pids[i] == pids[j] else 0)
    return np.array(pairs), np.array(labels)
# ===== PAIRS FROM FEEDBACK =====
def build_pairs_from_feedback(df_fb, cache):
    pairs, labels = [], []
    for _, row in tqdm(df_fb.iterrows(), total=len(df_fb), desc="Embedding feedback"):
        t1 = str(row.get('import', ''))
        t2 = str(row.get('matched_name', ''))
        label = int(row.get('label', 0))
        if not t1 or not t2:
            continue
        emb1 = get_embedding(t1, cache)
        emb2 = get_embedding(t2, cache)
        pairs.append(build_features(t1, t2, emb1, emb2))
        labels.append(label)
    return np.array(pairs), np.array(labels)
# ===== MAIN =====
if __name__ == '__main__':
    cache = load_cache(EMB_CACHE_PATH)
    # - check feedback -
    if not os.path.exists(FEEDBACK_PATH):
        raise FileNotFoundError(
            f"✗ (FEEDBACK_PATH) not found.\n"
            "Run Train_3_feedback_dataset.py first."
        )
    df_fb = pd.read_csv(FEEDBACK_PATH)
    print(f"Feedback rows : {len(df_fb)}")
    print(f"Label=1 (match): {(df_fb['label']==1).sum()}")
    print(f"Label=0 (no): {(df_fb['label']==0).sum()}")
    if len(df_fb) < ABSOLUTE_MIN:
        raise ValueError(
            f"✗ Only {len(df_fb)} feedback rows - minimum is {ABSOLUTE_MIN}.\n"
            "Collect more predictions first."
        )
    if len(df_fb) < RECOMMENDED_MIN:
        print(f"\n⚠ WARNING: Only {len(df_fb)} feedback rows.")
        print(f"Recommended minimum is {RECOMMENDED_MIN} for reliable retraining.")
        print(f"Results may not improve significantly.\n")
        ans = input("Continue anyway? (y/n): ").strip().lower()
        if ans == 'y':
            print("Aborted.")
            exit()
    # - original data -
    print(f"\nLoading original training data...")
    df_orig = pd.read_excel(ORIGINAL_PATH, sheet_name=ORIGINAL_SHEET)
    df_orig.columns = df_orig.columns.str.strip()
    df_orig = df_orig.dropna(subset=[f'import', f'product_id'])
    print(f"Original rows : {len(df_orig)}")
    # - build features -
    print(f"\nBuilding pairs from original data...")
    X_orig, y_orig = build_pairs_from_excel(df_orig, cache)
    print(f"\nBuilding pairs from feedback...")
    X_fb, y_fb = build_pairs_from_feedback(df_fb, cache)
    save_cache(cache, EMB_CACHE_PATH)
    # - combine -
    X = np.vstack([X_orig, X_fb])
    y = np.concatenate([y_orig, y_fb])
    print(f"\nCombined dataset : {X.shape[0]} pairs")
    print(f"Label=1 : {(y==1).sum()}")
    print(f"Label=0 : {(y==0).sum()}")
    # - split -
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )
    neg = np.sum(y_train == 0)
    pos = np.sum(y_train == 1)
    spw = neg / pos if pos > 0 else 1.0
    print(f"\nTask balance : neg={neg}, pos={pos}, ratio={spw:.1f}")
    # - train -
    print(f"\nTraining XGBoost v2...")
    model = XGBClassifier(
        n_estimators=500,
        max_depth=6,
        learning_rate=0.1,
        scale_pos_weight=spw,
        eval_metric='logloss'
    )
    model.fit(X_train, y_train)
    # - evaluate -
    y_pred = model.predict(X_test)
    print(f"\n==== Classification Report ===")
    print(classification_report(y_test, y_pred))
    f1_new = f1_score(y_test, y_pred, average='macro')
    print(f"F1 score: {f1_new:.4f}")
    # - save NEW file - never overwrites old -
    joblib.dump(model, NEW_MODEL_PATH)
    print(f"\nNew model saved : {NEW_MODEL_PATH}")
    print(f"MODEL_PATH = {NEW_MODEL_PATH}")
    print(f"\nTo use it, change MODEL_PATH in Train_2_predict_ml_product_matcher.py")
    # - list all existing models -
    existing = sorted([f for f in os.listdir(".") if f.startswith("product_matcher_xgb")])
    print(f"\nAll available models ({len(existing)}):")
    for m in existing:
        print(f"  {m}")

```

Συνοπτικά:

Train_1: Train_productid_ML.py -> δημιουργεί product_matcher_xgb.pkl -> όπου το χρησιμοποιεί το Train_2: Train_2_predict_ml.py Όπου δημιουργεί -> LLMEmbMatch_ML_matched.xlsx (Αρχείο με Αποτελέσματα) -> που τροφοδοτεί

Train_3: Train_3_feedback_dataset.py (νέα επέκταση) -> δημιουργεί/εμπλουτίζει

feedback_dataset.csv και μετά -> Train_4: Train_4_retrain_xgb.py -> όπου δημιουργεί νέο product_matcher_xgb_vxxx.pkl το οποίο μπορεί να χρησιμοποιηθεί στην επανεκπαίδευση του μοντέλου.

4.2.5 Πλέον Μελλοντικές Επεκτάσεις – Υλοποιήσεις.

Μια σημαντική μελλοντική επέκταση του συστήματος θα ήταν η μετάβαση από την τρέχουσα αποθήκευση embeddings σε αρχεία .pkl μέσω joblib σε μια εξειδικευμένη Vector Database, όπως το Qdrant ή το Milvus. Στην παρούσα υλοποίηση η αναζήτηση πραγματοποιείται με brute-force σύγκριση του embedding κάθε εισερχόμενου προϊόντος με όλα τα διαθέσιμα embeddings της βάσης, με πολυπλοκότητα $O(n)$. Η προσέγγιση αυτή είναι επαρκής για το μέγεθος των δεδομένων της εδώ εργασίας αλλά και για το πλήθος των περισσότερων επιχειρήσεων χιλιάδες δεκάδες lines, ίσως όμως δεν θα ήταν τόσο αποδοτική σε περιβάλλοντα με εκατομμύρια γραμμές – προϊόντα όπως σε δικτυακές πλατφόρμες οι οποίες όμως δεν αποτελούν αντικείμενο της παρούσας Διπλωματικής. Η χρήση Vector Database θα επέτρεπε την εφαρμογή αλγορίθμων Approximate Nearest Neighbor (ANN), όπως ο HNSW, μειώνοντας σημαντικά τον χρόνο αναζήτησης και επιτρέποντας κλιμάκωση σε επιχειρησιακά datasets. Η αλλαγή αυτή θα αφορούσε κυρίως το επίπεδο αναζήτησης των κοντινότερων embeddings, χωρίς να απαιτείται αλλαγή στην υπόλοιπη αρχιτεκτονική του pipeline.

Επιπλέον, μια σημαντική βελτίωση θα μπορούσε ίσως να πραγματοποιηθεί μέσω fine-tuning εξειδικευμένων μοντέλων με χρήση LoRA (Low-Rank Adaptation). Η παρούσα υλοποίηση χρησιμοποιεί γενικά προεκπαιδευμένα μοντέλα, όπως το all-MiniLM-L6-v2 για τη δημιουργία embeddings και LLM μοντέλα όπως Llama 3 ή Gemini για την τελική επαλήθευση. Παρότι τα μοντέλα αυτά έχουν υψηλές δυνατότητες κατανόησης φυσικής γλώσσας, δεν έχουν εκπαιδευτεί ειδικά στη γλώσσα των τεχνικών χαρακτηριστικών προϊόντων ηλεκτρονικού εμπορίου. Μέσω fine-tuning σε πραγματικά δεδομένα αντιστοίχισης προϊόντων, το μοντέλο θα μπορούσε να αποκτήσει εξειδικευμένη γνώση σχετικά με τεχνικές συντομογραφίες και χαρακτηριστικά, όπως DS (Dual SIM), 6+128GB, AMOLED ή διαφορές σε μνήμη RAM και αποθηκευτικό χώρο. Το πλεονέκτημα της τεχνικής LoRA είναι ότι δεν απαιτεί πλήρη επανεκπαίδευση του μοντέλου, αλλά προσαρμόζει μόνο ένα μικρό ποσοστό των παραμέτρων, μειώνοντας σημαντικά τις απαιτήσεις σε υπολογιστικούς πόρους. Δεν λειτουργεί όμως τόσο αποτελεσματικά στην πράξη σε Ελληνικές επιχειρήσεις που έχουν έναν δικό τους τρόπο και business γενικότερα πάλι θα δούλευε θεωρώ σε δικτυακές παγκόσμιες πλατφόρμες.

Τέλος, για χρήση του συστήματος σε πραγματικό επιχειρησιακό περιβάλλον θα μπορούσε να εφαρμοστεί και containerization και orchestration μέσω Docker και Kubernetes. Η τρέχουσα υλοποίηση εκτελείται σε τοπικό περιβάλλον με εξάρτηση από συγκεκριμένες εγκαταστάσεις Python, βιβλιοθηκών και Ollama. Μέσω Docker, ολόκληρο το σύστημα θα μπορούσε να συσκευαστεί σε ανεξάρτητα containers, τα οποία περιλαμβάνουν τον κώδικα, τις βιβλιοθήκες και το περιβάλλον εκτέλεσης. Μια ενδεικτική αρχιτεκτονική θα αποτελείται από ένα container για το κύριο matching pipeline και ένα δεύτερο container για το τοπικό

LLM μέσω Ollama, με επικοινωνία μέσω εσωτερικού δικτύου Docker. Με αυτό τον τρόπο το σύστημα θα μπορούσε να εκτελείται με τον ίδιο τρόπο σε διαφορετικές υποδομές χωρίς εξάρτηση από συγκεκριμένο υπολογιστή. Σε μεγαλύτερη κλίμακα, η χρήση Kubernetes θα επέτρεπε δυναμική διαχείριση πολλαπλών instances του matcher, αυτόματη επανεκκίνηση σε περίπτωση αποτυχίας και οριζόντια κλιμάκωση ανάλογα με τον φόρτο εργασίας. Συνεπώς, οι τεχνολογίες Docker και Kubernetes αποτελούν φυσική επέκταση της παρούσας αρχιτεκτονικής, μετατρέποντας ένα ερευνητικό prototype σε ένα πιο ώριμο επιχειρησιακό σύστημα υψηλής διαθεσιμότητας.

Πρακτικά το παραπάνω μπορεί να υλοποιηθεί και μέσω exe αρχείων similar ας πούμε του docker και bat καθώς και windows scheduler σαν Kubernetes ώστε να μην έχει κόστος μια μικρομεσαία επιχείρηση.

4.2.6 Βελτιστοποίηση με χρήση Qdrant Vector Database για Τεράστια Δεδομένα.

Στην παρούσα υλοποίηση πραγματοποιήθηκε βελτίωση της προηγούμενης έκδοσης του συστήματος αντιστοίχισης προϊόντων, αντικαθιστώντας τη διαδικασία αναζήτησης με χρήση απευθείας υπολογισμού cosine similarity σε ολόκληρο το σύνολο των προϊόντων με μια Vector Database αρχιτεκτονική βασισμένη στο Qdrant. Στην προηγούμενη έκδοση, για κάθε προϊόν προμηθευτή δημιουργούνταν embedding και πραγματοποιούνταν σύγκριση με όλα τα διαθέσιμα embeddings της βάσης δεδομένων, γεγονός που αύξανε το υπολογιστικό κόστος όσο μεγάλωνε ο αριθμός των προϊόντων. Στη νέα υλοποίηση, τα embeddings των προϊόντων της SQL βάσης αποθηκεύονται σε Qdrant collection μαζί με τα αντίστοιχα metadata (ProductID, περιγραφή προϊόντος και τιμή πώλησης), επιτρέποντας ταχύτερη αναζήτηση των πλησιέστερων διανυσμάτων μέσω Approximate Nearest Neighbor (ANN) search.

Σε επίπεδο κώδικα, η βασική λογική του συστήματος παρέμεινε ίδια. Διατηρήθηκαν η κανονικοποίηση κειμένου, ο αυστηρός διαχωρισμός αριθμητικών χαρακτηριστικών, το μοντέλο παραγωγής embeddings all-MiniLM-L6-v2, το τοπικό LLM μοντέλο Ollama για επιβεβαίωση αμφίβολων αντιστοιχίσεων, καθώς και οι επιχειρησιακοί κανόνες για RAM, αποθηκευτικό χώρο, CPU/GPU και αριθμητικές αποκλίσεις.

Η βασική αλλαγή αφορά την προσθήκη του QdrantClient, τη δημιουργία vector collection, την εισαγωγή των προϊόντων ως PointStruct objects και την αναζήτηση του καλύτερου αποτελέσματος μέσω της μεθόδου query_points αντί για τη διαδοχική σύγκριση όλων των embeddings με cosine_similarity. Επιπλέον, η νέα έκδοση αποθηκεύει τα αποτελέσματα σε ξεχωριστό αρχείο Excel (LLMEmbMatchOll_VectorDB.xlsx), ώστε να διατηρείται ανέπαφο το αρχικό αρχείο δεδομένων.

Η εκτέλεση της νέας αρχιτεκτονικής πραγματοποιήθηκε με σύνολο 11.611 προϊόντων από τη SQL βάση και 7.516 προϊόντα προμηθευτών από το αρχείο XML θα μπορούσε θεωρητικά και με Εκατομμύρια. Κατά το πρώτο στάδιο δημιουργήθηκε το vector index στο Qdrant με επιτυχία, όπου αποθηκεύτηκαν συνολικά 11.611 embeddings προϊόντων. Η διαδικασία τελικής αντιστοίχισης ολοκληρώθηκε σε χρόνο περίπου 7 λεπτών και 53 δευτερολέπτων, παράγοντας 7.516 αποτελέσματα. Από τα αποτελέσματα προέκυψαν 7.477 επιτυχείς αντιστοιχίσεις (Final Match = Yes) και 39 περιπτώσεις χωρίς αποδεκτή αντιστοίχιση (Final Match = No), επιβεβαιώνοντας ότι η χρήση Vector Database διατηρεί την ακρίβεια της προηγούμενης προσέγγισης ενώ παρέχει πιο επεκτάσιμη αρχιτεκτονική για μεγαλύτερα σύνολα προϊόντων.

Import	Ma tch ing ID	Matching Name	Em bed ding S co re	Olla ma Veri fica tion	Fi n al M at ch	Sup plier Pric e	Ou rPr ice	Pri ce Dif f	Bu yOr No t
Dell U4924DW - 49-Inch 5K Dual QHD IPS Curved Monitor - 5120x1440 - 32:9 - 5ms - 2000:1 - USB - HDMI - DP - Speakers - Black (U4924DW)	12 01 62	Dell U4924DW, 49-Inch 5K Dual QHD IPS Curved Monitor, 5120x1440, 32:9, 5ms, 2000:1, USB, HDMI, DP, Speakers, Black (U4924DW)	100	Yes	Yes	1162,85	1256.73	93,88	Yes
t7 samsung Shield 1TB Portable SSD - USB 3.2 Gen 2 - Black (MU-PE1T0S/EU)	12 01 91	Samsung T7 Shield 1TB Portable SSD, USB 3.2 Gen 2, Black (MU-PE1T0S/EU)	100	Yes	Yes	113,46	103.53	-9,93	No
Samsung SM-A146 Galaxy A14 5G / 4GB/64GB / Dual SIM / NFC / Silver (EU) (SM-A146PZSDEUE)(8806094825343)	12 02 18	Samsung SM-A146 Galaxy A14 5G, 4GB/64GB, Dual SIM, NFC, Silver (EU) (SM-A146PZSDEUE)(8806094825343)	100	Yes	Yes	129,02	132.05	3,03	Yes
Samsung SM-A146 Galaxy A14 5G - Black (EU) (SM-A146PZKDEUE)(8806094825374)(8806094825381) - 4GB ram/64GB - NFC - Dual SIM	12 02 19	Samsung SM-A146 Galaxy A14 5G, 4GB/128GB, Dual SIM, NFC, Black (EU) (SM-A146PZKGEUE)(8806094825350)(8706094825350)	97,85	No	No	132,36	155.59	23,23	Yes

Final Match	Καταμέτρηση από Matching ID
No	39
Yes	7477
Γενικό Άθροισμα	7516

```
# BDDVectorsLRBetterMatchingWithOllamaPR / MatchingWithOllama_VectorDB.py
# Same logic as MatchingWithOllama_Price.py BUT uses Qdrant Vector DB
# instead of brute-force cosine similarity.
# Output: LLMEmbMatchOll_VectorDB.xlsx (NEW file - original untouched)
# Requirements: pip install qdrant-client
# Run Qdrant locally: docker run -p 6333:6333 qdrant/qdrant

import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn")
import os
import re
import xml.etree.ElementTree as ET
import pyodbc
import pandas as pd
import numpy as np
import joblib
from tqdm import tqdm
from sentence_transformers import SentenceTransformer
from openai import OpenAI
from qdrant_client import QdrantClient
from qdrant_client.models import PointStruct, VectorParams, Distance

# ===== OLLAMA =====
client = OpenAI(
    base_url="http://127.0.0.1:11434/v1",
    api_key="ollama"
)
LLM_MODEL = "llama3.2:3b"

# ===== QDRANT =====
qdrant = QdrantClient(path="./qdrant_data") # qdrant = QdrantClient("localhost", port=6333) if i use Docker.
COLLECTION_NAME = "products"
VECTOR_SIZE = 384 # all-MiniLM-L6-v2 output size
# ===== EMBEDDING =====
embedding_model = SentenceTransformer("all-MiniLM-L6-v2") # or LORA_MODEL_PATH
CACHE_PATH = "embedding_cache_semantic.pkl"
SQL_FILE = r"C:\Users\User\Desktop\PY\SQLQuerybigData.sql"
EXCEL_PATH = r"C:\Users\User\Desktop\PY\LLMEmbMatchOll.xlsx"
# NEW output file - original untouched
OUTPUT_PATH = r"C:\Users\User\Desktop\PY\LLMEmbMatchOll_VectorDB.xlsx"
XML_PATH = r"C:\Users\User\Desktop\PY\SupplierProducts.xml"
SHEET_NAME = "ai"
# ===== SQL =====
conn_str = (
    "DRIVER={SQL Server};"
    "SERVER=DESKTOP-MNERR63;"
    "DATABASE=mag;"
    "Trusted_Connection=yes;"
)

def get_db_connection():
    return pyodbc.connect(conn_str)

def fetch_product_data():
    if not os.path.exists(SQL_FILE):
        raise FileNotFoundError(f"SQL file not found: {SQL_FILE}")
    with open(SQL_FILE, "r", encoding="utf-8") as f:
        sql_query = f.read()
    conn = get_db_connection()
    cursor = conn.cursor()
    print("Executing SQL...")
    cursor.execute(sql_query)
    while cursor.description is None:
        if not cursor.nextset():
            raise ValueError("SQL query did not return SELECT results")
    columns = [col[0] for col in cursor.description]
    rows = cursor.fetchall()
    conn.close()
    print(f"Rows fetched: {len(rows)}")
    return [dict(zip(columns, row)) for row in rows]

# ===== XML =====
def read_xml_prices(xml_path):
    if not os.path.exists(xml_path):
        print(f"⚠ XML not found - SupplierPrice will be empty")
        return {}
    tree = ET.parse(xml_path)
    root = tree.getroot()
    prices = {}
    for product in root.findall(".//product"):
        name = product.findtext("name")
        price = product.findtext("price")
        if name and price:
            try:
                prices[name.strip()] = float(price)
            except ValueError:
                pass
    print(f"XML prices loaded: {len(prices)}")
    return prices
```

```
# ===== CACHE =====
def load_embedding_cache():
    return joblib.load(CACHE_PATH) if os.path.exists(CACHE_PATH) else {}
def save_embedding_cache(cache):
    joblib.dump(cache, CACHE_PATH)
# ===== TOKENIZER =====
def normalize_text(text):
    text = str(text).lower()
    for ch in ["&", "/", "=", "(", ")", ".", " "]:
        text = text.replace(ch, " ")
    text = re.sub(r"\s+", " ", text).strip()
    return text
def split_alnum_tokens(text):
    text = normalize_text(text)
    return re.findall(r"[^\W\d_]+\d+", text, flags=re.UNICODE)
def token_sort_strict(text):
    tokens = split_alnum_tokens(text)
    num_tokens = [t for t in tokens if t.isdigit()]
    word_tokens = sorted(t for t in tokens if not t.isdigit())
    return " ".join(word_tokens + num_tokens)
def extract_numbers(text):
    return set(re.findall(r"\d+", normalize_text(text)))
def numeric_mismatch(a, b):
    return extract_numbers(a) != extract_numbers(b)
# ===== EMBEDDINGS =====
def get_embedding(text, cache):
    norm = token_sort_strict(text)
    if norm in cache:
        return cache[norm]
    emb = embedding_model.encode(norm)
    cache[norm] = emb
    return emb
# ===== QDRANT SETUP =====
def setup_qdrant_collection(product_data, cache):
    """Creates Qdrant collection and indexes all products."""
    # recreate collection (fresh index)
    if qdrant.collection_exists(COLLECTION_NAME):
        qdrant.delete_collection(COLLECTION_NAME)
    qdrant.create_collection(
        collection_name=COLLECTION_NAME,
        vectors_config=VectorParams(size=VECTOR_SIZE, distance=Distance.COSINE)
    )
    points = []
    for p in tqdm(product_data, desc="Indexing to Qdrant"):
        emb = get_embedding(p["Descr"], cache)
        points.append(PointStruct(
            id=int(p["ProductID"]),
            vector=emb.tolist(),
            payload={
                "name": p["Descr"],
                "price": p.get("PriceToSell")
            }
        ))
    # upload in batches of 100
    batch_size = 100
    for i in range(0, len(points), batch_size):
        qdrant.upsert(
            collection_name=COLLECTION_NAME,
            points=points[i:i+batch_size]
        )
    print(f"Qdrant indexed: {len(points)} products")
# ===== VECTOR SEARCH =====
def find_best_match_vector(import_emb, threshold=0.7):
    """ANN search via Qdrant - O(log n) instead of O(n)."""
    results = qdrant.query_points(
        collection_name=COLLECTION_NAME,
        query=import_emb.tolist(),
        limit=1
    ).points
    if not results:
        return None, 0.0
    best = results[0]
    score = best.score
    best_id = best.id
    name = best.payload.get("name", "")
    price = best.payload.get("price", None)
    if score < threshold:
        return None, score
    return {"id": best_id, "name": name, "price": price}, score
# ===== OLLAMA =====
def ask_ollama_yes_no(import_name, matched_name):
    prompt = """
You are a strict product matching system.
Decide if these two items are the same exact product.
Rules:
- Same brand and same model are required.
- Same RAM is required.
- Same storage is required.
- Same screen size is required.
- Same CPU/GPU is required when present.
- Different category means No.
- Different numbers usually mean No.
- Different punctuation, commas, slashes, dashes, capitalization, or word order do not matter.
Answer only: Yes or No
Import: {import_name}
Matched: {matched_name}
"""
    try:
        response = client.chat.completions.create(
            model=LLM_MODEL,
            messages=[{"role": "user", "content": prompt}],
            temperature=0.0,
            max_completion_tokens=10
        )
        reply = response.choices[0].message.content.strip().lower()
        return "Yes" if reply.startswith("yes") else "No"
    except Exception as e:
        print(f"OLLAMA error: {e}")
        return "Error"
```

```
# ===== MAIN =====
def process_supplier_matching():
    df = pd.read_excel(EXCEL_PATH, sheet_name=SHEET_NAME, engine="openpyxl")
    if "Import" not in df.columns:
        raise Exception("Import column not found")

    xml_prices = read_xml_prices(XML_PATH)
    supplier_price_col = [
        xml_prices.get(imp.strip(), None)
        for imp in df["Import"].fillna("").astype(str)
    ]
    cache = load_embedding_cache()
    product_data = fetch_product_data()

    # — index all products into Qdrant —
    setup_qdrant_collection(product_data, cache)
    save_embedding_cache(cache)
    import_texts = df["Import"].fillna("").astype(str).tolist()
    results = []
    for i, import_text in enumerate(tqdm(import_texts, desc="VectorDB Matching")):
        emb = get_embedding(import_text, cache)
        match, score = find_best_match_vector(emb)
        if match is None:
            results.append((None, "No Match", round(score * 100, 2),
                           "No", "No", None, None, None, None))
            continue
        matched_name = match["name"]
        our_price = match["price"]
        best_id = match["id"]
        # — triage (same logic as original) —
        if score < 0.80:
            verdict = "No"
            final = "No"
        elif score ≥ 0.95:
            if numeric_mismatch(import_text, matched_name):
                verdict = "No"
                final = "No"
            else:
                verdict = "Yes"
                final = "Yes"
        else:
            if extract_numbers(import_text) ≠ extract_numbers(matched_name):
                verdict = "No"
                final = "No"
            else:
                verdict = ask_ollama_yes_no(import_text, matched_name)
                final = "Yes" if verdict == "Yes" else "No"
        # — price logic —
        supplier_price = supplier_price_col[i]
        if supplier_price is not None and our_price is not None:
            try:
                diff = round(float(our_price) - float(supplier_price), 2)
                buy_or_not = "Yes" if diff > 0 else "No"
            except (ValueError, TypeError):
                diff = buy_or_not = None
        else:
            diff = buy_or_not = None
        results.append((
            best_id, matched_name, round(score * 100, 2),
            verdict, final,
            supplier_price, our_price, diff, buy_or_not
        ))
    result_df = pd.DataFrame(results, columns=[
        "Matching ID", "Matching Name", "Embedding Score",
        "Ollama Verification", "Final Match",
        "SupplierPrice", "OurPrice", "PriceDiff", "BuyOrNot"
    ])
    result_df.insert(0, "Import", df["Import"])
    # ← saves to NEW file — original LLMEmbMatchOll.xlsx untouched
    with pd.ExcelWriter(OUTPUT_PATH, engine="openpyxl") as writer:
        result_df.to_excel(writer, sheet_name=SHEET_NAME, index=False)
    save_embedding_cache(cache)
    print(f"\n Done — saved to: {OUTPUT_PATH}")
    print(" Original file untouched.")
    if __name__ == "__main__":
        process_supplier_matching()
```

4.3 Αξιολόγηση των Προτεινόμενων Συστημάτων.

Παρότι το προτεινόμενο σύστημα παρουσίασε ιδιαίτερα ικανοποιητικά αποτελέσματα, εξακολουθούν να υπάρχουν περιπτώσεις στις οποίες ιδιαίτερα σύνθετες ή ελλιπείς περιγραφές προϊόντων μπορούν να επηρεάσουν την τελική απόφαση αντιστοίχισης. Η απόδοση του συστήματος εξαρτάται επίσης από την ποιότητα των δεδομένων που εισάγονται, καθώς και από την πληρότητα των πληροφοριών που υπάρχουν στη βάση δεδομένων.

Στο μέλλον, το σύστημα μπορεί να επεκταθεί με νεότερα embedding models, εξειδικευμένα LLMs για το ηλεκτρονικό εμπόριο, τεχνικές reranking, vector databases και μηχανισμούς συνεχούς μάθησης από τις διορθώσεις των χρηστών. Επιπλέον, μπορεί να ενσωματωθεί πλήρως σε ERP ή e-commerce πλατφόρμες, ώστε η διαδικασία αντιστοίχισης να πραγματοποιείται αυτόματα κατά την εισαγωγή νέων προϊόντων, υποστηρίζοντας ακόμη μεγαλύτερους όγκους δεδομένων σε πραγματικό χρόνο.

Τέλος, θα πρέπει να επισημανθεί ότι ο χώρος της Τεχνητής Νοημοσύνης και ειδικότερα των Μεγάλων Γλωσσικών Μοντέλων εξελίσσεται με ιδιαίτερα ταχείς ρυθμούς. Τεχνικές και διαδικασίες που σήμερα απαιτούν συνδυασμό πολλαπλών εργαλείων, αλγορίθμων και επιχειρησιακών κανόνων ενδέχεται στο μέλλον να υλοποιούνται με σημαντικά απλούστερο τρόπο μέσω νέων βιβλιοθηκών, εξειδικευμένων μοντέλων ή ολοκληρωμένων platforms. Αντίστοιχα, είναι πιθανό να προκύψουν νέες απαιτήσεις και προκλήσεις που θα καταστήσουν αναγκαίες διαφορετικές προσεγγίσεις. Συνεπώς, η παρούσα εργασία δεν αποτελεί μια οριστική λύση στο πρόβλημα της αντιστοίχισης προϊόντων, αλλά μια ολοκληρωμένη πρόταση και έρευνα υλοποίησης βασισμένη στις διαθέσιμες τεχνολογίες και τις βέλτιστες πρακτικές της χρονικής περιόδου κατά την οποία εκπονήθηκε.

5. Βιβλιογραφία – Παραρτήματα:

Η βιβλιογραφία της παρούσας διπλωματικής εργασίας παρατίθεται στο σημείο όπου γίνεται η αντίστοιχη αναφορά μέσα στο κείμενο, ώστε κάθε πληροφορία, ορισμός, εικόνα ή επιστημονικό εύρημα να συνοδεύεται άμεσα από την αντίστοιχη πηγή του.

Οι εικόνες που χρησιμοποιήθηκαν στην εργασία συνοδεύονται από τη σχετική βιβλιογραφική αναφορά και τον σύνδεσμο της αρχικής πηγής στη σελίδα όπου εμφανίζονται. Παράδειγμα εικόνες που προέρχονται από διαδικτυακές δημοσιεύσεις ή επιστημονικά άρθρα αναγράφεται ο τίτλος της εικόνας και η αντίστοιχη πηγή, ενώ για επιστημονικές εργασίες χρησιμοποιείται η σχετική παραπομπή προς μαζί με τον τίτλο της δημοσίευσης.

Τα παραρτήματα της εργασίας περιλαμβάνουν τον πηγαίο κώδικα που αναπτύχθηκε στο πλαίσιο της υλοποίησης. Για λόγους αναγνωσιμότητας, τα τμήματα του κώδικα παρουσιάζονται ως εικόνες στα αντίστοιχα κεφάλαια όπου αναλύεται η λειτουργία τους, ενώ ο πλήρης κώδικας είναι στη διάθεσή σας.

Τα αρχιτεκτονικά διαγράμματα που παρουσιάζονται στην εργασία δημιουργήθηκαν με τη συνδρομή του μεγάλου γλωσσικού μοντέλου Claude, ενώ κατά τη διάρκεια της βιβλιογραφικής έρευνας, της τεχνικής τεκμηρίωσης και της συγγραφής αξιοποιήθηκαν επίσης τα εργαλεία ChatGPT και Perplexity ως βοηθητικά μέσα αναζήτησης, ανάλυσης και οργάνωσης της διαθέσιμης πληροφορίας.

Η τελική αξιολόγηση, ιδέα, επιλογή, σύνθεση του περιεχομένου, καθώς και η υλοποίηση με την πειραματική διαδικασία όπως και τα συμπεράσματα της εργασίας αποτελούν έργο και ιδιοκτησία του συγγραφέα Ηλία Γκόρου του Ναπολέοντα.

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.