**ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

**Τίτλος:**

# Ανάπτυξη Φωνητικού Bot για Αυτόματη Εξυπηρέτηση Πελατών μέσω Vicidial και Asterisk

**Κυριαζάκης Γεώργιος**
**Αριθμός Μητρώου: 329**

**Επιβλέπων**

**Βολογιαννίδης Σταύρος**

**ΣΕΡΡΕΣ - 2026**

# Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο του Προγράμματος Μεταπτυχιακών Σπουδών και πραγματεύεται τη σχεδίαση και υλοποίηση ενός ελληνόφωνου τηλεφωνικού συστήματος Voicebot βασισμένου σε τεχνολογίες τεχνητής νοημοσύνης και επεξεργασίας φυσικής ομιλίας. Στόχος της εργασίας είναι η ανάπτυξη μιας ολοκληρωμένης, πρακτικά εφαρμόσιμης λύσης αυτόματης τηλεφωνικής εξυπηρέτησης, ικανής να λειτουργεί σε πραγματικό περιβάλλον call center, χωρίς εξάρτηση από εμπορικές cloud υπηρεσίες. Η εργασία συνδυάζει θεωρητική ανάλυση των διαλογικών συστημάτων με πρακτική υλοποίηση βασισμένη σε τεχνολογίες ανοικτού κώδικα, όπως το Asterisk, το VICIdial, το μοντέλο Whisper για αναγνώριση ομιλίας και προσαρμοσμένο σύστημα κατανόησης φυσικής γλώσσας με χρήση PyTorch. Ιδιαίτερη έμφαση δίνεται στη modular αρχιτεκτονική, στη λειτουργία σε πραγματικό χρόνο και στην υποστήριξη της ελληνικής γλώσσας.

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, κ. Σταύρο Βολογιαννίδη, για την εμπιστοσύνη και για την ευκαιρία που μου έδωσε να υλοποιήσω μια ιδέα που είχα, στο πλαίσιο της παρούσας διπλωματικής εργασίας. Η στήριξη και η καθοδήγησή του ήταν πολύ σημαντικές σε όλη τη διάρκεια της προσπάθειας αυτής.

# Περίληψη στα Ελληνικά

Η παρούσα διπλωματική εργασία πραγματεύεται τη σχεδίαση και υλοποίηση ενός ελληνόφωνου τηλεφωνικού Voicebot, ενσωματωμένου σε πραγματικό περιβάλλον τηλεφωνικού κέντρου. Το προτεινόμενο σύστημα βασίζεται σε αρχιτεκτονική modular και αξιοποιεί τεχνολογίες επεξεργασίας φυσικής ομιλίας, μηχανικής μάθησης και τηλεφωνίας ανοικτού κώδικα. Η αρχιτεκτονική του συστήματος περιλαμβάνει υποσύστημα αναγνώρισης ομιλίας (Speech-to-Text) με χρήση του μοντέλου Whisper, υποσύστημα κατανόησης φυσικής γλώσσας (Natural Language Understanding) βασισμένο σε αναπαράσταση Bag-of-Words και νευρωνικό δίκτυο τύπου MLP, μηχανισμό διαχείρισης διαλόγου μέσω state machine και υποσύστημα σύνθεσης ομιλίας (Text-to-Speech). Η τηλεφωνική ενσωμάτωση πραγματοποιείται μέσω Asterisk και VICIdial, αξιοποιώντας τη διεπαφή AGI για έλεγχο της ροής της κλήσης. Το σύστημα υλοποιείται σε πλήρως on-premise υποδομή, χωρίς εξάρτηση από εμπορικές cloud υπηρεσίες, διασφαλίζοντας αυξημένο έλεγχο δεδομένων και επιχειρησιακή ευελιξία. Η εργασία καταδεικνύει ότι είναι εφικτή η ανάπτυξη ενός αξιόπιστου ελληνόφωνου Voicebot, ικανού να υποστηρίξει σύνθετες τηλεφωνικές ροές, όπως η παραγγελιοληψία, σε πραγματικές συνθήκες λειτουργίας.

**Λέξεις Κλειδιά:**

Voicebot, Τεχνητή Νοημοσύνη, Αναγνώριση Ομιλίας, Natural Language Understanding, Asterisk, VICIdial, Dialog Management, On-Premise AI

# Summary

This thesis presents the design and implementation of a Greek-speaking telephone Voicebot integrated into a real call center environment. The proposed system follows a modular architecture and leverages technologies from speech processing, machine learning, and open-source telephony platforms. The system architecture consists of a Speech-to-Text module based on the Whisper model, a Natural Language Understanding module using a Bag-of-Words representation and an MLP neural network, a dialog management mechanism implemented as a state machine, and a Text-to-Speech module for voice response generation.

Telephony integration is achieved through Asterisk and VICIdial, using the AGI interface to control call flow and interaction logic. The solution is deployed in a fully on-premise infrastructure, eliminating dependence on commercial cloud services and ensuring greater control over data and operational flexibility. The thesis demonstrates that it is feasible to develop a reliable Greek-language Voicebot capable of handling multi-step telephone processes, such as order placement, under real-world telephony constraints.

**Keywords:**

Voicebot, Artificial Intelligence, Speech Recognition, Natural Language Understanding, Asterisk, VICIdial, Dialog Management, On-Premise Deployment

## Πίνακας Περιεχομένων

ΚΕΦΑΛΑΙΟ 1 ..... 8
1.1 Αντικείμενο της Διπλωματικής Εργασίας ..... 8
1.2 Κίνητρο και Αναγκαιότητα της Εργασίας ..... 9
1.3 Στόχοι της Διπλωματικής ..... 10
1.4 Οργάνωση του Τόμου ..... 11
ΚΕΦΑΛΑΙΟ 2 ..... 12
2.1 Περιγραφή του Θέματος & Θεωρητικό Υπόβαθρο ..... 12
2.2 Θεωρητικό Υπόβαθρο ..... 13
2.2.1 Speech-to-Text (Αναγνώριση Ομιλίας) ..... 13
2.2.2 Natural Language Understanding (NLU) ..... 14
2.2.3 Text-to-Speech (TTS) ..... 14
2.2.4 AGI & Τηλεφωνική Ενσωμάτωση (Asterisk / VICIdial) ..... 14
2.2.5 Συνδυασμός Τεχνολογιών σε Διαλογικά Συστήματα ..... 15
2.3 Σχετικές Εργασίες και Εμπορικά Συστήματα ..... 15
2.4 Συνεισφορά της Εργασίας ..... 16
2.5 Συνοπτική Σύνοψη του Κεφαλαίου ..... 17
ΚΕΦΑΛΑΙΟ 3 ..... 19
3.1 Αρχιτεκτονική του Συστήματος ..... 19
3.2 Υποσύστημα Τηλεφωνίας (Asterisk / VICIdial) ..... 21
3.3 Υποσύστημα Speech-to-Text (Whisper) ..... 22
3.4 Υποσύστημα NLU (Intent Classifier) ..... 23
3.5 Σχεδιασμός Συστήματος Διαλόγου (Dialog Management) ..... 25
3.6 Υποσύστημα Text-to-Speech (gTTS) ..... 26
3.7 Υποσύστημα Καταγραφής Παραγγελιών (Order Logging) ..... 27
3.8 Web Interface (Django Management Console) ..... 28
3.9 Συνολική Ανάλυση και Σχεδιαστική Συνοχή ..... 28
ΚΕΦΑΛΑΙΟ 4 ..... 30
4.1 Εισαγωγή ..... 30
4.2 Τηλεφωνική Υποδομή: VICIdial έναντι FreePBX και Twilio ..... 31
4.3 Αναγνώριση Ομιλίας: Whisper έναντι Εμπορικών STT Υπηρεσιών ..... 34
4.4 Κατανόηση Φυσικής Γλώσσας: Custom NLU έναντι Dialogflow / Rasa ..... 35
4.5 Παραγωγή Ομιλίας: gTTS έναντι Εμπορικών TTS Λύσεων ..... 36
4.6 Συνολική Αξιολόγηση Τεχνολογικών Επιλογών ..... 39

ΚΕΦΑΛΑΙΟ 5 ..... 40
5.1 Εισαγωγή στην Υλοποίηση ..... 40
5.2 AGI Script – Το Κεντρικό Υποσύστημα (vici_voicebot.py) ..... 41
5.2.1 Ροή Εκτέλεσης AGI Script ..... 42
5.2.2 Βασικός Σκελετός AGI Script ..... 43
5.2.3 Ηχογράφηση φωνής από τον χρήστη ..... 45
5.2.4 Speech-to-Text (Whisper) εντός AGI ..... 46
5.2.5 Κλήση NLU Μοντέλου ..... 47
5.2.6 Διαχείριση Διαλόγου (State Machine) ..... 49
5.2.7 Παραγωγή Απάντησης (TTS) ..... 50
5.3 Υλοποίηση NLU Μοντέλου (nlu_intent_pt.py) ..... 51
5.3.1 Vectorization (Bag of Words) ..... 52
5.3.2 Μοντέλο PyTorch ..... 54
5.3.3 Πρόβλεψη Πρόθεσης (Intent Prediction) ..... 55
5.4 Εκπαίδευση Μοντέλου (train_intents.py) ..... 56
5.4.1 Αποθήκευση και Διαχείριση Μοντέλου ..... 57
5.5 Πρότυπα Απαντήσεων (response_templates.py) ..... 58
5.5.1 Αντιστοίχιση intents με λεκτικές αποκρίσεις ..... 58
5.5.2 Πλεονεκτήματα του διαχωρισμού περιεχομένου–λογικής ..... 58
5.5.3 Σταθερές και δυναμικές αποκρίσεις ..... 59
5.5.4 Σύνδεση με AGI και state machine ..... 60
5.5.5 Συνολική συμβολή της προσέγγισης ..... 60
5.6 Django Web Console – NLU Manager ..... 60
5.6.1 Διαχείριση δεδομένων εκπαίδευσης (CRUD) ..... 60
5.6.2 Εισαγωγή και εξαγωγή δεδομένων (Import / Export CSV) ..... 62
5.6.3 Επανεκπαίδευση μοντέλου μέσω Web Interface ..... 62
5.6.4 Intent Playground – Δοκιμή Προτάσεων ..... 63
5.6.5 Προβολή καταγεγραμμένων παραγγελιών (Voicebot Orders Viewer) ..... 64
5.6.6 Σχεδίαση interface ..... 64
5.7 Καταχώριση Παραγγελιών (order_log.csv) ..... 64
5.7.1 Δομή και περιεχόμενο του αρχείου καταγραφής ..... 65
5.7.2 Επιλογή μορφής CSV ..... 65
5.7.3 Ενσωμάτωση με το Web Περιβάλλον Διαχείρισης ..... 66
5.7.4 Πρακτική σημασία της καταγραφής ..... 66
5.8 Διαχείριση Αρχείων Ήχου ..... 66

5.8.1 Τύποι αρχείων ήχου ..... 66
5.8.2 Ονοματοδοσία και οργάνωση αρχείων ..... 67
5.8.3 Διαχείριση αποθηκευτικού χώρου και καθαρισμός αρχείων ..... 67
5.8.4 Ρόλος των αρχείων ήχου στη συνολική λειτουργία ..... 68
5.9 Θέματα Απόδοσης και Βελτιστοποίησης ..... 68
5.9.1 Βελτιστοποίηση υποσυστήματος Speech-to-Text ..... 68
5.9.2 Βελτιστοποίηση υποσυστήματος NLU ..... 69
5.9.3 Βελτιστοποίηση υποσυστήματος Text-to-Speech ..... 69
5.9.4 Διαχείριση πόρων και σταθερότητα συστήματος ..... 69
5.9.5 Συνολική αξιολόγηση απόδοσης ..... 69
ΚΕΦΑΛΑΙΟ 6 ..... 70
6.1 Εισαγωγή ..... 70
6.2 Αξιολόγηση Υποσυστήματος Κατανόησης Φυσικής Γλώσσας (NLU) ..... 70
6.2.1 Μετρικές Αξιολόγησης ..... 71
6.2.2 Αποτελέσματα Εκπαίδευσης ..... 71
6.2.3 Confusion Matrix ..... 74
6.3 Αξιολόγηση STT (Whisper Small) στο περιβάλλον τηλεφωνίας ..... 75
6.3.1 Μετρικές ..... 75
6.3.2 Πραγματικά Αποτελέσματα Κλήσεων ..... 75
6.4 Αξιολόγηση Διαλόγου & Λειτουργικότητας ..... 76
6.4.1 Ροή Παραγγελίας ..... 76
6.4.2 Χρόνος Απόκρισης ..... 77
6.5 Απόδοση Συστήματος & Αξιοπιστία ..... 78
6.5.1 Χρήση Υπολογιστικών Πόρων (CPU) ..... 78
6.5.2 Σφάλματα & Ανάκτηση ..... 79
6.6 Συμπεράσματα Αξιολόγησης ..... 79
ΚΕΦΑΛΑΙΟ 7 ..... 80
7.1 Συμπεράσματα ..... 80
7.2 Περιορισμοί της Υλοποίησης ..... 81
7.3 Μελλοντική Εργασία ..... 82
7.4 Τελική Εικόνα ..... 82
Βιβλιογραφία ..... 83
ΠΑΡΑΡΤΗΜΑ Α ..... 84
ΠΑΡΑΡΤΗΜΑ Α.1 — AGI Script (vici_voicebot.py) ..... 84
ΠΑΡΑΡΤΗΜΑ Α.2 — Εκπαίδευση NLU (train_intents.py) ..... 96

ΠΑΡΑΡΤΗΜΑ Β — Web Interface (Django) ..... 101
ΠΑΡΑΡΤΗΜΑ Β.1 — models.py ..... 101
ΠΑΡΑΡΤΗΜΑ Β.2 — views.py (Intent Testing & Retrain) ..... 101
ΠΑΡΑΡΤΗΜΑ Β.3 — admin.py ..... 110
ΠΑΡΑΡΤΗΜΑ Β.4 — URL Configuration (urls.py) ..... 110

ΚΕΦΑΛΑΙΟ 1

# Εισαγωγή

## 1.1 Αντικείμενο της Διπλωματικής Εργασίας

Η ραγδαία εξέλιξη των συστημάτων τεχνητής νοημοσύνης κατά τις τελευταίες δεκαετίες και ειδικότερα των τεχνολογιών επεξεργασίας φυσικής ομιλίας (Speech Processing) έχει αλλάξει τον τρόπο με τον οποίο ο άνθρωπος αλληλεπιδρά με την τεχνολογία και τους υπολογιστές. Η δυνατότητα κατανόησης και παραγωγής φυσικού λόγου από υπολογιστικά συστήματα έχει οδηγήσει στην ανάπτυξη ευφυών εφαρμογών, οι οποίες προσπαθούν να προσεγγίσουν τη φυσική ανθρώπινη επικοινωνία, τόσο σε επίπεδο μορφής όσο και σε επίπεδο λειτουργικότητας (Jurafsky and Martin, 2023).
Τα σύγχρονα συστήματα επεξεργασίας φυσικής ομιλίας βασίζονται στον συνδυασμό τεχνικών αυτόματης αναγνώρισης ομιλίας (Speech-to-Text), κατανόησης φυσικής γλώσσας (Natural Language Understanding) και σύνθεσης ομιλίας (Text-to-Speech) (Jurafsky and Martin, 2023). Ο συνδυασμός αυτών των τεχνολογιών επιτρέπει τη δημιουργία ψηφιακών βοηθών ικανών να δέχονται φωνητικά μηνύματα, να εξάγουν το νόημα και την πρόθεση του χρήστη και να παράγουν κατάλληλες λεκτικές απαντήσεις. Τα συστήματα αυτά δεν περιορίζονται πλέον σε πειραματικές ή ερευνητικές εφαρμογές αλλά αξιοποιούνται σε πραγματικά περιβάλλοντα επικοινωνίας, όπως τηλεφωνικά κέντρα, γραμμές εξυπηρέτησης πελατών και συστήματα αυτοματοποιημένης υποστήριξης. Ένα χαρακτηριστικό παράδειγμα τέτοιων συστημάτων αποτελεί το Voicebot, δηλαδή ένα αυτόνομο διαλογικό σύστημα το οποίο επικοινωνεί με τον χρήστη μέσω φυσικής ομιλίας, χωρίς την ανάγκη ανθρώπινου χειριστή. Τα Voicebots είναι σε θέση να αναλαμβάνουν επαναλαμβανόμενες και χρονοβόρες εργασίες, όπως η παροχή πληροφοριών, η καθοδήγηση χρηστών, η καταγραφή αιτημάτων και η αυτοματοποίηση απλών επιχειρησιακών διαδικασιών. Η χρήση τους έχει αυξηθεί σημαντικά σε τομείς όπως η εξυπηρέτηση πελατών, η δημόσια διοίκηση, η υγεία και το ηλεκτρονικό εμπόριο, βοηθώντας στη βελτίωση της αποδοτικότητας και της ποιότητας των παρεχόμενων υπηρεσιών (Young et al., 2013).

Αντικείμενο της παρούσας διπλωματικής εργασίας αποτελεί η σχεδίαση και υλοποίηση ενός ολοκληρωμένου ελληνόφωνου συστήματος Voicebot, το οποίο ενσωματώνεται σε πραγματικό τηλεφωνικό κέντρο. Το σύστημα βασίζεται στο τηλεφωνικό κέντρο Asterisk και στην πλατφόρμα VICIdial, επιτρέποντας την άμεση σύνδεση με υφιστάμενες υποδομές τηλεφωνικών κέντρων. Το προτεινόμενο Voicebot είναι σε θέση να διαχειρίζεται εισερχόμενες τηλεφωνικές κλήσεις, να αναγνωρίζει την ελληνική ομιλία, να κατανοεί τις προθέσεις και τα αιτήματα του χρήστη, να οργανώνει τη ροή της συνομιλίας μέσω μηχανισμού κατάστασης (state machine), να παράγει συνθετική ομιλία και να αυτοματοποιεί διαδικασίες όπως η καταχώριση παραγγελιών. Ιδιαίτερη σημασία δίνεται στο γεγονός ότι το σύστημα σχεδιάστηκε ώστε να λειτουργεί σε πραγματικές συνθήκες τηλεφωνίας, λαμβάνοντας υπόψη περιορισμούς όπως η μειωμένη ποιότητα ήχου, ο θόρυβος της γραμμής και οι απαιτήσεις απόκρισης σε πραγματικό χρόνο. Επίσης δίνεται έμφαση στη χρήση τεχνολογιών ανοιχτού κώδικα (open-source) και στη λειτουργία του συστήματος σε πλήρως on-premise υποδομή (στο δικό της κτίριο ή data center, χωρίς εξάρτηση από εμπορικές cloud υπηρεσίες. Η επιλογή αυτή αυξάνει τον έλεγχο των

δεδομένων, παρέχει ενισχυμένη ασφάλεια και ιδιωτικότητα και ευελιξία προσαρμογής στις ιδιαίτερες επιχειρησιακές ανάγκες κάθε οργανισμού (ITU-T, 2016). Συνολικά, το αντικείμενο της παρούσας διπλωματικής εργασίας τοποθετείται στο σημείο σύγκλισης της τεχνητής νοημοσύνης, της επεξεργασίας φυσικής ομιλίας και των τηλεφωνικών συστημάτων, με στόχο την ανάπτυξη ενός πρακτικά αξιοποιήσιμου ελληνόφωνου Voicebot, ικανού να υποστηρίξει σύγχρονες ανάγκες αυτοματοποιημένης τηλεφωνικής εξυπηρέτησης.

## 1.2 Κίνητρο και Αναγκαιότητα της Εργασίας

Η συνεχής αύξηση του όγκου των τηλεφωνικών κλήσεων και η ανάγκη για άμεση ποιοτική και αδιάλειπτη εξυπηρέτηση πελατών αποτελούν βασικές προκλήσεις για τα σύγχρονα τηλεφωνικά κέντρα και οργανισμούς παροχής υπηρεσιών. Οι επιχειρήσεις καλούνται να ανταποκριθούν σε αυξημένες απαιτήσεις χρηστών, οι οποίοι αναμένουν γρήγορη εξυπηρέτηση, σαφείς απαντήσεις και συνεπή εμπειρία επικοινωνίας, ανεξαρτήτως χρονικής στιγμής ή φόρτου εργασίας. Παράλληλα, η ανάγκη για μείωση του λειτουργικού κόστους και για αποδοτικότερη αξιοποίηση ανθρώπινων πόρων καθιστά αναγκαία την υιοθέτηση αυτοματοποιημένων. Η παραδοσιακή προσέγγιση τηλεφωνικής εξυπηρέτησης, η οποία βασίζεται αποκλειστικά σε ανθρώπινους εκπροσώπους, παρουσιάζει σημαντικούς περιορισμούς. Το κόστος στελέχωσης και εκπαίδευσης προσωπικού είναι υψηλό, ενώ η διαθεσιμότητα περιορίζεται συνήθως σε συγκεκριμένο ωράριο λειτουργίας.

Οι χρόνοι αναμονής αυξάνονται σημαντικά σε περιόδους αιχμής, γεγονός που οδηγεί σε μειωμένη ικανοποίηση των χρηστών. Παράλληλα, η ποιότητα της εξυπηρέτησης μπορεί να διαφοροποιείται ανάλογα με τον εκάστοτε εκπρόσωπο, γεγονός που δυσχεραίνει τη διατήρηση ενιαίου επιπέδου παρεχόμενων υπηρεσιών. Ως εναλλακτική λύση, τα κλασικά συστήματα IVR (Interactive Voice Response) χρησιμοποιούνται ευρέως για την αυτοματοποίηση βασικών λειτουργιών. Ωστόσο τα συστήματα αυτά βασίζονται σε προκαθορισμένα αριθμητικά μενού επιλογών, τα οποία συχνά χαρακτηρίζονται από άκαμπτη δομή και περιορισμένη ευελιξία. Η πλοήγηση σε πολύπλοκα μενού απαιτεί χρόνο και προσπάθεια από τον χρήστη, ενώ δεν επιτρέπει την ελεύθερη διατύπωση αιτημάτων, γεγονός που συχνά οδηγεί σε αρνητική εμπειρία χρήσης (Young et al., 2013). Στο πλαίσιο αυτό, τα Voicebots που αξιοποιούν τεχνολογίες τεχνητής νοημοσύνης και επεξεργασίας φυσικής γλώσσας αναδεικνύονται ως μια σύγχρονη και αποτελεσματική λύση. Η δυνατότητα φυσικής λεκτικής αλληλεπίδρασης επιτρέπει στους χρήστες να εκφράζουν τα αιτήματά τους με τον τρόπο που θα μιλούσαν σε έναν άνθρωπο χωρίς την ανάγκη απομνημόνευσης ή επιλογής αριθμητικών μενού. Παράλληλα, τα Voicebots μπορούν να εξυπηρετούν ταυτόχρονα μεγάλο αριθμό κλήσεων, να παρέχουν απαντήσεις και να λειτουργούν 24/7, ανεξάρτητα από ανθρώπινους περιορισμούς (Sutton and Barto, 2018).

Ιδιαίτερη σημασία αποκτά η ανάπτυξη ελληνόφωνων Voicebots, καθώς η υποστήριξη της ελληνικής γλώσσας παραμένει περιορισμένη σε πολλές εμπορικές πλατφόρμες, ειδικά σε περιβάλλοντα τηλεφωνικής ποιότητας ήχου. Η ανάγκη για συστήματα που μπορούν να κατανοούν και να επεξεργάζονται αξιόπιστα τον ελληνικό προφορικό λόγο, λαμβάνοντας υπόψη ιδιαιτερότητες όπως η μορφολογία, η προφορά και οι γλωσσικές παραλλαγές, καθιστά την ανάπτυξη τέτοιων λύσεων ιδιαίτερα επίκαιρη (OpenAI, 2023).

Στο πλαίσιο αυτό, η παρούσα διπλωματική εργασία στοχεύει στην κάλυψη ενός υπαρκτού τεχνολογικού και επιχειρησιακού κενού, προτείνοντας ένα ελληνόφωνο Voicebot που λειτουργεί σε πραγματικό τηλεφωνικό περιβάλλον και μπορεί να παραμετροποιηθεί και να εκπαιδευτεί από τον ίδιο τον οργανισμό-εταιρεία. Η επιλογή υλοποίησης σε πλήρως on-premise υποδομή, χωρίς εξάρτηση από εμπορικές cloud υπηρεσίες, ενισχύει περαιτέρω τη χρησιμότητα της προτεινόμενης λύσης, προσφέροντας αυξημένο έλεγχο, ασφάλεια και ευελιξία προσαρμογής στις ανάγκες διαφορετικών οργανισμών.

## 1.3 Στόχοι της Διπλωματικής

Οι βασικοί στόχοι της παρούσας διπλωματικής εργασίας συνοψίζονται ως εξής:

1. Ανάπτυξη συστήματος αυτόματης αναγνώρισης ομιλίας (Speech-to-Text). Ενσωμάτωση του μοντέλου Whisper STT για τη μεταγραφή της ελληνικής ομιλίας σε κείμενο, λαμβάνοντας υπόψη τις ιδιαιτερότητες της τηλεφωνικής ποιότητας ήχου και τους περιορισμούς πραγματικού χρόνου.

2. Εκπαίδευση μοντέλου κατανόησης φυσικής γλώσσας (Natural Language Understanding – NLU). Υλοποίηση ταξινομητή προθέσεων με χρήση της βιβλιοθήκης PyTorch, βασισμένου σε αναπαράσταση κειμένου τύπου Bag-of-Words και σε απλό νευρωνικό δίκτυο τύπου MLP.

3. Ενσωμάτωση του Voicebot σε τηλεφωνικό σύστημα VICIdial μέσω Asterisk AGI. Διασύνδεση του συστήματος με υπάρχουσα τηλεφωνική υποδομή, επιτρέποντας την αυτόματη εξυπηρέτηση εισερχόμενων τηλεφωνικών κλήσεων χωρίς ανθρώπινη παρέμβαση.

4. Υλοποίηση διαλόγου πολλαπλών βημάτων για διαδικασία παραγγελιοληψίας. Σχεδίαση και υλοποίηση μηχανισμού διαλόγου βασισμένου σε state machine, μέσω του οποίου συλλέγονται διαδοχικά τα απαραίτητα στοιχεία της παραγγελίας με ελεγχόμενο και αξιόπιστο τρόπο.

5. Ανάπτυξη Web interface διαχείρισης με χρήση Django. Δημιουργία γραφικού περιβάλλοντος για την εκπαίδευση, τον έλεγχο και την αξιολόγηση του NLU μοντέλου, καθώς και για την προβολή και διαχείριση των καταγεγραμμένων παραγγελιών.

## 1.4 Οργάνωση του Τόμου

Η διπλωματική εργασία οργανώνεται σε επτά κεφάλαια, τα οποία καλύπτουν διαδοχικά το θεωρητικό υπόβαθρο, τον σχεδιασμό, τις τεχνολογικές επιλογές, την υλοποίηση και την αξιολόγηση του προτεινόμενου συστήματος:

• **Κεφάλαιο 1 – Εισαγωγή:**

Παρουσιάζεται το αντικείμενο της εργασίας, το κίνητρο και η αναγκαιότητα της έρευνας, οι βασικοί στόχοι της διπλωματικής, καθώς και η συνολική δομή του τόμου.

• **Κεφάλαιο 2 – Περιγραφή του Θέματος:**

Αναλύεται το θεωρητικό υπόβαθρο που αφορά στα συστήματα Voicebot, στην αυτόματη αναγνώριση ομιλίας, στην κατανόηση φυσικής γλώσσας και στη σύνθεση ομιλίας. Παρουσιάζονται επίσης σχετικές ερευνητικές και εμπορικές προσεγγίσεις, καθώς και η συνεισφορά της παρούσας εργασίας.

• **Κεφάλαιο 3 – Ανάλυση και Σχεδίαση:**

Περιγράφεται η αρχιτεκτονική του προτεινόμενου συστήματος, τα επιμέρους υποσυστήματα, η ροή πληροφορίας, καθώς και ο σχεδιασμός του διαλόγου, του NLU μοντέλου και της συνολικής λειτουργικής συνοχής του Voicebot.

• **Κεφάλαιο 4 – Εργαλεία και Τεχνολογικές Επιλογές:**

Παρουσιάζονται και αξιολογούνται συγκριτικά οι βασικές τεχνολογίες που χρησιμοποιήθηκαν (Asterisk/VICIdial, Whisper, custom NLU, gTTS), σε αντιπαραβολή με εναλλακτικές και εμπορικές λύσεις. Τεκμηριώνονται οι λόγοι επιλογής τους με βάση κριτήρια όπως η on-premise λειτουργία, το κόστος, ο έλεγχος δεδομένων και η καταλληλότητα για τηλεφωνικά περιβάλλοντα.

• **Κεφάλαιο 5 – Υλοποίηση:**

Αναλύονται οι τεχνικές λεπτομέρειες υλοποίησης του συστήματος, η ενσωμάτωση με το Asterisk και το VICIdial, η λειτουργία του AGI script, η εκπαίδευση και αξιοποίηση του NLU μοντέλου, καθώς και το web περιβάλλον διαχείρισης.

• **Κεφάλαιο 6 – Πειραματική Αξιολόγηση:**

Παρουσιάζονται τα αποτελέσματα της πειραματικής λειτουργίας του Voicebot, με ανάλυση μετρικών απόδοσης για τα υποσυστήματα STT και NLU, καθώς και παρατηρήσεις από τη χρήση του συστήματος σε πραγματικές συνθήκες.

• **Κεφάλαιο 7 – Συμπεράσματα και Μελλοντική Εργασία:**

Συνοψίζονται τα βασικά συμπεράσματα της διπλωματικής εργασίας, αναλύονται οι περιορισμοί της υλοποίησης και προτείνονται κατευθύνσεις για μελλοντική βελτίωση και επέκταση του συστήματος.

# Περιγραφή του θέματος

## 2.1 Περιγραφή του Θέματος & Θεωρητικό Υπόβαθρο

Τα συστήματα αυτόματης τηλεφωνικής εξυπηρέτησης αποτελούν βασικό στοιχείο της σύγχρονης επιχειρησιακής επικοινωνίας, καθώς χρησιμοποιούνται ευρέως για τη διαχείριση εισερχόμενων και εξερχόμενων κλήσεων, την παροχή πληροφοριών και την αυτοματοποίηση βασικών διαδικασιών εξυπηρέτησης πελατών. Η εξέλιξη αυτών των συστημάτων κατά τις τελευταίες δεκαετίες υπήρξε ιδιαίτερα σημαντική, ακολουθώντας την τεχνολογική πρόοδο στους τομείς της πληροφορικής, των τηλεπικοινωνιών και πιο πρόσφατα της τεχνητής νοημοσύνης. Τα πρώτα συστήματα αυτόματης τηλεφωνικής εξυπηρέτησης βασίζονταν σε απλές δομές IVR (Interactive Voice Response), οι οποίες παρείχαν περιορισμένη λειτουργικότητα και απαιτούσαν από τον χρήστη να αλληλεπιδρά μέσω αριθμητικών επιλογών (Weizenbaum, 1966; McTear, 2010).

Η χρήση αριθμητικών μενού, αν και αποτελεσματική για απλές δρομολογήσεις κλήσεων, αποδείχθηκε ανεπαρκής για πιο σύνθετες ανάγκες επικοινωνίας. Η αυστηρά προκαθορισμένη δομή των IVR συστημάτων δεν επιτρέπει την ελεύθερη διατύπωση αιτημάτων, ενώ η πλοήγηση σε εκτεταμένα μενού συχνά οδηγεί σε σύγχυση, αυξημένους χρόνους εξυπηρέτησης και μειωμένη ικανοποίηση των χρηστών (McTear, 2010). Οι περιορισμοί αυτοί ανέδειξαν την ανάγκη για πιο ευέλικτες και «ανθρώπινες» μορφές τηλεφωνικής αλληλεπίδρασης. Η ανάπτυξη τεχνολογιών επεξεργασίας φυσικής γλώσσας και μηχανικής μάθησης κατέστησε δυνατή τη μετάβαση από στατικά τηλεφωνικά συστήματα σε έξυπνα συστήματα διαλόγου, ικανά να κατανοούν τον προφορικό λόγο και να αντιδρούν δυναμικά στο περιεχόμενο της συνομιλίας. Σε αυτό το πλαίσιο τα Voicebots αναδύθηκαν ως μια νέα κατηγορία τηλεφωνικών συστημάτων, τα οποία συνδυάζουν την αναγνώριση ομιλίας, την κατανόηση φυσικής γλώσσας και τη σύνθεση ομιλίας με στόχο την όσο το δυνατόν πιο φυσική επικοινωνία με τον χρήστη (Jurafsky & Martin, 2023).

Η παρούσα εργασία επικεντρώνεται στην ανάπτυξη ενός Voicebot, το οποίο λειτουργεί ως αυτόματος τηλεφωνικός βοηθός και αντικαθιστά, σε συγκεκριμένα σενάρια τον άνθρωπο. Το Voicebot επιτρέπει στον χρήστη να εκφράζει ελεύθερα το αίτημά του μέσω φυσικής ομιλίας, χωρίς την ανάγκη επιλογής προκαθορισμένων μενού. Η δυνατότητα αυτή καθιστά τη συνομιλία πιο άμεση και φιλική, ενώ ταυτόχρονα επιτρέπει στο σύστημα να επεξεργάζεται πιο σύνθετες πληροφορίες και να προσαρμόζει τη συμπεριφορά του ανάλογα με τις απαιτήσεις του συνομιλητή. Στο πλαίσιο της παρούσας εργασίας, το Voicebot σχεδιάζεται ως ένας πλήρως αυτοματοποιημένος τηλεφωνικός μηχανισμός, ο οποίος αναλαμβάνει ολόκληρη τη ροή της επικοινωνίας με τον συνομιλητή. Το σύστημα είναι σε θέση να απαντά αυτόματα σε εισερχόμενες κλήσεις, να καταγράφει την ομιλία του χρήστη, να μετατρέπει το ηχητικό σήμα σε κείμενο, να αναλύει το περιεχόμενο της φράσης και να εντοπίζει την πρόθεση του συνομιλητή. Με βάση την αναγνωρισμένη πρόθεση, το Voicebot μπορεί να εκκινεί κατάλληλες ενέργειες, όπως η παροχή πληροφοριών ή η

καταχώριση μιας παραγγελίας, και να παράγει φωνητικές απαντήσεις με συνεπή και κατανοητό τρόπο. Ιδιαίτερη σημασία δίνεται στο γεγονός ότι το σύστημα δεν σχεδιάζεται ως θεωρητικό πρωτότυπο αλλά ως λειτουργική λύση για πραγματικές τηλεφωνικές ανάγκες. Τα σενάρια χρήσης περιλαμβάνουν διαδικασίες όπως η λήψη παραγγελιών, η ενημέρωση πελατών, η αναζήτηση πληροφοριών και η απάντηση σε συχνά ερωτήματα. Η αρχιτεκτονική του Voicebot είναι επεκτάσιμη, επιτρέποντας τη μελλοντική ενσωμάτωση πιο σύνθετων λειτουργιών, όπως τεχνική υποστήριξη και διαχείριση αιτημάτων.
Η υλοποίηση του προτεινόμενου συστήματος βασίζεται σε τρεις θεμελιώδεις τεχνολογικούς άξονες: την αυτόματη αναγνώριση ομιλίας (Speech-to-Text), την κατανόηση φυσικής γλώσσας (Natural Language Understanding) και τη σύνθεση ομιλίας (Text-to-Speech). Οι τεχνολογίες αυτές λειτουργούν συμπληρωματικά, σχηματίζοντας μια ενιαία αλυσίδα επεξεργασίας που επιτρέπει τη μετατροπή της ανθρώπινης ομιλίας σε επεξεργάσιμη πληροφορία και στη συνέχεια στη παραγωγή κατάλληλων φωνητικών απαντήσεων.

## 2.2 Θεωρητικό Υπόβαθρο

Η ανάπτυξη ενός σύγχρονου Voicebot εντάσσεται στο ευρύτερο πεδίο των διαλογικών συστημάτων (conversational agents), τα οποία στοχεύουν στη φυσική και αποδοτική επικοινωνία μεταξύ ανθρώπου και μηχανής. Τα συστήματα αυτά αποτελούν συνδυασμό τεχνολογιών από διαφορετικούς επιστημονικούς τομείς, όπως η επεξεργασία φυσικής γλώσσας, η μηχανική μάθηση, η επεξεργασία σήματος και οι τηλεπικοινωνίες. Για τη σωστή κατανόηση της λειτουργίας και των δυνατοτήτων ενός Voicebot, είναι απαραίτητη η ανάλυση των βασικών τεχνολογικών συνιστωσών που το απαρτίζουν
Στον πυρήνα κάθε Voicebot βρίσκεται η ικανότητα επεξεργασίας της ανθρώπινης ομιλίας, τόσο σε επίπεδο αναγνώρισης όσο και σε επίπεδο κατανόησης και παραγωγής απαντήσεων. Η διαδικασία αυτή υλοποιείται μέσω μιας αλυσίδας επεξεργασίας, η οποία ξεκινά από τη λήψη του ηχητικού σήματος και καταλήγει στη σύνθεση κατάλληλης φωνητικής απόκρισης. Κάθε στάδιο της αλυσίδας αυτής επηρεάζει άμεσα τη συνολική απόδοση και την ποιότητα της επικοινωνίας (Jurafsky & Martin, 2023; McTear, 2010).

### 2.2.1 Speech-to-Text (Αναγνώριση Ομιλίας)

Η αναγνώριση ομιλίας (Speech-to-Text, STT) αποτελεί το πρώτο και θεμελιώδες στάδιο στη λειτουργία ενός Voicebot. Στόχος της είναι η μετατροπή του ηχητικού σήματος που παράγεται από τον χρήστη σε γραπτό κείμενο, το οποίο μπορεί στη συνέχεια να αναλυθεί από υπολογιστικά συστήματα. Η διαδικασία αυτή περιλαμβάνει τεχνικές επεξεργασίας σήματος και μοντέλα μηχανικής μάθησης, τα οποία επιχειρούν να αντιστοιχίσουν ακουστικά μοτίβα σε γλωσσικές μονάδες. Η αναγνώριση ομιλίας σε τηλεφωνικό περιβάλλον παρουσιάζει πρόσθετες δυσκολίες, καθώς το φάσμα συχνοτήτων είναι περιορισμένο (συνήθως 300–3400 Hz), ενώ συχνά παρατηρούνται θόρυβος γραμμής, συμπίεση σήματος και μεταβολές στην ποιότητα της φωνής. Τα σύγχρονα συστήματα STT βασίζονται σε βαθιά νευρωνικά δίκτυα, τα οποία έχουν εκπαιδευτεί σε μεγάλα σύνολα δεδομένων και μπορούν να γενικεύουν αποτελεσματικά ακόμη και σε υποβαθμισμένες συνθήκες ήχου. Στην παρούσα εργασία χρησιμοποιείται το Whisper, ένα μοντέλο νευρωνικών δικτύων το οποίο έχει σχεδιαστεί για πολυγλωσσική αναγνώριση ομιλίας και παρουσιάζει αυξημένη ανθεκτικότητα σε θορυβώδη δεδομένα. Η επιλογή του Whisper βασίζεται τόσο στην ακρίβειά του στην ελληνική γλώσσα όσο και στη δυνατότητα

εκτέλεσής του σε τοπικό περιβάλλον, χωρίς εξάρτηση από εξωτερικές υπηρεσίες. Η αξιοπιστία του σταδίου STT είναι κρίσιμη, καθώς τυχόν σφάλματα στη μεταγραφή επηρεάζουν άμεσα όλα τα επόμενα στάδια της αλυσίδας επεξεργασίας (Radford et al., 2023).

### 2.2.2 Natural Language Understanding (NLU)

Το στάδιο της κατανόησης φυσικής γλώσσας (Natural Language Understanding, NLU) αποτελεί τον λογικό πυρήνα του Voicebot. Ενώ το STT μετατρέπει την ομιλία σε κείμενο, το NLU είναι υπεύθυνο για την ερμηνεία του νοήματος της φράσης και για την εξαγωγή της πρόθεσης (intent) του χρήστη. Η διαδικασία αυτή περιλαμβάνει τεχνικές ανάλυσης κειμένου, αναπαράστασης γλωσσικής πληροφορίας και ταξινόμησης. Σε πρακτικές εφαρμογές Voicebot, το NLU καλείται να απαντήσει σε ερωτήματα όπως: «Τι θέλει να κάνει ο χρήστης;», «Ποια πληροφορία παρέχει;» και «Ποια ενέργεια πρέπει να εκτελεστεί;». Για τον σκοπό αυτό, χρησιμοποιούνται μοντέλα ταξινόμησης προθέσεων, τα οποία αντιστοιχίζουν το κείμενο εισόδου σε ένα προκαθορισμένο σύνολο intents. Σε πιο σύνθετα συστήματα, το NLU μπορεί να περιλαμβάνει και μηχανισμούς εξαγωγής οντοτήτων (slots), επιτρέποντας λεπτομερέστερη κατανόηση του περιεχομένου (Jurafsky & Martin, 2023).

Στην παρούσα εργασία επιλέγεται μια προσέγγιση βασισμένη σε αναπαράσταση Bag-of-Words και σε νευρωνικό δίκτυο τύπου Multilayer Perceptron (MLP). Παρότι πρόκειται για σχετικά απλή προσέγγιση σε σύγκριση με σύγχρονα μοντέλα βασισμένα σε μετασχηματιστές (transformers), προσφέρει ικανοποιητική ακρίβεια με χαμηλές υπολογιστικές απαιτήσεις. Η επιλογή αυτή είναι ιδιαίτερα σημαντική για συστήματα που λειτουργούν σε πραγματικό χρόνο και σε περιβάλλοντα περιορισμένων πόρων, όπως τηλεφωνικοί servers (Manning et al., 2008).

### 2.2.3 Text-to-Speech (TTS)

Το υποσύστημα σύνθεσης ομιλίας (Text-to-Speech, TTS) αποτελεί το τελικό στάδιο της αλυσίδας επεξεργασίας και είναι υπεύθυνο για τη μετατροπή του κειμένου απόκρισης σε ακουστική μορφή. Η ποιότητα της παραγόμενης φωνής επηρεάζει άμεσα την εμπειρία του χρήστη και την αίσθηση «φυσικότητας» της συνομιλίας
Τα σύγχρονα συστήματα TTS βασίζονται σε στατιστικά ή νευρωνικά μοντέλα, τα οποία παράγουν φωνή με ρεαλιστικά χαρακτηριστικά προφοράς, τονισμού και ρυθμού. Στο πλαίσιο της εργασίας χρησιμοποιείται το εργαλείο gTTS, το οποίο παρέχει υποστήριξη για την ελληνική γλώσσα και παράγει κατανοητή και ευχάριστη συνθετική φωνή (Taylor, 2009). Ιδιαίτερη πρόκληση αποτελεί η ενσωμάτωση του TTS σε τηλεφωνικό περιβάλλον, καθώς απαιτείται μετατροπή των αρχείων ήχου σε μορφές συμβατές με τα τηλεφωνικά πρωτόκολλα και τον Asterisk. Η σωστή ρύθμιση του υποσυστήματος TTS εξασφαλίζει ομαλή ροή του διαλόγου και αποφυγή καθυστερήσεων ή τεχνικών προβλημάτων κατά την αναπαραγωγή της φωνής.

### 2.2.4 AGI & Τηλεφωνική Ενσωμάτωση (Asterisk / VICIdial)

Η λειτουργία ενός Voicebot σε πραγματικό τηλεφωνικό περιβάλλον προϋποθέτει τη διασύνδεσή του με συστήματα τηλεφωνίας, τα οποία είναι υπεύθυνα για τη διαχείριση των κλήσεων. Το Asterisk αποτελεί ένα από τα πλέον διαδεδομένα ανοικτού κώδικα τηλεφωνικά συστήματα και προσφέρει εκτεταμένες δυνατότητες παραμετροποίησης και

αυτοματοποίησης. Μέσω της διεπαφής Asterisk Gateway Interface (AGI), εξωτερικά προγράμματα μπορούν να αλληλεπιδρούν με το τηλεφωνικό σύστημα, λαμβάνοντας πληροφορίες για την κλήση, ηχογραφώντας την ομιλία του χρήστη και ελέγχοντας τη ροή της συνομιλίας. Η ενσωμάτωση του Voicebot στο περιβάλλον VICIdial, το οποίο χρησιμοποιείται ευρέως σε τηλεφωνικά κέντρα, καθιστά το σύστημα άμεσα εφαρμόσιμο σε πραγματικές επιχειρησιακές συνθήκες (Asterisk, 2024).

### 2.2.5 Συνδυασμός Τεχνολογιών σε Διαλογικά Συστήματα

Η ουσία ενός Voicebot δεν βρίσκεται σε ένα μεμονωμένο υποσύστημα, αλλά στον συνδυασμό και τη συνεργασία όλων των παραπάνω τεχνολογιών. Η επιτυχής ενοποίηση STT, NLU, TTS και τηλεφωνίας δημιουργεί ένα συνεκτικό σύστημα, το οποίο μπορεί να λειτουργεί σε πραγματικό χρόνο και να προσφέρει φυσική εμπειρία συνομιλίας (McTear, 2010). Η θεωρητική κατανόηση των επιμέρους σταδίων επιτρέπει τον ορθό σχεδιασμό της αρχιτεκτονικής και τη σωστή επιλογή τεχνολογιών, στοιχεία που αποτελούν τη βάση για την ανάλυση και τον σχεδιασμό του συστήματος που παρουσιάζεται στο επόμενο κεφάλαιο.

## 2.3 Σχετικές Εργασίες και Εμπορικά Συστήματα

Η ανάπτυξη διαλογικών συστημάτων και Voicebots έχει αποτελέσει αντικείμενο εκτεταμένης έρευνας και εμπορικής εκμετάλλευσης τα τελευταία χρόνια, τόσο σε ακαδημαϊκό όσο και σε βιομηχανικό επίπεδο. Στη διεθνή βιβλιογραφία, τα Voicebots εντάσσονται στο ευρύτερο πεδίο των spoken dialogue systems και των conversational agents, με στόχο τη φυσική αλληλεπίδραση ανθρώπου–μηχανής μέσω φωνής (Jurafsky & Martin, 2023).

**Ακαδημαϊκές και Ερευνητικές Προσεγγίσεις**

Στο ακαδημαϊκό περιβάλλον, μεγάλο μέρος της έρευνας επικεντρώνεται στη βελτίωση των επιμέρους υποσυστημάτων που απαρτίζουν ένα Voicebot, όπως η αναγνώριση ομιλίας, η κατανόηση φυσικής γλώσσας και η διαχείριση διαλόγου. Πολλές ερευνητικές εργασίες προτείνουν σύνθετα στατιστικά ή νευρωνικά μοντέλα για την αναγνώριση προθέσεων και την εξαγωγή οντοτήτων, καθώς και εξελιγμένους μηχανισμούς διαχείρισης διαλόγου βασισμένους σε πιθανοτικά μοντέλα ή ενισχυτική μάθηση (reinforcement learning). Ιδιαίτερη έμφαση δίνεται στη χρήση μοντέλων βαθιάς μάθησης, όπως recurrent neural networks (RNNs), long short-term memory networks (LSTMs) και, πιο πρόσφατα, μοντέλων μετασχηματιστών (transformers), τα οποία έχουν επιδείξει υψηλή απόδοση σε σύνθετα γλωσσικά καθήκοντα. Ωστόσο, πολλές από αυτές τις προσεγγίσεις παρουσιάζουν αυξημένες υπολογιστικές απαιτήσεις και προϋποθέτουν τη χρήση μεγάλων και επιμελημένων συνόλων δεδομένων, γεγονός που δυσχεραίνει την πρακτική τους εφαρμογή σε περιβάλλοντα πραγματικού χρόνου ή σε συστήματα περιορισμένων πόρων.

Επιπλέον, σημαντικό μέρος της σχετικής βιβλιογραφίας εστιάζει σε αγγλόφωνα δεδομένα, με περιορισμένη κάλυψη γλωσσών όπως η ελληνική. Αυτό δημιουργεί ένα κενό μεταξύ θεωρητικής έρευνας και πρακτικής εφαρμογής σε ελληνόφωνα περιβάλλοντα, ιδιαίτερα σε τηλεφωνικά συστήματα.

**Εμπορικά Συστήματα και Πλατφόρμες**

Παράλληλα με την ακαδημαϊκή έρευνα, έχουν αναπτυχθεί πολυάριθμες εμπορικές πλατφόρμες για τη δημιουργία Voicebots και conversational agents. Ενδεικτικά αναφέρονται λύσεις όπως τα Google Dialogflow, Amazon Lex και Twilio Programmable Voice, οι οποίες παρέχουν ολοκληρωμένα εργαλεία για την ανάπτυξη, εκπαίδευση και ανάπτυξη διαλογικών συστημάτων. Οι πλατφόρμες αυτές προσφέρουν πλεονεκτήματα όπως εύχρηστα γραφικά περιβάλλοντα, ενσωματωμένα μοντέλα κατανόησης φυσικής γλώσσας και έτοιμες συνδέσεις με τηλεφωνικές ή διαδικτυακές υπηρεσίες. Ωστόσο, στηρίζονται κατά κύριο λόγο σε cloud υποδομές, γεγονός που συνεπάγεται συνεχή εξάρτηση από τρίτους παρόχους, αυξημένο κόστος λειτουργίας και περιορισμένο έλεγχο επί των δεδομένων. Επίσης η υποστήριξη της ελληνικής γλώσσας στις εμπορικές αυτές πλατφόρμες είναι συχνά περιορισμένη ή δευτερεύουσας προτεραιότητας, ειδικά σε σενάρια τηλεφωνικής επικοινωνίας χαμηλής ποιότητας ήχου. Σε αρκετές περιπτώσεις, η προσαρμογή των συστημάτων στις ιδιαίτερες ανάγκες ενός οργανισμού απαιτεί επιπλέον κόστος ή τεχνικούς περιορισμούς που δεν επιτρέπουν πλήρη παραμετροποίηση (Google, 2024; Amazon, 2024).

**Συγκριτική Αποτίμηση και Θέση της Παρούσας Εργασίας**

Σε σύγκριση με τις παραπάνω προσεγγίσεις, η παρούσα εργασία διαφοροποιείται ουσιαστικά τόσο από ακαδημαϊκές όσο και από εμπορικές λύσεις. Αντί να βασίζεται σε σύνθετα και υπολογιστικά απαιτητικά μοντέλα, προτείνει μια ισορροπημένη προσέγγιση που συνδυάζει ικανοποιητική απόδοση με χαμηλές απαιτήσεις σε υπολογιστικούς πόρους. Παράλληλα, αποφεύγει την εξάρτηση από cloud υπηρεσίες, υλοποιώντας ένα πλήρως on-premise σύστημα. Ακόμα η υποστήριξη της ελληνικής γλώσσας και η δυνατότητα επανεκπαίδευσης του συστήματος από τον ίδιο τον οργανισμό-επιχείρηση αποτελούν σημαντικά πλεονεκτήματα έναντι τόσο των εμπορικών όσο και των περισσότερων ερευνητικών λύσεων. Η παρούσα εργασία, επομένως, τοποθετείται ως μια πρακτικά αξιοποιήσιμη, ευέλικτη και προσαρμόσιμη προσέγγιση ανάπτυξης τηλεφωνικών Voicebots για τον ελληνόφωνο χώρο.

## 2.4 Συνεισφορά της Εργασίας

Η παρούσα διπλωματική εργασία συνεισφέρει ουσιαστικά στον τομέα των τηλεφωνικών συστημάτων τεχνητής νοημοσύνης, προτείνοντας μια ολοκληρωμένη και πρακτικά εφαρμόσιμη λύση ανάπτυξης ελληνόφωνου Voicebot, η οποία γεφυρώνει το χάσμα μεταξύ θεωρητικής έρευνας και πραγματικής επιχειρησιακής εφαρμογής. Η συνεισφορά της εργασίας δεν περιορίζεται σε επίπεδο υλοποίησης, αλλά εκτείνεται και σε επίπεδο σχεδιασμού, αρχιτεκτονικής και τεχνολογικών επιλογών. Πρωταρχική συνεισφορά της εργασίας αποτελεί η υλοποίηση ενός πλήρως αυτόνομου (on-premise) τηλεφωνικού Voicebot, ο οποίος λειτουργεί χωρίς εξάρτηση από εμπορικές cloud υπηρεσίες ή τρίτους παρόχους. Η προσέγγιση αυτή διαφοροποιείται σημαντικά από τις περισσότερες σύγχρονες λύσεις, οι οποίες βασίζονται σε απομακρυσμένες υποδομές, και προσφέρει αυξημένο έλεγχο επί των δεδομένων, ενισχυμένη ασφάλεια και συμμόρφωση με απαιτήσεις προστασίας προσωπικών δεδομένων. Ιδιαίτερη σημασία έχει η υποστήριξη της ελληνικής γλώσσας σε όλα τα στάδια της αλυσίδας επεξεργασίας, δηλαδή στην αναγνώριση ομιλίας,

στην κατανόηση φυσικής γλώσσας και στη σύνθεση ομιλίας. Δεδομένου ότι η ελληνική γλώσσα παρουσιάζει μορφολογικές και φωνολογικές ιδιαιτερότητες, οι οποίες συχνά δεν καλύπτονται επαρκώς από εμπορικές πλατφόρμες, η ανάπτυξη ενός λειτουργικού ελληνόφωνου Voicebot αποτελεί σημαντική προσθήκη στο υπάρχον τεχνολογικό οικοσύστημα. Επιπλέον, η εργασία συνεισφέρει μέσω της υλοποίησης ενός προσαρμοσμένου συστήματος κατανόησης φυσικής γλώσσας (NLU), το οποίο μπορεί να εκπαιδεύεται και να επανεκπαιδεύεται εύκολα από τον ίδιο τον οργανισμό. Η δυνατότητα αυτή παρέχεται μέσω απλού web περιβάλλοντος διαχείρισης, επιτρέποντας την προσθήκη νέων παραδειγμάτων, την προσαρμογή των προθέσεων και τη συνεχή βελτίωση της απόδοσης του συστήματος χωρίς εξειδικευμένες γνώσεις μηχανικής μάθησης. Σημαντική συνεισφορά αποτελεί επίσης η υλοποίηση μηχανισμού διαλόγου πολλαπλών βημάτων, βασισμένου σε state machine, ο οποίος επιτρέπει τη διαχείριση σύνθετων τηλεφωνικών διαδικασιών, όπως η παραγγελιοληψία. Μέσω του μηχανισμού αυτού, το Voicebot είναι σε θέση να συλλέγει διαδοχικά πληροφορίες από τον χρήστη, να ελέγχει την εγκυρότητά τους και να ολοκληρώνει τη διαδικασία με συνεπή και ελεγχόμενο τρόπο. Η προσέγγιση αυτή αποδεικνύει ότι τα Voicebots μπορούν να υποστηρίξουν όχι μόνο απλές ερωτήσεις, αλλά και πιο σύνθετες επιχειρησιακές ροές. Η ενσωμάτωση του Voicebot σε πραγματικό τηλεφωνικό περιβάλλον μέσω του Asterisk και του VICIdial αποτελεί ακόμη μία σημαντική συνεισφορά της εργασίας. Σε αντίθεση με πολλές προσεγγίσεις που παραμένουν σε πειραματικό ή προσομοιωμένο επίπεδο, το προτεινόμενο σύστημα έχει σχεδιαστεί και δοκιμαστεί σε συνθήκες πραγματικής τηλεφωνίας, λαμβάνοντας υπόψη περιορισμούς όπως η ποιότητα ήχου, ο χρόνος απόκρισης και η σταθερότητα λειτουργίας. Τέλος, η εργασία προσφέρει ένα ολοκληρωμένο παράδειγμα αρχιτεκτονικής σχεδίασης τηλεφωνικού Voicebot, το οποίο μπορεί να αποτελέσει βάση για μελλοντική έρευνα ή για πρακτική αξιοποίηση από οργανισμούς και επιχειρήσεις. Η συνδυαστική προσέγγιση θεωρίας, σχεδιασμού και υλοποίησης καθιστά τη συνεισφορά της εργασίας σημαντική τόσο σε ακαδημαϊκό όσο και σε εφαρμοσμένο επίπεδο.

## 2.5 Συνοπτική Σύνοψη του Κεφαλαίου

Στο παρόν κεφάλαιο πραγματοποιήθηκε εκτενής ανάλυση του αντικειμένου των τηλεφωνικών Voicebots και του θεωρητικού πλαισίου μέσα στο οποίο εντάσσεται η παρούσα διπλωματική εργασία. Αρχικά, παρουσιάστηκε η εξέλιξη των συστημάτων αυτόματης τηλεφωνικής εξυπηρέτησης και η μετάβαση από τα παραδοσιακά συστήματα IVR σε σύγχρονα διαλογικά συστήματα βασισμένα στη φυσική ομιλία, αναδεικνύοντας τις ανάγκες που οδήγησαν στην ανάπτυξη ευφυών Voicebots.

Στη συνέχεια, αναλύθηκαν οι βασικές τεχνολογικές συνιστώσες που απαρτίζουν ένα σύγχρονο Voicebot, δηλαδή η αυτόματη αναγνώριση ομιλίας (Speech-to-Text), η κατανόηση φυσικής γλώσσας (Natural Language Understanding), η σύνθεση ομιλίας (Text-to-Speech) και η τηλεφωνική ενσωμάτωση μέσω συστημάτων όπως το Asterisk και το VICIdial. Η παρουσίαση αυτή ανέδειξε τη σημασία της ορθής συνεργασίας των επιμέρους υποσυστημάτων για την επίτευξη φυσικής και αξιόπιστης τηλεφωνικής αλληλεπίδρασης. Παράλληλα, εξετάστηκαν σχετικές ακαδημαϊκές εργασίες και εμπορικές πλατφόρμες ανάπτυξης Voicebots, επισημαίνοντας τα πλεονεκτήματα αλλά και τους περιορισμούς τους,

ιδιαίτερα όσον αφορά την εξάρτηση από cloud υποδομές και την περιορισμένη υποστήριξη της ελληνικής γλώσσας. Μέσα από τη συγκριτική αυτή αποτίμηση, αναδείχθηκε η ανάγκη για αυτόνομες, προσαρμόσιμες και ελληνόφωνες λύσεις τηλεφωνικής εξυπηρέτησης.

Τέλος, παρουσιάστηκε η συνεισφορά της παρούσας εργασίας, η οποία έγκειται στην υλοποίηση ενός πλήρως on-premise ελληνόφωνου Voicebot, ενσωματωμένου σε πραγματικό τηλεφωνικό περιβάλλον, με δυνατότητα επανεκπαίδευσης, διαχείρισης διαλόγου πολλαπλών βημάτων και πρακτικής αξιοποίησης σε επιχειρησιακές συνθήκες.

Η θεωρητική και συγκριτική ανάλυση του κεφαλαίου αυτού θέτει τις βάσεις για το επόμενο κεφάλαιο, στο οποίο παρουσιάζεται η ανάλυση και ο σχεδιασμός της αρχιτεκτονικής του προτεινόμενου συστήματος, καθώς και ο τρόπος με τον οποίο οι θεωρητικές έννοιες που αναπτύχθηκαν μεταφράζονται σε συγκεκριμένες σχεδιαστικές και τεχνικές επιλογές.

ΚΕΦΑΛΑΙΟ 3

# Ανάλυση και σχεδίαση

## 3.1 Αρχιτεκτονική του Συστήματος

Η αρχιτεκτονική του Voicebot βασίζεται σε ένα σύνολο διακριτών αλλά στενά συνεργαζόμενων υποσυστημάτων, τα οποία συνθέτουν μια ολοκληρωμένη αλυσίδα επεξεργασίας τηλεφωνικής αλληλεπίδρασης. Ο σχεδιασμός ακολουθεί τις αρχές της αρθρωτής (modular) αρχιτεκτονικής, με διαχωρισμό αρμοδιοτήτων μεταξύ των επιμέρους τμημάτων, ώστε να επιτυγχάνεται αυξημένη ευελιξία και ευκολία συντήρησης. Βασικός στόχος του αρχιτεκτονικού σχεδιασμού είναι η υποστήριξη φυσικής συνομιλίας σε πραγματικό χρόνο, λαμβάνοντας υπόψη τους περιορισμούς και τις ιδιαιτερότητες της τηλεφωνικής επικοινωνίας. Το σύστημα σχεδιάστηκε έτσι ώστε να διασφαλίζει αξιόπιστη λειτουργία, χαμηλό χρόνο απόκρισης (latency) και ανθεκτικότητα σε σφάλματα, ενώ παράλληλα επιτρέπει την ανεξάρτητη βελτίωση ή αντικατάσταση κάθε υποσυστήματος χωρίς να επηρεάζεται η συνολική λειτουργία του Voicebot (McTear, 2010). Στο διάγραμμα συνολικής ροής (conceptual representation) αποτυπώνεται η διαδρομή των δεδομένων από τη στιγμή της εισερχόμενης κλήσης έως την ολοκλήρωση της τηλεφωνικής αλληλεπίδρασης. Η ροή αυτή οργανώνεται σε διακριτά στάδια, τα οποία εκτελούνται διαδοχικά και συνεργάζονται μέσω σαφώς ορισμένων διεπαφών.

**1. Τηλεφωνική Κλήση (Asterisk / VICIdial)**

Η αλληλεπίδραση ξεκινά με την εισερχόμενη τηλεφωνική κλήση του χρήστη προς έναν αριθμό που τερματίζει στο τηλεφωνικό κέντρο Asterisk. Το Asterisk λειτουργεί ως ο πυρήνας της τηλεφωνικής υποδομής, αναλαμβάνοντας τη διαχείριση της κλήσης, ενώ το περιβάλλον VICIdial καθορίζει τη δρομολόγηση και τη λογική χειρισμού της. Βάσει της παραμετροποίησης του VICIdial, η κλήση κατευθύνεται στο Voicebot και ενεργοποιείται το αντίστοιχο AGI script.

**2. AGI Script (vici_voicebot.py)**

Το AGI script αποτελεί τον κεντρικό συντονιστή της αρχιτεκτονικής και λειτουργεί ως «κόμβος ελέγχου» της συνολικής ροής. Μέσω της διεπαφής AGI, το script αποκτά πλήρη έλεγχο της κλήσης και είναι υπεύθυνο για:

- την ηχογράφηση της φωνής του χρήστη,
- τη διαχείριση των αρχείων ήχου,
- την αποστολή των δεδομένων στα υποσυστήματα επεξεργασίας,
- την επιλογή της κατάλληλης απάντησης,
- την αναπαραγωγή φωνητικών μηνυμάτων στον χρήστη.

Η επιλογή χρήσης AGI επιτρέπει την υλοποίηση πολύπλοκης λογικής σε γλώσσα Python, εκτός του τηλεφωνικού dialplan, προσφέροντας σημαντική ευελιξία και δυνατότητα επέκτασης.

**3. Speech-to-Text (Whisper)**

Το ηχογραφημένο αρχείο ομιλίας αποστέλλεται στο υποσύστημα αναγνώρισης ομιλίας, όπου μετατρέπεται σε κείμενο με τη χρήση του μοντέλου Whisper. Στην παρούσα υλοποίηση χρησιμοποιείται η έκδοση *small* του μοντέλου, η οποία επιλέχθηκε ως βέλτιστος συμβιβασμός μεταξύ ακρίβειας αναγνώρισης και υπολογιστικού κόστους. Η επιλογή αυτή είναι ιδιαίτερα σημαντική για τηλεφωνικά περιβάλλοντα, όπου απαιτείται σταθερή απόδοση σε πραγματικό χρόνο και περιορισμένοι υπολογιστικοί πόροι (Radford et al., 2023).

**4. Natural Language Understanding (NLU)**

Το παραγόμενο κείμενο εισόδου προωθείται στο υποσύστημα κατανόησης φυσικής γλώσσας. Το NLU μοντέλο, βασισμένο σε PyTorch και σε αναπαράσταση Bag-of-Words με νευρωνικό δίκτυο τύπου MLP, αναλύει το κείμενο με στόχο τον εντοπισμό της πρόθεσης του χρήστη (intent) και την εξαγωγή βασικών πληροφοριών (slots). Το αποτέλεσμα του σταδίου αυτού αποτελεί κρίσιμο στοιχείο για τη λογική απόφαση του συστήματος και καθορίζει την επόμενη ενέργεια στη ροή του διαλόγου (Manning et al., 2008; Jurafsky & Martin, 2023).

**5. Διαχείριση Διαλόγου (Dialog Management – State Machine)**

Με βάση την ανιχνευμένη πρόθεση και το τρέχον συμφραζόμενο της συνομιλίας, το υποσύστημα διαχείρισης διαλόγου αποφασίζει το επόμενο βήμα της αλληλεπίδρασης. Ο μηχανισμός αυτός υλοποιείται μέσω state machine, επιτρέποντας τη σαφή οριοθέτηση καταστάσεων και μεταβάσεων. Η προσέγγιση αυτή είναι ιδιαίτερα κρίσιμη για σύνθετες διεργασίες, όπως η παραγγελιοληψία πολλαπλών βημάτων, όπου απαιτείται διατήρηση κατάστασης και έλεγχος της ροής της συνομιλίας (McTear, 2010).

**6. Text-to-Speech (TTS)**

Η απάντηση που επιλέγεται από το σύστημα μετατρέπεται σε συνθετική ομιλία μέσω του υποσυστήματος Text-to-Speech. Η χρήση του gTTS επιτρέπει την παραγωγή ελληνικής φωνής, η οποία στη συνέχεια μετασχηματίζεται σε μορφή συμβατή με τον Asterisk. Το παραγόμενο ηχητικό αρχείο αποστέλλεται για αναπαραγωγή στον καλούντα, ολοκληρώνοντας τον κύκλο αλληλεπίδρασης (Taylor, 2009).

**7. Καταγραφή Δεδομένων και Διαχείριση Πληροφορίας**

Σε περιπτώσεις ολοκληρωμένων διεργασιών, όπως η καταχώριση νέας παραγγελίας, τα δεδομένα που συλλέγονται κατά τη διάρκεια της συνομιλίας αποθηκεύονται σε δομημένη μορφή (CSV). Τα δεδομένα αυτά καθίστανται διαθέσιμα μέσω του web περιβάλλοντος διαχείρισης (Django admin dashboard), επιτρέποντας την εύκολη εποπτεία και αξιοποίηση των αποτελεσμάτων του Voicebot. Η αρχιτεκτονική του συστήματος υλοποιεί μια καθαρή και επεκτάσιμη αλυσίδα επεξεργασίας τηλεφωνικής αλληλεπίδρασης. Η modular προσέγγιση επιτρέπει την ανεξάρτητη αναβάθμιση ή αντικατάσταση επιμέρους υποσυστημάτων, όπως το STT ή το NLU, χωρίς να απαιτείται ανασχεδιασμός του συνόλου, καθιστώντας το σύστημα κατάλληλο για μελλοντικές βελτιώσεις και επεκτάσεις.

## 3.2 Υποσύστημα Τηλεφωνίας (Asterisk / VICIdial)

Το τηλεφωνικό υπόβαθρο του προτεινόμενου συστήματος βασίζεται στο Asterisk PBX, το οποίο αποτελεί τον κεντρικό πυρήνα της τηλεφωνικής λειτουργικότητας και της διαχείρισης κλήσεων. Το Asterisk είναι ένα ανοικτού κώδικα τηλεφωνικό σύστημα (Private Branch Exchange), το οποίο χρησιμοποιείται ευρέως τόσο σε μικρές όσο και σε μεγάλης κλίμακας τηλεφωνικές υποδομές, λόγω της ευελιξίας, της επεκτασιμότητας και της δυνατότητας πλήρους παραμετροποίησής του. Στο πλαίσιο της παρούσας εργασίας, το Asterisk αναλαμβάνει βασικές αλλά κρίσιμες λειτουργίες, όπως η απάντηση εισερχόμενων κλήσεων, η καταγραφή της ομιλίας του καλούντος, η αναπαραγωγή ηχητικών αρχείων και η διασύνδεση με εξωτερικά προγράμματα μέσω της διεπαφής AGI (Asterisk Gateway Interface) (Asterisk, 2024).

Η επιλογή του Asterisk ως τηλεφωνικού πυρήνα επιτρέπει την υλοποίηση του Voicebot σε πραγματικό τηλεφωνικό περιβάλλον, χωρίς τη χρήση προσομοιώσεων ή εικονικών κλήσεων. Επιπλέον, το Asterisk παρέχει χαμηλού επιπέδου έλεγχο της τηλεφωνικής ροής, επιτρέποντας στο σύστημα να ανταποκρίνεται άμεσα σε γεγονότα της κλήσης, όπως η έναρξη, η παύση ή ο τερματισμός της συνομιλίας. Η δυνατότητα αυτή είναι ιδιαίτερα σημαντική για εφαρμογές Voicebot, όπου απαιτείται ακριβής συγχρονισμός μεταξύ ηχογράφησης, επεξεργασίας και αναπαραγωγής φωνής.

Το VICIdial λειτουργεί ως επέκταση του Asterisk και προσθέτει δυνατότητες επιπέδου call center, οι οποίες διευκολύνουν τη διαχείριση μεγάλου όγκου κλήσεων και την ενσωμάτωση του Voicebot σε πραγματικές επιχειρησιακές ροές. Συγκεκριμένα, το VICIdial παρέχει μηχανισμούς για τη δημιουργία εισερχόμενων ουρών, τη διαχείριση καμπανιών, τον έλεγχο της δρομολόγησης κλήσεων και την εποπτεία της τηλεφωνικής λειτουργίας μέσω γραφικού περιβάλλοντος διαχείρισης. Μέσα σε αυτό το πλαίσιο, το Voicebot ενσωματώνεται ως ειδική διαδρομή κλήσης (voicebot-in context), επιτρέποντας την αυτόματη εξυπηρέτηση εισερχόμενων κλήσεων χωρίς ανθρώπινη παρέμβαση.

Η αρχιτεκτονική αυτή καθιστά δυνατή τη συνύπαρξη του Voicebot με ανθρώπινους εκπροσώπους στο ίδιο τηλεφωνικό σύστημα. Ανάλογα με τη ρύθμιση της καμπάνιας, μια κλήση μπορεί να εξυπηρετηθεί εξ ολοκλήρου από το Voicebot ή να μεταφερθεί σε ανθρώπινο agent, εφόσον αυτό απαιτείται. Η δυνατότητα αυτή προσφέρει σημαντική ευελιξία και επιτρέπει τη σταδιακή ενσωμάτωση του Voicebot σε υπάρχουσες τηλεφωνικές υποδομές. Η αλληλεπίδραση του Voicebot με το τηλεφωνικό σύστημα υλοποιείται μέσω της διεπαφής AGI, η οποία επιτρέπει σε εξωτερικά προγράμματα, γραμμένα σε γλώσσες όπως η Python, να ελέγχουν τη ροή της κλήσης. Μέσω βασικών AGI εντολών, όπως οι RECORD FILE, STREAM FILE, EXEC Playback και VERBOSE, το Python script αποκτά πλήρη έλεγχο της τηλεφωνικής αλληλεπίδρασης. Οι εντολές αυτές επιτρέπουν την καταγραφή της φωνής του χρήστη, την αναπαραγωγή ηχητικών απαντήσεων, την εκτέλεση τηλεφωνικών εφαρμογών και την καταγραφή διαγνωστικών μηνυμάτων, αντίστοιχα.
Η χρήση AGI αποτελεί συνειδητή σχεδιαστική επιλογή, καθώς επιτρέπει την αποσύνδεση της επιχειρησιακής λογικής του Voicebot από το τηλεφωνικό dialplan. Με τον τρόπο αυτό, η σύνθετη λογική επεξεργασίας ομιλίας, κατανόησης φυσικής γλώσσας και διαχείρισης διαλόγου υλοποιείται σε περιβάλλον υψηλού επιπέδου, διατηρώντας το τηλεφωνικό σύστημα απλό και ευέλικτο. Παράλληλα, η προσέγγιση αυτή διευκολύνει τη συντήρηση και την επέκταση του συστήματος, καθώς οι αλλαγές στη λογική του Voicebot δεν απαιτούν τροποποιήσεις στον πυρήνα της τηλεφωνικής υποδομής (Sommerville, 2016; Asterisk, 202).
Το υποσύστημα τηλεφωνίας αποτελεί τη θεμελιώδη βάση πάνω στην οποία οικοδομείται το προτεινόμενο Voicebot. Ο συνδυασμός Asterisk και VICIdial προσφέρει ένα σταθερό, αξιόπιστο και ευρέως δοκιμασμένο περιβάλλον τηλεφωνικής λειτουργίας, ικανό να υποστηρίξει την απαιτητική αλυσίδα επεξεργασίας φυσικής ομιλίας και να εξασφαλίσει την απρόσκοπτη λειτουργία του συστήματος σε πραγματικές συνθήκες.

## 3.3 Υποσύστημα Speech-to-Text (Whisper)

Το υποσύστημα αναγνώρισης ομιλίας (Speech-to-Text – STT) αποτελεί ένα από τα πλέον κρίσιμα τμήματα του προτεινόμενου Voicebot, καθώς λειτουργεί ως το σημείο εισόδου της πληροφορίας από τον χρήστη προς το σύστημα. Η ποιότητα και η αξιοπιστία της μετατροπής της ανθρώπινης ομιλίας σε κείμενο επηρεάζουν άμεσα τη συνολική απόδοση του Voicebot, δεδομένου ότι τυχόν σφάλματα σε αυτό το στάδιο μεταφέρονται και επηρεάζουν όλα τα επόμενα υποσυστήματα. Στην υλοποίηση χρησιμοποιείται το μοντέλο Whisper στην έκδοση *small*, το οποίο βασίζεται σε αρχιτεκτονική deep neural networks και έχει εκπαιδευτεί σε μεγάλο όγκο πολυγλωσσικών δεδομένων. Το μοντέλο εκτελείται σε λειτουργία *float32* και αξιοποιεί αποκλειστικά τον επεξεργαστή (CPU) του VICIdial server, χωρίς χρήση επιτάχυνσης μέσω GPU. Η επιλογή αυτή πραγματοποιήθηκε με γνώμονα την πρακτική εφαρμογή του συστήματος σε πραγματικό περιβάλλον τηλεφωνικού κέντρου, όπου η ύπαρξη εξειδικευμένου hardware δεν μπορεί να θεωρείται δεδομένη (Radford et al., 2023).
Παρά τους περιορισμένους υπολογιστικούς πόρους του server, το Whisper Small επιδεικνύει ιδιαίτερα ικανοποιητική ακρίβεια στην ελληνική γλώσσα, καθιστώντας το κατάλληλο για εφαρμογές τηλεφωνικής αλληλεπίδρασης. Η επιλογή της συγκεκριμένης έκδοσης του μοντέλου αποτελεί έναν συνειδητό συμβιβασμό μεταξύ ακρίβειας αναγνώρισης και χρόνου απόκρισης, στοιχεία τα οποία είναι καθοριστικά για τη διατήρηση φυσικού ρυθμού συνομιλίας.

Η τηλεφωνική ομιλία, σε σύγκριση με την ομιλία υψηλής ποιότητας (π.χ. από μικρόφωνα studio), παρουσιάζει σημαντικές προκλήσεις για τα συστήματα αναγνώρισης ομιλίας. Συγκεκριμένα:

- περιορίζεται σε εύρος συχνοτήτων 300–3400 Hz, γεγονός που αφαιρεί φωνητικές λεπτομέρειες.
- περιλαμβάνει θόρυβο γραμμής και παρεμβολές.
- υπόκειται σε συμπίεση μέσω codecs όπως G.711 ή GSM, οι οποίοι αλλοιώνουν το αρχικό ηχητικό σήμα.

Οι παραπάνω παράγοντες καθιστούν την αναγνώριση ομιλίας σε τηλεφωνικά περιβάλλοντα ιδιαίτερα απαιτητική. Το Whisper, ωστόσο, ενσωματώνει μηχανισμούς robust speech recognition, οι οποίοι επιτρέπουν στο μοντέλο να διατηρεί υψηλά επίπεδα ακρίβειας ακόμη και σε συνθήκες υποβαθμισμένου ήχου. Η εκπαίδευσή του σε δεδομένα με διαφορετικά επίπεδα ποιότητας και θορύβου συμβάλλει καθοριστικά στην ανθεκτικότητά του. Στο πλαίσιο της παρούσας υλοποίησης, η είσοδος στο υποσύστημα STT παρέχεται ως αρχείο WAV με ρυθμό δειγματοληψίας 16 kHz. Το αρχείο αυτό δημιουργείται δυναμικά από το AGI script, το οποίο αρχικά ηχογραφεί τη φωνή του χρήστη σε τηλεφωνική ποιότητα και στη συνέχεια πραγματοποιεί μετατροπή της συχνότητας δειγματοληψίας. Η προσαρμογή αυτή είναι απαραίτητη, καθώς το Whisper απαιτεί είσοδο συγκεκριμένων τεχνικών χαρακτηριστικών για τη βέλτιστη λειτουργία του.
Η ενσωμάτωση του Whisper στο Voicebot πραγματοποιείται με τρόπο πλήρως διαφανή για το τηλεφωνικό σύστημα. Το AGI script αναλαμβάνει την κλήση του μοντέλου, τη διαχείριση των αρχείων ήχου και την παραλαβή του αποτελέσματος της μεταγραφής, το οποίο προωθείται άμεσα στο υποσύστημα κατανόησης φυσικής γλώσσας. Με τον τρόπο αυτό, το STT λειτουργεί ως ανεξάρτητο αλλά κρίσιμο δομικό στοιχείο της συνολικής αρχιτεκτονικής. Συνολικά, το υποσύστημα Speech-to-Text παρέχει την απαραίτητη γέφυρα μεταξύ της ανθρώπινης ομιλίας και της υπολογιστικής επεξεργασίας, επιτρέποντας στο Voicebot να λειτουργεί αποτελεσματικά σε πραγματικό τηλεφωνικό περιβάλλον. Η επιλογή του Whisper Small και η προσαρμογή του στις ιδιαιτερότητες της τηλεφωνικής επικοινωνίας συνεισφέρουν καθοριστικά στη σταθερότητα και τη χρηστικότητα του συστήματος.

## 3.4 Υποσύστημα NLU (Intent Classifier)

Το υποσύστημα κατανόησης φυσικής γλώσσας (Natural Language Understanding – NLU) αποτελεί κρίσιμο στοιχείο της συνολικής αρχιτεκτονικής του Voicebot, καθώς είναι υπεύθυνο για τη μετατροπή του απλού κειμένου που προκύπτει από το υποσύστημα αναγνώρισης ομιλίας σε ουσιαστική πληροφορία, κατανοητή από το σύστημα. Μέσω του NLU, το Voicebot αποκτά τη δυνατότητα να ερμηνεύει το νόημα της εισερχόμενης πρότασης και να προσδιορίζει την πρόθεση (intent) του χρήστη, γεγονός που καθορίζει άμεσα την επόμενη ενέργεια στον διάλογο (Jurafsky & Martin, 2023).
Η προσέγγιση που ακολουθήθηκε στην παρούσα εργασία βασίζεται σε συνδυασμό αναπαράστασης κειμένου τύπου Bag-of-Words (BoW) και νευρωνικού δικτύου τύπου Multilayer Perceptron (MLP). Η επιλογή αυτή πραγματοποιήθηκε με γνώμονα τη χαμηλή υπολογιστική πολυπλοκότητα, την ταχύτητα εκτέλεσης και τη δυνατότητα πλήρους ελέγχου και επανεκπαίδευσης του μοντέλου σε on-premise περιβάλλον. Παρότι πιο σύνθετες

μέθοδοι, όπως τα προεκπαιδευμένα γλωσσικά μοντέλα (transformers), μπορούν να προσφέρουν υψηλότερη ακρίβεια, η συγκεκριμένη λύση κρίθηκε επαρκής και αποδοτική για τις απαιτήσεις ενός τηλεφωνικού Voicebot σε πραγματικό χρόνο.

Η αναπαράσταση Bag-of-Words που χρησιμοποιείται στο σύστημα περιλαμβάνει στάδια προεπεξεργασίας ειδικά προσαρμοσμένα στην ελληνική γλώσσα. Συγκεκριμένα:

- εφαρμόζεται ελληνική κανονικοποίηση κειμένου (lowercasing, απομάκρυνση ειδικών χαρακτήρων).
- πραγματοποιείται αφαίρεση τόνων, ώστε να μειώνεται η διασπορά των λέξεων στο λεξιλόγιο.
- παρέχεται δυνατότητα χρήσης bigrams και character-ngrams, ενισχύοντας την αναγνώριση σύντομων φράσεων και μορφολογικών παραλλαγών.
- το τελικό αποτέλεσμα μετατρέπεται σε sparse διανύσματα υψηλής διάστασης, κατάλληλα για είσοδο σε νευρωνικό δίκτυο.

Η συγκεκριμένη μορφή αναπαράστασης επιτρέπει στο σύστημα να αποτυπώνει την παρουσία και τη συχνότητα λέξεων ή φραστικών μοτίβων, χωρίς να απαιτείται πολύπλοκη γλωσσική ανάλυση. Η απλότητα αυτή συνεισφέρει σημαντικά στη σταθερότητα και την ταχύτητα του συστήματος, στοιχεία κρίσιμα για τηλεφωνικές εφαρμογές (Manning et al., 2008).
Το νευρωνικό δίκτυο MLP που χρησιμοποιείται για την ταξινόμηση των intents αποτελείται από διαδοχικά γραμμικά επίπεδα (linear layers), συνάρτηση ενεργοποίησης ReLU και μηχανισμό Dropout για την αποφυγή υπερπροσαρμογής. Το τελικό επίπεδο εξόδου εφαρμόζει συνάρτηση softmax, η οποία επιστρέφει πιθανότητες για κάθε διαθέσιμη κατηγορία πρόθεσης. Η αρχιτεκτονική αυτή επιλέχθηκε λόγω της απλότητάς της και της ικανότητάς της να αποδίδει ικανοποιητικά σε προβλήματα ταξινόμησης μικρής και μεσαίας κλίμακας.
Η εκπαίδευση του μοντέλου πραγματοποιείται μέσω του script train_intents.py, το οποίο αναλαμβάνει ολόκληρη τη διαδικασία προετοιμασίας και εκμάθησης. Το script διαχειρίζεται τη φόρτωση των δεδομένων εκπαίδευσης από αρχεία CSV, την κατασκευή του λεξιλογίου, την εκπαίδευση του μοντέλου για συγκεκριμένο αριθμό εποχών, καθώς και την αποθήκευση των παραγόμενων παραμέτρων. Με τον τρόπο αυτό, το μοντέλο μπορεί να επαναχρησιμοποιηθεί άμεσα από το AGI script κατά τη φάση της πρόβλεψης, χωρίς πρόσθετο υπολογιστικό κόστος. Ιδιαίτερη σημασία δίνεται στη διαχείριση των δεδομένων εκπαίδευσης μέσω του web περιβάλλοντος Django. Το Django UI επιτρέπει τη δυναμική προσθήκη, τροποποίηση και διαγραφή παραδειγμάτων εκπαίδευσης, καθώς και την επανεκπαίδευση του μοντέλου με το πάτημα ενός κουμπιού. Η προσέγγιση αυτή καθιστά το σύστημα προσβάσιμο ακόμη και σε χρήστες χωρίς εξειδικευμένες γνώσεις machine learning, επιτρέποντας την προσαρμογή του Voicebot στις ανάγκες κάθε οργανισμού.
Το υποσύστημα NLU λειτουργεί ως ο «εγκέφαλος» της λογικής του Voicebot, μεταφράζοντας την ανθρώπινη γλώσσα σε δομημένες προθέσεις και ενέργειες. Η επιλογή ενός απλού αλλά αποτελεσματικού μοντέλου, σε συνδυασμό με τη δυνατότητα εύκολης επανεκπαίδευσης, εξασφαλίζει την αξιοπιστία, την επεκτασιμότητα και την πρακτική χρηστικότητα του συστήματος σε πραγματικές τηλεφωνικές συνθήκες.

## 3.5 Σχεδιασμός Συστήματος Διαλόγου (Dialog Management)

Για την υποστήριξη φυσικού, ελεγχόμενου και προβλέψιμου διαλόγου, σχεδιάστηκε και υλοποιήθηκε μηχανισμός διαχείρισης συνομιλίας βασισμένος σε state machine. Ο μηχανισμός αυτός αποτελεί το ενδιάμεσο επίπεδο μεταξύ της κατανόησης της πρόθεσης του χρήστη (NLU) και της παραγωγής της τελικής απόκρισης, διασφαλίζοντας ότι η συνομιλία εξελίσσεται με λογική συνοχή και σύμφωνα με τους στόχους του συστήματος. Η επιλογή state machine ως βασικό μοντέλο διαχείρισης διαλόγου πραγματοποιήθηκε με γνώμονα την απλότητα, την αξιοπιστία και τον πλήρη έλεγχο της ροής της συνομιλίας. Σε αντίθεση με πιο σύνθετες προσεγγίσεις που βασίζονται σε στατιστικά ή ενισχυτικά μοντέλα, η state machine επιτρέπει τον σαφή ορισμό καταστάσεων, γεγονός που είναι ιδιαίτερα σημαντικό σε τηλεφωνικά περιβάλλοντα, όπου απαιτείται περιορισμός λαθών (McTear, 2010).

Κάθε κατάσταση της state machine αντιστοιχεί σε ένα συγκεκριμένο στάδιο της συνομιλίας και διαθέτει:

- προκαθορισμένο φωνητικό prompt, το οποίο καθοδηγεί τον χρήστη.
- κανόνες επικύρωσης (validation rules) για τον έλεγχο της εισερχόμενης πληροφορίας.
- σαφώς ορισμένες μεταβάσεις προς επόμενες καταστάσεις, ανάλογα με την απάντηση του χρήστη και το αποτέλεσμα του NLU.
- πρόσβαση σε κοινό context data, στο οποίο αποθηκεύονται τα στοιχεία που συλλέγονται κατά τη διάρκεια της συνομιλίας.

Η χρήση context data επιτρέπει στο σύστημα να «θυμάται» πληροφορίες που έχουν ήδη δοθεί από τον χρήστη, όπως το όνομα, το τηλέφωνο ή το προϊόν ενδιαφέροντος. Με τον τρόπο αυτό, το Voicebot μπορεί να διατηρεί συνοχή στη συνομιλία και να αποφεύγει επαναλαμβανόμενες ερωτήσεις, βελτιώνοντας τη φυσικότητα της αλληλεπίδρασης. Ενδεικτικό παράδειγμα εφαρμογής του μηχανισμού διαλόγου αποτελεί η ροή «Νέα Παραγγελία». Σε αυτή την περίπτωση, το σύστημα ξεκινά από την ανίχνευση της πρόθεσης new_order και στη συνέχεια μεταβαίνει διαδοχικά σε καταστάσεις συλλογής στοιχείων, όπως το ονοματεπώνυμο του πελάτη, το τηλέφωνο επικοινωνίας, το επιθυμητό προϊόν και η ποσότητα. Αφού συγκεντρωθούν όλα τα απαραίτητα δεδομένα, το σύστημα μεταβαίνει σε κατάσταση επιβεβαίωσης, όπου επαναλαμβάνει συνοπτικά τα στοιχεία στον χρήστη πριν την τελική καταχώριση της παραγγελίας (McTear, 2010; Jurafsky & Martin, 2023).

Η συγκεκριμένη προσέγγιση επιτρέπει την υλοποίηση σύνθετων conversational flows με ελεγχόμενο τρόπο, μειώνοντας την πιθανότητα σφαλμάτων και ασάφειας στη συνομιλία. Επιπλέον, η state machine διευκολύνει την ανίχνευση και τη διαχείριση εξαιρέσεων, όπως ακατανόητες απαντήσεις ή μη έγκυρα δεδομένα, επιτρέποντας στο σύστημα να επαναλαμβάνει ερωτήσεις ή να κατευθύνει εκ νέου τον χρήστη. Η επιλογή υλοποίησης του dialog management χωρίς εξάρτηση από εξωτερικά frameworks, όπως το Dialogflow ή το Rasa, αποτελεί συνειδητή σχεδιαστική απόφαση. Αν και τα frameworks αυτά προσφέρουν πλούσιες λειτουργίες, συνεπάγονται αυξημένη πολυπλοκότητα, εξαρτήσεις από cloud υπηρεσίες ή επιπλέον υπολογιστικό κόστος. Αντίθετα, η προτεινόμενη λύση προσφέρει πλήρη έλεγχο της λογικής διαλόγου, απλότητα υλοποίησης και ευκολία ενσωμάτωσης στο

υπάρχον τηλεφωνικό περιβάλλον. Ο μηχανισμός διαχείρισης διαλόγου αποτελεί βασικό δομικό στοιχείο του Voicebot, επιτρέποντας τη φυσική και αξιόπιστη εξέλιξη της συνομιλίας. Η χρήση state machine, σε συνδυασμό με context data και σαφώς ορισμένες μεταβάσεις, εξασφαλίζει ότι το σύστημα μπορεί να χειριστεί αποτελεσματικά σύνθετες διεργασίες, διατηρώντας παράλληλα χαμηλή πολυπλοκότητα και υψηλή πρόβλεψη.

## 3.6 Υποσύστημα Text-to-Speech (gTTS)

Το υποσύστημα Text-to-Speech (TTS) είναι υπεύθυνο για τη μετατροπή του κειμένου σε συνθετική ανθρώπινη ομιλία, επιτρέποντας στο Voicebot να επικοινωνεί με τον χρήστη με τρόπο φυσικό και κατανοητό. Το στάδιο αποτελεί το τελικό βήμα της αλυσίδας επεξεργασίας και επηρεάζει άμεσα την εμπειρία του χρήστη, καθώς η ποιότητα και η ταχύτητα της παραγόμενης φωνής καθορίζουν την ροή της συνομιλίας (Taylor, 2009).

Στην υλοποίηση χρησιμοποιείται το εργαλείο gTTS (Google Text-to-Speech), το οποίο μετατρέπει το κείμενο απόκρισης σε φυσική ελληνική φωνή. Η επιλογή του gTTS βασίστηκε στην ικανοποιητική ποιότητα φωνής που προσφέρει για την ελληνική γλώσσα, καθώς και στη σχετικά χαμηλή πολυπλοκότητα ενσωμάτωσής του στο σύστημα. Παρότι το gTTS δεν αποτελεί πλήρως παραμετροποιήσιμο συνθετικό μοντέλο φωνής, η απόδοσή του κρίθηκε επαρκής για τις ανάγκες τηλεφωνικής επικοινωνίας, όπου η καθαρότητα και η κατανόηση της ομιλίας υπερέχουν έναντι της εκφραστικότητας. Η διαδικασία παραγωγής ομιλίας περιλαμβάνει διαδοχικά στάδια. Αρχικά, το κείμενο που έχει επιλεγεί από το υποσύστημα διαχείρισης διαλόγου αποστέλλεται στο gTTS, το οποίο παράγει αρχείο ήχου σε μορφή υψηλής ποιότητας (συνήθως MP3). Στη συνέχεια, το αρχείο αυτό μετατρέπεται σε μορφή συμβατή με το τηλεφωνικό περιβάλλον του Asterisk, όπως GSM ή SLIN, με τη χρήση του εργαλείου *sox*. Η μετατροπή αυτή είναι απαραίτητη, καθώς το Asterisk απαιτεί συγκεκριμένα τεχνικά χαρακτηριστικά για την ορθή αναπαραγωγή ήχου κατά τη διάρκεια μιας κλήσης. Η επιλογή της μορφής GSM εξασφαλίζει συμβατότητα με τα συνήθη codecs τηλεφωνίας και μειώνει το μέγεθος των αρχείων, γεγονός που συμβάλλει στη γρήγορη αναπαραγωγή και στη μείωση του χρόνου καθυστέρησης. Το τελικό ηχητικό αρχείο αποθηκεύεται προσωρινά στον server και αναπαράγεται στον χρήστη μέσω εντολών AGI του Asterisk, ολοκληρώνοντας τον κύκλο αλληλεπίδρασης.
Η ταχύτητα παραγωγής ήχου αποτελεί κρίσιμο παράγοντα για τη διατήρηση φυσικού ρυθμού διαλόγου. Σε τηλεφωνικές εφαρμογές, ακόμη και μικρές καθυστερήσεις μπορούν να γίνουν αντιληπτές από τον χρήστη και να διαταράξουν τη ροή της συνομιλίας. Το gTTS ανταποκρίνεται ικανοποιητικά στις απαιτήσεις του συστήματος, παρέχοντας γρήγορη παραγωγή φωνής για σύντομα και μεσαίου μήκους μηνύματα, τα οποία είναι συνήθη σε σενάρια Voicebot.
Επίσης το υποσύστημα TTS υποστηρίζει τόσο σταθερές όσο και δυναμικά παραγόμενες απαντήσεις. Στην πρώτη περίπτωση, το Voicebot αναπαράγει προκαθορισμένα μηνύματα, όπως χαιρετισμούς ή γενικές πληροφορίες. Στη δεύτερη περίπτωση, το κείμενο της απόκρισης διαμορφώνεται δυναμικά, ενσωματώνοντας δεδομένα που έχουν συλλεχθεί κατά τη διάρκεια της συνομιλίας, όπως το όνομα του χρήστη ή τα στοιχεία της παραγγελίας. Η δυνατότητα αυτή ενισχύει τη φυσικότητα και την προσωποποίηση της αλληλεπίδρασης.
Το υποσύστημα Text-to-Speech αποτελεί βασικό στοιχείο της εμπειρίας χρήστη του

Voicebot. Η επιλογή του gTTS, σε συνδυασμό με τη σωστή προσαρμογή του παραγόμενου ήχου στις απαιτήσεις του Asterisk, επιτρέπει την αξιόπιστη και γρήγορη παραγωγή ελληνικής φωνής, συμβάλλοντας στη δημιουργία ενός συστήματος που μπορεί να υποστηρίξει φυσική τηλεφωνική συνομιλία σε πραγματικές συνθήκες.

## 3.7 Υποσύστημα Καταγραφής Παραγγελιών (Order Logging)

Στο τέλος της ροής παραγγελίας, τα δεδομένα που έχουν συλλεχθεί κατά τη διάρκεια της συνομιλίας καταγράφονται σε δομημένο αρχείο μορφής CSV (Comma-Separated Values). Το υποσύστημα καταγραφής παραγγελιών αποτελεί κρίσιμο τμήμα της συνολικής λειτουργικότητας του Voicebot, καθώς διασφαλίζει ότι τα αποτελέσματα της τηλεφωνικής αλληλεπίδρασης αποθηκεύονται με αξιόπιστο και αναγνώσιμο τρόπο, ώστε να μπορούν να αξιοποιηθούν σε μεταγενέστερο στάδιο.

Το αρχείο καταγραφής περιλαμβάνει βασικά αλλά ουσιαστικά πεδία, όπως τη χρονική σήμανση (timestamp) της κλήσης, τα στοιχεία του χρήστη (π.χ. caller ID), καθώς και τις λεπτομέρειες της παραγγελίας που συλλέχθηκαν μέσω του διαλόγου, όπως το όνομα πελάτη, το τηλέφωνο επικοινωνίας, το προϊόν και την ποσότητα. Η δομημένη αυτή μορφή επιτρέπει την εύκολη αναζήτηση, φιλτράρισμα και επεξεργασία των δεδομένων, ακόμη και με απλά εργαλεία, όπως λογιστικά φύλλα ή scripts επεξεργασίας δεδομένων.

Η επιλογή αποθήκευσης σε αρχείο CSV πραγματοποιήθηκε με γνώμονα την απλότητα, τη διαφάνεια και την αξιοπιστία. Σε αντίθεση με πιο σύνθετες λύσεις που βασίζονται σε συστήματα διαχείρισης βάσεων δεδομένων, το CSV δεν απαιτεί πρόσθετη υποδομή, ρυθμίσεις ή μηχανισμούς σύνδεσης, μειώνοντας έτσι την πολυπλοκότητα του συστήματος. Παράλληλα, η μορφή αυτή επιτρέπει την άμεση πρόσβαση στα δεδομένα, χωρίς τη μεσολάβηση ειδικού λογισμικού, γεγονός που είναι ιδιαίτερα χρήσιμο σε περιβάλλοντα όπου απαιτείται γρήγορη επισκόπηση ή εξαγωγή πληροφοριών.

Ένα επιπλέον πλεονέκτημα της προσέγγισης αυτής είναι η ανθεκτικότητα σε σφάλματα. Ακόμη και σε περίπτωση απρόσμενου τερματισμού της εφαρμογής ή του συστήματος, τα δεδομένα που έχουν ήδη καταγραφεί στο αρχείο παραμένουν διαθέσιμα και δεν χάνονται. Με τον τρόπο αυτό, το υποσύστημα καταγραφής συμβάλλει στη συνολική αξιοπιστία του Voicebot και στη διασφάλιση της ακεραιότητας των επιχειρησιακών δεδομένων.

Το υποσύστημα order logging είναι πλήρως ενοποιημένο με τον μηχανισμό διαχείρισης διαλόγου. Η καταγραφή πραγματοποιείται μόνο όταν ολοκληρωθεί επιτυχώς η ροή της παραγγελίας και επιβεβαιωθούν τα στοιχεία από τον χρήστη. Η προσέγγιση αυτή αποτρέπει την αποθήκευση ελλιπών ή λανθασμένων δεδομένων και διασφαλίζει ότι κάθε εγγραφή στο αρχείο αντιστοιχεί σε ολοκληρωμένη και έγκυρη συναλλαγή. Παρότι στην παρούσα υλοποίηση χρησιμοποιείται αρχείο CSV, η αρχιτεκτονική του συστήματος έχει σχεδιαστεί έτσι ώστε να επιτρέπει μελλοντική επέκταση. Το υποσύστημα καταγραφής μπορεί εύκολα να προσαρμοστεί ώστε να αποθηκεύει δεδομένα σε σχεσιακή βάση δεδομένων, σε σύστημα CRM ή σε backend ηλεκτρονικού εμπορίου, χωρίς να απαιτούνται σημαντικές αλλαγές στη λογική του Voicebot.

Το υποσύστημα καταγραφής παραγγελιών προσφέρει έναν απλό αλλά αποτελεσματικό μηχανισμό αποθήκευσης των αποτελεσμάτων της τηλεφωνικής αλληλεπίδρασης. Η επιλογή της μορφής CSV εξυπηρετεί τον στόχο της πρακτικής χρηστικότητας, της ευκολίας διαχείρισης και της διαφάνειας, καθιστώντας το σύστημα κατάλληλο για άμεση χρήση σε πραγματικές επιχειρησιακές συνθήκες.

## 3.8 Web Interface (Django Management Console)

Το web Interface διαχείρισης του συστήματος αποτελεί ένα κρίσιμο υποσύστημα υποστήριξης του Voicebot, καθώς λειτουργεί ως το βασικό σημείο αλληλεπίδρασης μεταξύ του ανθρώπινου διαχειριστή και της «νοημοσύνης» του συστήματος. Στόχος του Interface είναι να παρέχει έναν απλό, κατανοητό και λειτουργικό τρόπο παραμετροποίησης και ελέγχου του Voicebot, χωρίς να απαιτούνται εξειδικευμένες γνώσεις προγραμματισμού ή μηχανικής μάθησης. Το Interface υλοποιήθηκε με χρήση του Django, το οποίο προσφέρει ένα σταθερό και ασφαλές περιβάλλον ανάπτυξης web εφαρμογών. Ο διαχειριστής έχει τη δυνατότητα να εισάγει, να επεξεργάζεται και να διαγράφει δεδομένα εκπαίδευσης για το υποσύστημα NLU.

Τα δεδομένα αυτά αφορούν παραδείγματα προτάσεων και τις αντίστοιχες προθέσεις (intents), τα οποία χρησιμοποιούνται για την εκπαίδευση και τη βελτίωση του ταξινομητή προθέσεων. Ένα από τα σημαντικότερα πλεονεκτήματα του Interface είναι η δυνατότητα επανεκπαίδευσης του μοντέλου NLU μέσω γραφικού περιβάλλοντος. Ο διαχειριστής μπορεί να εκκινήσει τη διαδικασία εκπαίδευσης με το πάτημα ενός κουμπιού, χωρίς να απαιτείται άμεση πρόσβαση στο σύστημα αρχείων ή εκτέλεση scripts από τη γραμμή εντολών. Με τον τρόπο αυτό, το Voicebot μπορεί να προσαρμόζεται δυναμικά σε νέες ανάγκες, νέα προϊόντα ή αλλαγές στη γλώσσα των χρηστών. Η web κονσόλα περιλαμβάνει επίσης λειτουργία δοκιμών προτάσεων (intent testing), μέσω της οποίας ο διαχειριστής μπορεί να εισάγει ελεύθερο κείμενο και να παρατηρεί σε πραγματικό χρόνο την πρόθεση που αναγνωρίζεται από το NLU μοντέλο. Η δυνατότητα αυτή επιτρέπει την άμεση αξιολόγηση της απόδοσης του συστήματος και τη γρήγορη ανίχνευση πιθανών λαθών ή αστοχιών στην ταξινόμηση.

Επιπλέον, μέσω του Interface παρέχεται πρόσβαση στα δεδομένα που παράγονται από το υποσύστημα καταγραφής παραγγελιών. Οι καταγεγραμμένες παραγγελίες εμφανίζονται σε οργανωμένη μορφή πίνακα, επιτρέποντας την εύκολη επισκόπηση, φιλτράρισμα και εξαγωγή τους. Η δυνατότητα αυτή γεφυρώνει το χάσμα μεταξύ της αυτόματης τηλεφωνικής εξυπηρέτησης και των επιχειρησιακών διαδικασιών, καθιστώντας το Voicebot πρακτικά αξιοποιήσιμο από τον οργανισμό.

Η χρήση του Bootstrap για τη σχεδίαση του interface συμβάλλει στη δημιουργία μιας καθαρής, σύγχρονης και λειτουργικής παρουσίασης. Το responsive design διασφαλίζει ότι το interface μπορεί να χρησιμοποιηθεί από διαφορετικές συσκευές και αναλύσεις οθόνης, ενώ η απλότητα της διάταξης μειώνει το γνωστικό φορτίο του χρήστη.

Συνολικά, το Web Interface λειτουργεί ως γέφυρα μεταξύ του τεχνικού πυρήνα του Voicebot και του τελικού διαχειριστή. Παρέχει έλεγχο, διαφάνεια και ευκολία χρήσης, επιτρέποντας την αποτελεσματική διαχείριση και συνεχή βελτίωση του συστήματος. Η ύπαρξή του καθιστά το Voicebot όχι απλώς ένα πειραματικό σύστημα, αλλά ένα ολοκληρωμένο εργαλείο έτοιμο για ένταξη σε πραγματικά επιχειρησιακά περιβάλλοντα.

## 3.9 Συνολική Ανάλυση και Σχεδιαστική Συνοχή

Το σύνολο του προτεινόμενου συστήματος σχεδιάστηκε με κεντρικό άξονα τη λειτουργία σε πραγματικό χρόνο (real-time), τη modular αρχιτεκτονική και τη δυνατότητα μελλοντικής επέκτασης. Κάθε υποσύστημα αναπτύχθηκε με σαφώς ορισμένο ρόλο και διακριτές διεπαφές, επιτρέποντας τη συνεργασία τους χωρίς στενή σύζευξη (tight coupling).

Η προσέγγιση αυτή εξασφαλίζει ότι η συνολική λειτουργία του Voicebot παραμένει σταθερή, ακόμη και αν κάποιο επιμέρους τμήμα τροποποιηθεί ή αναβαθμιστεί. Η σχεδιαστική συνοχή του συστήματος επιτυγχάνεται μέσω της καθαρής ροής επεξεργασίας, η οποία ξεκινά από την τηλεφωνική κλήση και καταλήγει στην παραγωγή και αναπαραγωγή συνθετικής ομιλίας. Η αλληλουχία STT → NLU → Dialog Management → TTS διατηρείται σταθερή σε όλες τις ροές συνομιλίας, προσφέροντας προβλεψιμότητα και έλεγχο της συμπεριφοράς του συστήματος. Η χρήση state machine για τη διαχείριση διαλόγου ενισχύει περαιτέρω τη συνοχή, καθώς επιβάλλει σαφείς κανόνες μετάβασης και αποτρέπει απροσδιόριστες ή ασυνεπείς καταστάσεις.
Ιδιαίτερη έμφαση δόθηκε στον περιορισμό του χρόνου απόκρισης (latency), ο οποίος αποτελεί κρίσιμο παράγοντα σε τηλεφωνικές εφαρμογές. Η επιλογή ελαφριών και αποδοτικών μοντέλων, όπως το Whisper Small για την αναγνώριση ομιλίας και το BoW + MLP για το NLU, επιτρέπει την εκτέλεση του συστήματος σε υποδομές μέτριων προδιαγραφών, χωρίς να υποβαθμίζεται η εμπειρία του χρήστη. Παράλληλα, η αποσύνδεση της επιχειρησιακής λογικής από το τηλεφωνικό dialplan μέσω AGI συμβάλλει στη σταθερότητα και στην ταχύτερη εκτέλεση των επιμέρους διεργασιών.
Η αρχιτεκτονική του συστήματος παρουσιάζει αυξημένη ανθεκτικότητα σε σφάλματα, καθώς κάθε υποσύστημα λειτουργεί ως ανεξάρτητη μονάδα. Σε περίπτωση αστοχίας ή καθυστέρησης σε κάποιο στάδιο, το σύστημα μπορεί να εντοπίσει το πρόβλημα και να διαχειριστεί κατάλληλα τη ροή της συνομιλίας, χωρίς να καταρρεύσει συνολικά. Επιπλέον, η επιλογή απλών και διαφανών μηχανισμών αποθήκευσης δεδομένων, όπως η καταγραφή παραγγελιών σε αρχεία CSV, ενισχύει την αξιοπιστία και διευκολύνει τη διάγνωση και αποκατάσταση σφαλμάτων. Η modular σχεδίαση καθιστά το Voicebot ιδιαίτερα ευέλικτο ως προς τη συντήρηση και τη μελλοντική εξέλιξη. Νέα υποσυστήματα ή τεχνολογίες μπορούν να ενσωματωθούν σταδιακά, όπως πιο προηγμένα μοντέλα NLU, διαφορετικά TTS engines ή σύνδεση με εξωτερικά πληροφοριακά συστήματα, χωρίς να απαιτείται ριζικός ανασχεδιασμός της υπάρχουσας υποδομής. Η ύπαρξη του web περιβάλλοντος διαχείρισης ενισχύει περαιτέρω τη δυνατότητα προσαρμογής και συνεχούς βελτίωσης του συστήματος.

Συνολικά, η σχεδιαστική συνοχή και η αρχιτεκτονική του Voicebot το καθιστούν κατάλληλο για παραγωγική χρήση σε πραγματικά τηλεφωνικά περιβάλλοντα. Το σύστημα συνδυάζει τεχνική απλότητα, αξιοπιστία και επεκτασιμότητα, προσφέροντας μια ολοκληρωμένη λύση αυτόματης τηλεφωνικής εξυπηρέτησης βασισμένη σε τεχνολογίες τεχνητής νοημοσύνης και ανοιχτού λογισμικού.

ΚΕΦΑΛΑΙΟ 4

# Εργαλεία και Τεχνολογικές Επιλογές

## 4.1 Εισαγωγή

Στο παρόν κεφάλαιο παρουσιάζονται και τεκμηριώνονται οι βασικές τεχνολογικές επιλογές που πραγματοποιήθηκαν για την υλοποίηση του προτεινόμενου Voicebot, με έμφαση στη συγκριτική αξιολόγησή τους έναντι εναλλακτικών και εμπορικών λύσεων που χρησιμοποιούνται ευρέως στη διεθνή αγορά. Η ανάλυση αυτή δεν αποσκοπεί στην επανάληψη της λειτουργικής περιγραφής των επιμέρους εργαλείων, η οποία έχει ήδη παρουσιαστεί στα προηγούμενα κεφάλαια, αλλά στην αιτιολόγηση των αποφάσεων σχεδίασης και υλοποίησης που ελήφθησαν στο πλαίσιο της παρούσας εργασίας. Η επιλογή των τεχνολογιών έγινε με γνώμονα τη δημιουργία ενός συστήματος που να μπορεί να λειτουργήσει αξιόπιστα σε πραγματικό τηλεφωνικό περιβάλλον, να υποστηρίζει φυσικό διάλογο στην ελληνική γλώσσα και να ενσωματώνεται απρόσκοπτα σε υπάρχουσες τηλεφωνικές υποδομές. Για τον λόγο αυτό, απορρίφθηκαν λύσεις που, παρότι τεχνολογικά ώριμες ή ιδιαίτερα διαδεδομένες, δεν ανταποκρίνονταν επαρκώς στις απαιτήσεις αυτονομίας, ελέγχου και χαμηλού λειτουργικού κόστους που τέθηκαν από την αρχή του έργου.

Οι αποφάσεις που λήφθηκαν βασίστηκαν σε ένα σύνολο σαφώς ορισμένων κριτηρίων, τα οποία θεωρήθηκαν καθοριστικά για την επιτυχία και τη βιωσιμότητα του συστήματος. Μεταξύ αυτών περιλαμβάνονται:

- η δυνατότητα πλήρους λειτουργίας σε on-premise υποδομή, χωρίς εξάρτηση από εξωτερικές cloud υπηρεσίες.
- η επίτευξη χαμηλής καθυστέρησης (latency), απαραίτητης για τη διατήρηση φυσικού ρυθμού τηλεφωνικής συνομιλίας.
- η αξιόπιστη υποστήριξη της ελληνικής γλώσσας σε όλα τα στάδια της επεξεργασίας.
- η δυνατότητα παραμετροποίησης και πλήρους ελέγχου της επιχειρησιακής λογικής.
- η καταλληλότητα των εργαλείων για χρήση σε πραγματικά τηλεφωνικά κέντρα.
- καθώς και το χαμηλό ή μηδενικό λειτουργικό κόστος κατά τη συνεχή χρήση του συστήματος.

Η εργασία ακολουθεί μια συνειδητά επιλεγμένη προσέγγιση, η οποία δίνει προτεραιότητα στην πρακτική χρήση και στη αξιοπιστία έναντι της χρήσης σύνθετων λύσεων. Η συγκριτική ανάλυση που ακολουθεί αναδεικνύει τα πλεονεκτήματα και τους περιορισμούς κάθε τεχνολογικής επιλογής, σε σχέση με τις επικρατέστερες εναλλακτικές, και τεκμηριώνει τον τρόπο με τον οποίο οι επιλεγμένες τεχνολογίες συνθέτουν μια συνεκτική και ισορροπημένη αρχιτεκτονική Voicebot. Στις επόμενες ενότητες εξετάζονται ξεχωριστά οι βασικές τεχνολογικές συνιστώσες του συστήματος, συγκρίνονται με αντίστοιχες εναλλακτικές λύσεις και αξιολογούνται ως προς την καταλληλότητα τους για το συγκεκριμένο επιχειρησιακό και τεχνικό πλαίσιο της παρούσας διπλωματικής εργασίας.

## 4.2 Τηλεφωνική Υποδομή: VICIdial έναντι FreePBX και Twilio

Για την τηλεφωνική υποδομή του προτεινόμενου Voicebot εξετάστηκαν τρεις βασικές κατηγορίες λύσεων:
α) πλατφόρμες call center βασισμένες σε Asterisk (VICIdial),
β) γενικού σκοπού PBX συστήματα (FreePBX),
γ) cloud-based τηλεφωνικές πλατφόρμες (Twilio).

**VICIdial**

Το VICIdial αποτελεί μια εξειδικευμένη πλατφόρμα call center, βασισμένη στο Asterisk, η οποία έχει σχεδιαστεί εξαρχής για τη διαχείριση μεγάλου όγκου εισερχόμενων και εξερχόμενων κλήσεων σε περιβάλλοντα συνεχούς λειτουργίας. Σε αντίθεση με γενικού σκοπού PBX συστήματα, το VICIdial ενσωματώνει εγγενώς μηχανισμούς διαχείρισης ουρών, καμπανιών, call routing και στατιστικής παρακολούθησης, καθιστώντας το ιδιαίτερα κατάλληλο για παραγωγική χρήση σε call centers. Η άμεση και πλήρης υποστήριξη AGI scripts επιτρέπει την υλοποίηση σύνθετων αυτοματοποιημένων ροών, όπως η ενσωμάτωση Voicebots, χωρίς περιορισμούς στη χαμηλού επιπέδου τηλεφωνική λογική. Επιπλέον, η πλήρως on-premise εγκατάσταση εξασφαλίζει μηδενικό κόστος ανά κλήση και πλήρη έλεγχο των δεδομένων. Ωστόσο, η αυξημένη λειτουργικότητα συνοδεύεται από μεγαλύτερη πολυπλοκότητα εγκατάστασης και συντήρησης, απαιτώντας βασικές γνώσεις Asterisk και Linux. Παρά ταύτα, για περιβάλλοντα με αυξημένες απαιτήσεις αξιοπιστίας και κλιμάκωσης, το VICIdial αποτελεί ώριμη και δοκιμασμένη επιλογή.

**FreePBX**

Το FreePBX είναι ένα γενικού σκοπού PBX σύστημα βασισμένο στο Asterisk, το οποίο χρησιμοποιείται ευρέως σε μικρές και μεσαίες τηλεφωνικές εγκαταστάσεις. Η κύρια δύναμή του έγκειται στο φιλικό και εύχρηστο web interface, το οποίο επιτρέπει τη γρήγορη εγκατάσταση και παραμετροποίηση βασικών τηλεφωνικών λειτουργιών χωρίς εις βάθος τεχνικές γνώσεις. Υποστηρίζει τυπικά σενάρια IVR, extensions και απλή διαχείριση κλήσεων, καθιστώντας το κατάλληλο για οργανισμούς με περιορισμένες τηλεφωνικές ανάγκες. Ωστόσο, το FreePBX δεν έχει σχεδιαστεί ως ολοκληρωμένη πλατφόρμα call center και παρουσιάζει σημαντικούς περιορισμούς στη διαχείριση εισερχόμενων ουρών μεγάλης κλίμακας και στη λεπτομερή υλοποίηση αυτοματοποιημένων AGI ροών. Η υποστήριξη πιο σύνθετων σεναρίων απαιτεί συχνά πρόσθετα modules ή εκτενή προσαρμογή, γεγονός που μειώνει την ευελιξία του σε απαιτητικά περιβάλλοντα τηλεφωνικής αυτοματοποίησης. Για τον λόγο αυτό, το FreePBX κρίνεται καταλληλότερο για απλές PBX εγκαταστάσεις και όχι για σύνθετα συστήματα Voicebot σε call center περιβάλλον.

**Twilio (Cloud Telephony)**

Το Twilio αποτελεί μια εμπορική cloud πλατφόρμα τηλεφωνίας και επικοινωνίας, η οποία προσφέρει ισχυρά APIs για την ταχεία ανάπτυξη εφαρμογών φωνής και μηνυμάτων. Η βασική του υπεροχή έγκειται στην ευκολία αρχικής υλοποίησης και στην υψηλή αξιοπιστία της υποδομής του, επιτρέποντας τη δημιουργία proof-of-concept εφαρμογών με ελάχιστο χρόνο ανάπτυξης. Παράλληλα, προσφέρει άμεση διασύνδεση με cloud υπηρεσίες αναγνώρισης και σύνθεσης ομιλίας, διευκολύνοντας τη δημιουργία σύγχρονων conversational εφαρμογών. Ωστόσο, η χρήση του Twilio συνεπάγεται συνεχή χρέωση ανά λεπτό κλήσης και ανά λειτουργία, γεγονός που καθιστά το λειτουργικό κόστος σημαντικό σε σενάρια μεγάλου όγκου κλήσεων. Επιπλέον, η πλήρης εξάρτηση από cloud υποδομές συνεπάγεται αποστολή φωνητικών δεδομένων σε τρίτους παρόχους και περιορισμένο έλεγχο στη χαμηλού επιπέδου τηλεφωνική ροή. Οι παράγοντες αυτοί καθιστούν το Twilio λιγότερο κατάλληλο για on-premise περιβάλλοντα με αυξημένες απαιτήσεις ελέγχου, ασφάλειας και μακροχρόνιας οικονομικής βιωσιμότητας.

**Συγκριτικός Πίνακας: VICIdial vs FreePBX vs Twilio**

| Χαρακτηριστικό | VICIdial | FreePBX | Twilio (Cloud Telephony) |
|---|---|---|---|
| **Τύπος λύσης** | Call center πλατφόρμα (Asterisk-based) | Γενικού σκοπού PBX | Cloud τηλεφωνική πλατφόρμα |
| **Τρόπος λειτουργίας** | Πλήρως on-premise | On-premise | Αποκλειστικά cloud |
| **Κόστος ανά κλήση** | Μηδενικό | Μηδενικό | Χρέωση ανά λεπτό |
| **Έλεγχος τηλεφωνικής ροής** | Πλήρης | Μέτριος | Περιορισμένος |
| **Υποστήριξη AGI scripts** | Πλήρης & εγγενής | Υποστηρίζεται, όχι βελτιστοποιημένη | Μέσω API, όχι AGI |
| **Σχεδίαση για call center** | Ναι (εγγενώς) | Όχι | Ναι (cloud-based) |
| **Inbound queues μεγάλης κλίμακας** | Άριστη υποστήριξη | Περιορισμένη | Καλή |
| **Ενσωμάτωση Voicebot** | Άμεση & ευέλικτη | Εφικτή αλλά με περιορισμούς | Εύκολη αλλά εξαρτώμενη από cloud |
| **Latency (real-time διάλογος)** | Χαμηλό | Χαμηλό | Εξαρτάται από δίκτυο |
| **Έλεγχος δεδομένων** | Πλήρης (τοπικά) | Πλήρης (τοπικά) | Περιορισμένος (σε τρίτο πάροχο) |
| **Κλιμάκωση (scalability)** | Υψηλή (on-premise) | Μέτρια | Πολύ υψηλή (cloud) |
| **Ευκολία εγκατάστασης** | Χαμηλή | Υψηλή | Πολύ υψηλή |
| **Καταλληλότητα για παραγωγή** | Πολύ υψηλή | Μέτρια | Υψηλή |
| **Εξάρτηση από τρίτους** | Όχι | Όχι | Ναι |
| **Καταλληλότητα για AI Voicebot** | Άριστη | Περιορισμένη | Καλή |

Η επιλογή του VICIdial κρίθηκε καταλληλότερη για την παρούσα εργασία, καθώς συνδυάζει τα πλεονεκτήματα της πλήρους τηλεφωνικής ελευθερίας του Asterisk με εξειδικευμένες δυνατότητες call center. Σε αντίθεση με το FreePBX, το VICIdial είναι σχεδιασμένο εξαρχής για διαχείριση μαζικών τηλεφωνικών ροών, ενώ σε σύγκριση με cloud λύσεις όπως το Twilio, προσφέρει πλήρη αυτονομία, χαμηλό κόστος και απόλυτο έλεγχο των δεδομένων.

Η επιλογή αυτή ευθυγραμμίζεται με τον στόχο της διπλωματικής εργασίας, ο οποίος είναι η ανάπτυξη ενός λειτουργικού, επεκτάσιμου και παραγωγικά αξιοποιήσιμου ελληνόφωνου Voicebot, ικανού να ενσωματωθεί σε πραγματικό call center περιβάλλον χωρίς εξαρτήσεις από εμπορικές υπηρεσίες.

## 4.3 Αναγνώριση Ομιλίας: Whisper έναντι Εμπορικών STT Υπηρεσιών

Για το υποσύστημα αναγνώρισης ομιλίας (Speech-to-Text – STT) εξετάστηκαν τόσο ανοιχτές όσο και εμπορικές λύσεις, με κύριο στόχο την αξιόπιστη μεταγραφή ελληνικής ομιλίας σε συνθήκες τηλεφωνικού ήχου. Μεταξύ των διαθέσιμων επιλογών περιλαμβάνονται εμπορικές υπηρεσίες όπως το Google Speech-to-Text, το Microsoft Azure Speech Services και το Amazon Transcribe, οι οποίες αποτελούν ώριμες και ευρέως χρησιμοποιούμενες λύσεις σε cloud περιβάλλοντα.

Οι εμπορικές αυτές υπηρεσίες προσφέρουν συνήθως υψηλά ποσοστά ακρίβειας και προηγμένες δυνατότητες βελτιστοποίησης, ιδιαίτερα σε περιβάλλοντα υψηλής ποιότητας ήχου. Ωστόσο, η χρήση τους προϋποθέτει αποκλειστική λειτουργία μέσω cloud υποδομών, γεγονός που συνεπάγεται συνεχή λειτουργικό κόστος, εξάρτηση από τρίτους παρόχους και αποστολή φωνητικών δεδομένων εκτός του ελεγχόμενου περιβάλλοντος του οργανισμού. Παράλληλα, οι υπηρεσίες αυτές λειτουργούν ως «μαύρα κουτιά», χωρίς δυνατότητα ουσιαστικής παραμετροποίησης ή ελέγχου του υποκείμενου μοντέλου.

Αντίθετα, στην παρούσα εργασία επιλέχθηκε το Whisper, ένα ανοιχτού κώδικα μοντέλο αναγνώρισης ομιλίας, το οποίο μπορεί να εκτελεστεί πλήρως σε on-premise περιβάλλον. Το Whisper παρουσιάζει ιδιαίτερα καλή απόδοση στην ελληνική γλώσσα και εμφανίζει αυξημένη ανθεκτικότητα σε θορυβώδη ή υποβαθμισμένο ήχο, χαρακτηριστικό ιδιαίτερα σημαντικό για τηλεφωνικές εφαρμογές. Επιπλέον, η χρήση του δεν συνεπάγεται κόστος ανά λεπτό ομιλίας και επιτρέπει πλήρη έλεγχο της διαδικασίας μεταγραφής.

Η επιλογή της έκδοσης **Whisper small** αποτέλεσε συνειδητό συμβιβασμό μεταξύ ακρίβειας και υπολογιστικού κόστους. Το συγκεκριμένο μοντέλο είναι αρκετά ελαφρύ ώστε να εκτελείται σε CPU-only περιβάλλον, διατηρώντας παράλληλα ικανοποιητική ακρίβεια για real-time τηλεφωνική χρήση. Η δυνατότητα άμεσης ενσωμάτωσης σε Python περιβάλλον το καθιστά ιδιαίτερα κατάλληλο για χρήση σε AGI scripts και συστήματα όπως το παρόν Voicebot.

**Συγκριτικός Πίνακας: Whisper vs Εμπορικές Υπηρεσίες STT**

| Χαρακτηριστικό | Whisper (Open-source) | Google STT | Azure Speech | Amazon Transcribe |
|---|---|---|---|---|
| **Τρόπος λειτουργίας** | Πλήρως on-premise | Cloud | Cloud | Cloud |
| **Κόστος χρήσης** | Μηδενικό | Χρέωση ανά λεπτό | Χρέωση ανά λεπτό | Χρέωση ανά λεπτό |
| **Υποστήριξη ελληνικής γλώσσας** | Πολύ καλή | Καλή | Καλή | Μέτρια |
| **Ανθεκτικότητα σε τηλεφωνικό θόρυβο** | Υψηλή | Υψηλή | Υψηλή | Μέτρια |
| **Έλεγχος / παραμετροποίηση** | Πλήρης | Όχι | Όχι | Όχι |

| | | | | |
|---|---|---|---|---|
| **μοντέλου** | | | | |
| **Αποστολή δεδομένων σε τρίτους** | Όχι | Ναι | Ναι | Ναι |
| **Latency (real-time χρήση)** | Χαμηλό (on-premise) | Εξαρτάται από δίκτυο | Εξαρτάται από δίκτυο | Εξαρτάται από δίκτυο |
| **Ενσωμάτωση σε Python / AGI** | Άμεση | Μέσω API | Μέσω API | Μέσω API |
| **Καταλληλότητα για τηλεφωνικό Voicebot** | Άριστη | Καλή | Καλή | Μέτρια |
| **Εξάρτηση από cloud** | Όχι | Ναι | Ναι | Ναι |

Με βάση τη συγκριτική αξιολόγηση, το Whisper κρίθηκε καταλληλότερο καθώς προσφέρει τον βέλτιστο συνδυασμό ακρίβειας, αυτονομίας και ελέγχου. Η δυνατότητα πλήρους on-premise λειτουργίας, η καλή υποστήριξη της ελληνικής γλώσσας και η ανθεκτικότητα σε τηλεφωνικό ήχο καθιστούν το Whisper ιδανική επιλογή για ένα παραγωγικό τηλεφωνικό Voicebot. Παρότι οι εμπορικές λύσεις προσφέρουν υψηλή ακρίβεια, οι περιορισμοί κόστους, εξάρτησης και ελέγχου τις καθιστούν λιγότερο κατάλληλες για το συγκεκριμένο επιχειρησιακό πλαίσιο.

## 4.4 Κατανόηση Φυσικής Γλώσσας: Custom NLU έναντι Dialogflow / Rasa

Για το υποσύστημα κατανόησης φυσικής γλώσσας (Natural Language Understanding – NLU) εξετάστηκαν δύο βασικές προσεγγίσεις: η χρήση έτοιμων conversational AI frameworks, όπως το Google Dialogflow και το Rasa, και η ανάπτυξη ενός προσαρμοσμένου (custom) μοντέλου κατανόησης προθέσεων. Στην παρούσα εργασία επιλέχθηκε η δεύτερη προσέγγιση, με την υλοποίηση ενός custom NLU μοντέλου βασισμένου σε αναπαράσταση Bag-of-Words και νευρωνικό δίκτυο τύπου Multilayer Perceptron (MLP).

Τα έτοιμα frameworks, όπως το Dialogflow και το Rasa, προσφέρουν εκτεταμένες δυνατότητες, όπως αυτόματη εξαγωγή οντοτήτων, υποστήριξη σύνθετων διαλόγων και προηγμένα εργαλεία διαχείρισης συνομιλιών. Ωστόσο, οι δυνατότητες αυτές συνοδεύονται από αυξημένη πολυπλοκότητα, σημαντικές εξαρτήσεις από τρίτα συστήματα και, σε αρκετές περιπτώσεις, περιορισμένο έλεγχο της εσωτερικής λειτουργίας του NLU μηχανισμού.

Το Google Dialogflow λειτουργεί αποκλειστικά ως cloud υπηρεσία και παρέχει περιορισμένη διαφάνεια ως προς τον τρόπο εκπαίδευσης και λήψης αποφάσεων του μοντέλου, γεγονός που το καθιστά ακατάλληλο για περιβάλλοντα με αυξημένες απαιτήσεις ελέγχου και προστασίας δεδομένων. Αντίστοιχα, το Rasa, παρότι προσφέρει on-premise δυνατότητες, απαιτεί σημαντική παραμετροποίηση, σύνθετη ρύθμιση pipelines και μεγαλύτερο υπολογιστικό κόστος, ιδιαίτερα όταν χρησιμοποιούνται σύγχρονες embedding-based προσεγγίσεις.

Αντίθετα, η ανάπτυξη custom NLU μοντέλου επιλέχθηκε με γνώμονα τη στοχευμένη φύση του τηλεφωνικού διαλόγου. Στο συγκεκριμένο σενάριο, τα intents είναι σαφώς ορισμένα και περιορισμένου πλήθους, γεγονός που καθιστά περιττή τη χρήση πολύπλοκων και βαριών πλαισίων. Η custom προσέγγιση επιτρέπει πλήρη έλεγχο της εκπαίδευσης, ελάχιστο χρόνο απόκρισης και εύκολη προσαρμογή σε πραγματικά δεδομένα χρήσης.

Ιδιαίτερη σημασία έχει και η υποστήριξη της ελληνικής γλώσσας. Τα έτοιμα frameworks δεν προσφέρουν πάντα βέλτιστη απόδοση σε ελληνικά δεδομένα, ειδικά σε τηλεφωνικό περιβάλλον, ενώ η custom υλοποίηση επιτρέπει την άμεση ενσωμάτωση γλωσσικών ιδιαιτεροτήτων και πραγματικών παραδειγμάτων χρήσης.

**Συγκριτικός Πίνακας: Custom NLU vs Dialogflow vs Rasa**

| Χαρακτηριστικό | Custom NLU (BoW + MLP) | Google Dialogflow | Rasa |
|---|---|---|---|
| **Τρόπος λειτουργίας** | Πλήρως on-premise | Cloud | On-premise |
| **Έλεγχος μοντέλου** | Πλήρης | Περιορισμένος (black-box) | Μερικός |
| **Κόστος χρήσης** | Μηδενικό | Χρέωση ανά αίτημα | Μηδενικό (self-hosted) |
| **Υποστήριξη ελληνικής γλώσσας** | Πολύ καλή (προσαρμοσμένη) | Καλή | Μέτρια |
| **Latency** | Πολύ χαμηλό | Εξαρτάται από δίκτυο | Μέτριο |
| **Πολυπλοκότητα ρύθμισης** | Χαμηλή | Χαμηλή | Υψηλή |
| **Επανεκπαίδευση από μη ειδικούς** | Ναι (μέσω Web UI) | Περιορισμένη | Δύσκολη |
| **Καταλληλότητα για τηλεφωνικά intents** | Άριστη | Καλή | Καλή |
| **Εξαρτήσεις από τρίτους** | Όχι | Ναι | Όχι |
| **Επεκτασιμότητα** | Υψηλή (ελεγχόμενη) | Υψηλή | Υψηλή αλλά σύνθετη |

Με βάση τη συγκριτική αξιολόγηση, η επιλογή custom NLU μοντέλου κρίθηκε καταλληλότερη καθώς ευθυγραμμίζεται με τη στοχευμένη φύση του τηλεφωνικού Voicebot. Η απλότητα της αρχιτεκτονικής, το ελάχιστο latency και ο πλήρης έλεγχος της εκπαίδευσης υπερτερούν έναντι της θεωρητικής ευελιξίας που προσφέρουν τα έτοιμα frameworks.
Για τηλεφωνικά συστήματα με περιορισμένο και σαφώς ορισμένο σύνολο προθέσεων, η χρήση απλού και αξιόπιστου NLU μηχανισμού αποδεικνύεται πιο αποτελεσματική από την ενσωμάτωση βαριών conversational πλατφορμών. Η custom προσέγγιση που υιοθετήθηκε παρέχει σταθερότητα, διαφάνεια και δυνατότητα εξέλιξης με βάση πραγματικά δεδομένα χρήσης, χωρίς περιττή πολυπλοκότητα.

## 4.5 Παραγωγή Ομιλίας: gTTS έναντι Εμπορικών TTS Λύσεων

Για το υποσύστημα παραγωγής συνθετικής ομιλίας (Text-to-Speech – TTS) επιλέχθηκε το εργαλείο gTTS (Google Text-to-Speech Python library), αντί εμπορικών λύσεων όπως το Amazon Polly, το Google Cloud Text-to-Speech και το Microsoft Neural TTS. Η επιλογή αυτή δεν βασίστηκε στη σύγκριση αποκλειστικά της φυσικότητας της παραγόμενης φωνής, αλλά σε ένα ευρύτερο σύνολο τεχνικών και επιχειρησιακών κριτηρίων, τα οποία είναι ιδιαίτερα

κρίσιμα για τηλεφωνικά συστήματα πραγματικού χρόνου. Οι εμπορικές πλατφόρμες TTS προσφέρουν προηγμένες, νευρωνικές φωνές υψηλής φυσικότητας, με δυνατότητες όπως διαφορετικοί τόνοι, συναισθηματική απόχρωση και ποικιλία φωνών. Ωστόσο οι δυνατότητες αυτές συνοδεύονται από σημαντικά μειονεκτήματα στο πλαίσιο ενός τηλεφωνικού Voicebot που λειτουργεί on-premise. Συγκεκριμένα, οι υπηρεσίες αυτές λειτουργούν αποκλειστικά μέσω cloud, απαιτώντας τη μεταφορά του κειμένου απόκρισης σε απομακρυσμένους servers και την επιστροφή του παραγόμενου ήχου, γεγονός που εισάγει επιπλέον καθυστέρηση και αυξάνει την εξάρτηση από τη δικτυακή συνδεσιμότητα. Επιπλέον, οι εμπορικές TTS λύσεις χρεώνονται συνήθως ανά χαρακτήρα ή ανά λεπτό παραγόμενης ομιλίας, κάτι που σε περιβάλλον call center με μεγάλο όγκο κλήσεων μπορεί να οδηγήσει σε σημαντικό λειτουργικό κόστος. Παράλληλα, η αποστολή ακόμη και του παραγόμενου κειμένου σε τρίτους παρόχους ενδέχεται να δημιουργεί ζητήματα συμμόρφωσης με πολιτικές προστασίας δεδομένων.
Αντίθετα, το gTTS επιλέχθηκε ως μια ελαφριά και πρακτική λύση, επαρκής για τις ανάγκες τηλεφωνικού διαλόγου. Παρότι η φυσικότητα της φωνής δεν φτάνει το επίπεδο των σύγχρονων neural TTS συστημάτων, η ποιότητα κρίνεται απολύτως ικανοποιητική για τηλεφωνική χρήση, όπου το εύρος συχνοτήτων και η συμπίεση του ήχου περιορίζουν ούτως ή άλλως την αντιληπτή λεπτομέρεια. Η ταχύτητα παραγωγής ήχου, η απλότητα ενσωμάτωσης σε Python περιβάλλον και η απουσία κόστους καθιστούν το gTTS ιδιαίτερα κατάλληλο για real-time Voicebots, όπου η άμεση απόκριση είναι σημαντικότερη από τη φωνητική εκφραστικότητα. Επιπλέον, η υποστήριξη της ελληνικής γλώσσας χωρίς πρόσθετη ρύθμιση αποτέλεσε καθοριστικό παράγοντα επιλογής.

**Συγκριτικός Πίνακας: gTTS vs Amazon Polly vs Google TTS vs Microsoft Neural TTS**

| Χαρακτηριστικό | gTTS | Amazon Polly | Google TTS | Microsoft Neural TTS |
|---|---|---|---|---|
| **Τρόπος λειτουργίας** | Τοπική χρήση μέσω βιβλιοθήκης | Cloud | Cloud | Cloud |
| **Κόστος χρήσης** | Μηδενικό | Ανά χαρακτήρα | Ανά χαρακτήρα | Ανά χαρακτήρα |
| **Υποστήριξη ελληνικής γλώσσας** | Ναι | Ναι | Ναι | Ναι |
| **Φυσικότητα φωνής** | Μέτρια | Πολύ υψηλή | Πολύ υψηλή | Πολύ υψηλή |
| **Latency** | Πολύ χαμηλό | Εξαρτάται από δίκτυο | Εξαρτάται από δίκτυο | Εξαρτάται από δίκτυο |
| **On-premise λειτουργία** | Ναι | Όχι | Όχι | Όχι |
| **Εξάρτηση από τρίτους** | Όχι | Ναι | Ναι | Ναι |
| **Ευκολία ενσωμάτωσης** | Πολύ υψηλή | Μέτρια | Μέτρια | Μέτρια |
| **Καταλληλότητα για τηλεφωνία** | Υψηλή | Υψηλή | Υψηλή | Υψηλή |
| **Καταλληλότητα για call center μεγάλης κλίμακας** | Υψηλή (χωρίς κόστος) | Μέτρια (κόστος) | Μέτρια (κόστος) | Μέτρια (κόστος) |

Η επιλογή του gTTS δίνει προτεραιότητα στην αυτονομία, τη χαμηλή καθυστέρηση και το μηδενικό λειτουργικό κόστος. Σε τηλεφωνικά Voicebots, όπου η ομιλία μεταδίδεται μέσω περιορισμένου φασματικού εύρους και συμπίεσης, η απόλυτη φυσικότητα της φωνής έχει μικρότερη σημασία σε σχέση με την ταχύτητα και τη σταθερότητα της απόκρισης. Το gTTS βέβαια, αποτελεί μια τεχνικά επαρκή και επιχειρησιακά αποδοτική λύση για τηλεφωνική χρήση, ενώ διατηρεί τη δυνατότητα μελλοντικής αντικατάστασης από πιο προηγμένα TTS συστήματα, εφόσον αλλάξουν οι απαιτήσεις ή οι περιορισμοί του συστήματος.

## 4.6 Συνολική Αξιολόγηση Τεχνολογικών Επιλογών

Οι τεχνολογικές επιλογές που επιλέχθηκαν συγκροτούν ένα συνεκτικό, ισορροπημένο και ρεαλιστικό πλαίσιο ανάπτυξης τηλεφωνικών Voicebots. Αντί της επιλογής ολοκληρωμένων εμπορικών ή αποκλειστικά cloud-based πλατφορμών, επιλέχθηκε συνειδητά ένας συνδυασμός ανοιχτών και ελεγχόμενων τεχνολογιών, ο οποίος ανταποκρίνεται στις ιδιαίτερες απαιτήσεις της τηλεφωνικής επικοινωνίας σε πραγματικό χρόνο. Ο συνδυασμός των Asterisk/VICIdial, Whisper, custom NLU και gTTS επιτρέπει την πλήρη αυτονομία του συστήματος, χωρίς εξάρτηση από εξωτερικούς παρόχους υπηρεσιών ή συνεχή σύνδεση με cloud υποδομές. Η επιλογή αυτή προσφέρει αυξημένο έλεγχο επί της τηλεφωνικής ροής, των δεδομένων ομιλίας και των λεκτικών αλληλεπιδράσεων, γεγονός ιδιαίτερα κρίσιμο σε περιβάλλοντα όπου ισχύουν αυστηρές πολιτικές προστασίας προσωπικών δεδομένων και κανονισμοί συμμόρφωσης. Η απουσία κόστους ανά κλήση, ανά λεπτό ομιλίας ή ανά χαρακτήρα καθιστά τη λύση οικονομικά βιώσιμη για κλιμάκωση σε πραγματικά call center σενάρια. Σε αντίθεση με τις εμπορικές πλατφόρμες, όπου το κόστος αυξάνεται γραμμικά με τον όγκο χρήσης, το προτεινόμενο σύστημα επιτρέπει σταθερό και προβλέψιμο λειτουργικό κόστος, ανεξάρτητο από τον αριθμό των κλήσεων.
Σε τεχνικό επίπεδο, οι επιλεγμένες τεχνολογίες προσφέρουν χαμηλό latency και αξιόπιστη λειτουργία σε περιβάλλοντα περιορισμένων υπολογιστικών πόρων. Το Whisper small αποδεικνύεται επαρκές για την αναγνώριση ελληνικής τηλεφωνικής ομιλίας, ενώ το custom NLU μοντέλο εξασφαλίζει ταχύτατη ταξινόμηση προθέσεων με ελάχιστο υπολογιστικό κόστος. Η επιλογή απλών αλλά αποτελεσματικών μοντέλων αντανακλά μια σχεδιαστική φιλοσοφία που προτάσσει τη σταθερότητα και την προβλεψιμότητα έναντι της θεωρητικής πολυπλοκότητας. Η modular αρχιτεκτονική του συστήματος διασφαλίζει, επιπλέον, υψηλό βαθμό ευελιξίας και επεκτασιμότητας. Κάθε υποσύστημα μπορεί να αναβαθμιστεί ή να αντικατασταθεί ανεξάρτητα (π.χ. μετάβαση σε πιο προηγμένο TTS ή NLU μοντέλο), χωρίς να απαιτείται συνολικός επανασχεδιασμός της τηλεφωνικής λογικής. Η δυνατότητα αυτή καθιστά το Voicebot κατάλληλο όχι μόνο για πιλοτική χρήση, αλλά και για μακροχρόνια παραγωγική λειτουργία.

Η προσέγγιση που ακολουθήθηκε αποδεικνύει ότι είναι εφικτή η ανάπτυξη ενός λειτουργικού, αξιόπιστου και ελληνόφωνου τηλεφωνικού Voicebot, ικανού να ανταγωνιστεί εμπορικές λύσεις σε πραγματικά επιχειρησιακά σενάρια. Η λύση που παρουσιάζεται συνδυάζει τεχνολογική ωριμότητα, πρακτική εφαρμοσιμότητα και στρατηγική ανεξαρτησία, αποτελώντας μια ισχυρή βάση για περαιτέρω εξέλιξη και αξιολόγηση, όπως αυτή παρουσιάζεται στο επόμενο κεφάλαιο της εργασίας.

ΚΕΦΑΛΑΙΟ 5

# Υλοποίηση

## 5.1 Εισαγωγή στην Υλοποίηση

Η υλοποίηση του Voicebot βασίζεται σε ανεξάρτητα "κομμάτια" αρχιτεκτονικής (modular αρχιτεκτονική), σύμφωνα με την οποία το σύστημα χωρίζεται σε διακριτά υποσυστήματα. Το κάθε σύστημα εκτελεί μια συγκεκριμένη λειτουργία. Κάθε υποσύστημα επικοινωνεί με τα υπόλοιπα μέσω σαφώς ορισμένων διεπαφών, χωρίς να εξαρτάται από τον εσωτερικό τρόπο υλοποίησής τους. Η προσέγγιση αυτή ακολουθεί βασικές αρχές μηχανικής λογισμικού, όπως ο διαχωρισμός ευθυνών και η περιορισμένη σύζευξη, με σκοπό τη δημιουργία ενός συστήματος απλούστερου στη συντήρηση και πιο ευέλικτου σε αλλαγές. Η modular σχεδίαση επιτρέπει την ανεξάρτητη ανάπτυξη, δοκιμή και βελτίωση κάθε επιμέρους τμήματος του Voicebot. Για παράδειγμα, το υποσύστημα αναγνώρισης ομιλίας μπορεί να αντικατασταθεί ή να αναβαθμιστεί με διαφορετικό μοντέλο, χωρίς να απαιτούνται αλλαγές στη λογική του διαλόγου ή στο τηλεφωνικό περιβάλλον. Αντίστοιχα, το μοντέλο κατανόησης φυσικής γλώσσας μπορεί να επανεκπαιδευτεί ή να τροποποιηθεί, χωρίς να επηρεάζεται η λειτουργία της τηλεφωνικής υποδομής. Με τον τρόπο αυτό, το σύστημα καθίσταται πιο ανθεκτικό σε μελλοντικές αλλαγές και ευκολότερο στην επέκταση, καθώς νέες λειτουργίες ή βελτιώσεις μπορούν να προστεθούν χωρίς να απαιτείται συνολικός επανασχεδιασμός.

Όλα τα επιμέρους υποσυστήματα — Speech-to-Text (STT), Natural Language Understanding (NLU), Dialog Management, Text-to-Speech (TTS) και Telephony — συνεργάζονται σε πραγματικό χρόνο, σχηματίζοντας μια ενιαία αλυσίδα επεξεργασίας τηλεφωνικής αλληλεπίδρασης. Η χρονική αλληλουχία των σταδίων αυτών είναι κρίσιμη, καθώς οποιαδήποτε καθυστέρηση σε ένα από τα επιμέρους βήματα γίνεται άμεσα αντιληπτή από τον χρήστη και επηρεάζει τη φυσικότητα της συνομιλίας. Για τον λόγο αυτό, ιδιαίτερη έμφαση δόθηκε στη βελτιστοποίηση της ροής εκτέλεσης και στη μείωση του συνολικού χρόνου καθυστέρησης (latency). Ιδιαίτερη σημασία έχει η επιλογή λειτουργίας του συστήματος σε πλήρως on-premise περιβάλλον. Το Voicebot εκτελείται εξολοκλήρου στον VICIdial server, χωρίς εξάρτηση από εμπορικές cloud υπηρεσίες ή εξωτερικά APIs. Η επιλογή αυτή δεν αποτελεί μόνο τεχνική προτίμηση, αλλά συνειδητή σχεδιαστική απόφαση, η οποία εξασφαλίζει:

- αυξημένο έλεγχο επί των δεδομένων και της επεξεργασίας τους,
- συμμόρφωση με αυστηρές πολιτικές προστασίας προσωπικών δεδομένων,
- μειωμένο λειτουργικό κόστος, καθώς δεν απαιτούνται συνδρομές ή χρεώσεις ανά χρήση,
- ανεξαρτησία από τη διαθεσιμότητα ή τους περιορισμούς τρίτων παρόχων.

Η υλοποίηση του Voicebot βασίζεται σε ένα σύνολο διακριτών δομικών στοιχείων, τα οποία συνεργάζονται στενά μεταξύ τους. Κεντρικό ρόλο διαδραματίζει το AGI script vici_voicebot.py, το οποίο λειτουργεί ως ο βασικός ελεγκτής της τηλεφωνικής ροής και αναλαμβάνει τη διασύνδεση του τηλεφωνικού περιβάλλοντος με τα υποσυστήματα τεχνητής νοημοσύνης. Παράλληλα, το μοντέλο κατανόησης φυσικής γλώσσας υλοποιείται στο αρχείο nlu_intent_pt.py, ενώ η διαδικασία εκπαίδευσης και επανεκπαίδευσης του μοντέλου πραγματοποιείται μέσω του script train_intents.py.

Τα πρότυπα απαντήσεων ορίζονται στο αρχείο response_templates.py, επιτρέποντας τη διαχείριση σταθερών και δυναμικών απαντήσεων με δομημένο τρόπο. Η ύπαρξη ξεχωριστού αρχείου προτύπων ενισχύει τη διαχωρισμένη λογική μεταξύ περιεχομένου και λειτουργικότητας. Επιπλέον, το web περιβάλλον διαχείρισης, υλοποιημένο με χρήση Django, παρέχει τα απαραίτητα εργαλεία για την επεξεργασία δεδομένων εκπαίδευσης, την επανεκπαίδευση του μοντέλου και την εποπτεία των καταγεγραμμένων παραγγελιών.

Τέλος, η υλοποίηση αξιοποιεί βιβλιοθήκες ανοιχτού κώδικα, όπως το Whisper για την αναγνώριση ομιλίας και το PyTorch για την υλοποίηση και εκπαίδευση του NLU μοντέλου. Η επιλογή των συγκεκριμένων εργαλείων συνάδει με τη συνολική φιλοσοφία του έργου, η οποία εστιάζει στη χρήση ανοικτών τεχνολογιών, στη διαφάνεια και στη δυνατότητα πλήρους ελέγχου της λειτουργίας του συστήματος.

## 5.2 AGI Script – Το Κεντρικό Υποσύστημα (vici_voicebot.py)

Το AGI script αποτελεί την «καρδιά» της υλοποίησης του Voicebot, λειτουργεί ως το κεντρικό σημείο συντονισμού όλων των υποσυστημάτων κατά τη διάρκεια μιας τηλεφωνικής κλήσης. Μέσω της διεπαφής Asterisk Gateway Interface (AGI), το σύστημα αποκτά άμεση πρόσβαση στις λειτουργίες του τηλεφωνικού κέντρου και μπορεί να ελέγχει πλήρως τη ροή της συνομιλίας, χωρίς να απαιτείται η ενσωμάτωση πολύπλοκης λογικής στο τηλεφωνικό dialplan.

Ο ρόλος του AGI script δεν περιορίζεται στην απλή εκτέλεση τηλεφωνικών εντολών, αλλά επεκτείνεται στην αλληλεπίδραση μεταξύ του τηλεφωνικού περιβάλλοντος και των υποσυστημάτων τεχνητής νοημοσύνης. Με τον τρόπο αυτό, το AGI λειτουργεί ως ενδιάμεσο επίπεδο, το οποίο μεταφράζει τα γεγονότα της κλήσης (ομιλία χρήστη, παύσεις, τερματισμό) σε υπολογιστικές διεργασίες και αντίστροφα μετατρέπει τις αποφάσεις του συστήματος σε τηλεφωνικές ενέργειες.

Συγκεκριμένα, το AGI script:

1. δέχεται την εισερχόμενη τηλεφωνική κλήση από το Asterisk και αναλαμβάνει την αρχικοποίησή της.
2. ενεργοποιεί τη διαδικασία ηχογράφησης της ομιλίας του συνομιλητή, λαμβάνοντας υπόψη χρονικά όρια και συνθήκες τερματισμού.
3. προωθεί το παραγόμενο ηχητικό αρχείο στο υποσύστημα αναγνώρισης ομιλίας (STT).
4. αποστέλλει το κείμενο που προκύπτει από το STT στο υποσύστημα κατανόησης φυσικής γλώσσας (NLU).
5. λαμβάνει την αναγνωρισμένη πρόθεση του χρήστη και, σε συνδυασμό με την τρέχουσα κατάσταση του διαλόγου, αποφασίζει την κατάλληλη επόμενη ενέργεια.
6. δημιουργεί το κείμενο της απόκρισης και ενεργοποιεί το υποσύστημα σύνθεσης ομιλίας (TTS).
7. τέλος, αναπαράγει το παραγόμενο ηχητικό μήνυμα στον χρήστη μέσω του Asterisk.

Η συγκεντρωτική αυτή διαχείριση καθιστά το AGI script το βασικό σημείο ελέγχου της συνομιλίας. Όλη η λογική που αφορά τη ροή του διαλόγου, τη διαχείριση καταστάσεων και τον συγχρονισμό των υποσυστημάτων υλοποιείται σε ένα ενιαίο επίπεδο, το οποίο είναι ανεξάρτητο από το τηλεφωνικό dialplan. Η επιλογή αυτή προσφέρει σημαντικά

πλεονεκτήματα, καθώς επιτρέπει την ανάπτυξη και τροποποίηση σύνθετων ροών συνομιλίας (conversational flows) χωρίς να απαιτούνται αλλαγές στη ρύθμιση του τηλεφωνικού συστήματος.

Επιπλέον, η υλοποίηση του AGI script σε γλώσσα Python διευκολύνει την ενσωμάτωση βιβλιοθηκών τεχνητής νοημοσύνης και την υλοποίηση πολύπλοκης λογικής, η οποία θα ήταν δύσκολο ή μη πρακτικό να υλοποιηθεί απευθείας στο Asterisk. Επίσης το AGI script έχει σχεδιαστεί ώστε να λειτουργεί ως stateless μηχανισμός σε επίπεδο τηλεφωνικής κλήσης, με την κατάσταση του διαλόγου και τα context data να διατηρούνται σε κατάλληλες δομές δεδομένων κατά τη διάρκεια της συνομιλίας. Με τον τρόπο αυτό, το σύστημα μπορεί να χειριστεί πολλαπλές ταυτόχρονες κλήσεις, χωρίς σύγκρουση δεδομένων μεταξύ διαφορετικών χρηστών. Το AGI script αποτελεί το κεντρικό υποσύστημα που ενοποιεί το τηλεφωνικό περιβάλλον με τα υποσυστήματα τεχνητής νοημοσύνης. Η αρχιτεκτονική αυτή επιλογή καθιστά το Voicebot ευέλικτο, επεκτάσιμο και κατάλληλο για υλοποίηση σύνθετων σεναρίων τηλεφωνικής εξυπηρέτησης, διατηρώντας παράλληλα τον πλήρη έλεγχο της ροής και της συμπεριφοράς του συστήματος.

### 5.2.1 Ροή Εκτέλεσης AGI Script

Η συνολική λογική εκτέλεσης του AGI script ακολουθεί μια σαφώς καθορισμένη και αυστηρά ελεγχόμενη ακολουθία βημάτων, η οποία διασφαλίζει την ομαλή και αξιόπιστη λειτουργία του Voicebot κατά τη διάρκεια μιας τηλεφωνικής κλήσης. Η ροή αυτή έχει σχεδιαστεί ώστε να καλύπτει ολόκληρο τον κύκλο της αλληλεπίδρασης, από τη στιγμή που ο χρήστης καλεί τον αριθμό έως την παραγωγή και αναπαραγωγή της τελικής φωνητικής απόκρισης.

Η ακολουθία εκτέλεσης μπορεί να αποτυπωθεί συνοπτικά ως εξής:

```
[Κλήση]
   ↓
Asterisk → Εκτέλεση AGI
   ↓
vici_voicebot.py → Welcome
   ↓
Record → Whisper (STT)
   ↓
NLU → Intent
   ↓
Dialog Manager (state machine)
   ↓
response_templates
   ↓
gTTS → GSM
   ↓
Playback στον χρήστη
```

Η ροή ξεκινά με την εισερχόμενη τηλεφωνική κλήση, η οποία διαχειρίζεται αρχικά από το Asterisk. Σύμφωνα με τη ρύθμιση του VICIdial, η κλήση δρομολογείται στο Voicebot και ενεργοποιείται το AGI script vici_voicebot.py. Από το σημείο αυτό και έπειτα, ο πλήρης έλεγχος της κλήσης μεταφέρεται στο AGI script, το οποίο καθορίζει τη συμπεριφορά του συστήματος.

Στο πρώτο στάδιο, το Voicebot αποστέλλει στον χρήστη ένα μήνυμα υποδοχής (welcome message), το οποίο παράγεται μέσω του υποσυστήματος Text-to-Speech και αναπαράγεται άμεσα. Το μήνυμα αυτό λειτουργεί ως εισαγωγή στη συνομιλία και ενημερώνει τον χρήστη ότι μπορεί να εκφράσει το αίτημά του. Στη συνέχεια, το AGI script ενεργοποιεί τη διαδικασία ηχογράφησης της φωνής του χρήστη. Η ηχογράφηση πραγματοποιείται με συγκεκριμένες παραμέτρους (χρονικό όριο, χαρακτήρες τερματισμού), ώστε να αποφεύγονται ανεπιθύμητες παύσεις ή υπερβολικά μεγάλα ηχητικά αρχεία. Το παραγόμενο αρχείο ήχου αποτελεί την είσοδο για το υποσύστημα αναγνώρισης ομιλίας. Το ηχητικό αρχείο προωθείται στο Whisper, το οποίο εκτελεί τη διαδικασία Speech-to-Text και επιστρέφει το αντίστοιχο κείμενο. Το αποτέλεσμα αυτό αντιπροσωπεύει τη λεκτική πρόθεση του χρήστη σε μορφή που μπορεί να επεξεργαστεί υπολογιστικά το σύστημα.

Στο επόμενο στάδιο, το κείμενο εισόδου αποστέλλεται στο υποσύστημα NLU, το οποίο αναλύει το περιεχόμενο και προσδιορίζει την πρόθεση (intent) του χρήστη. Η πληροφορία αυτή αποτελεί το βασικό κριτήριο για τη λήψη απόφασης σχετικά με τη συνέχιση της συνομιλίας. Στη συνέχεια ο μηχανισμός διαχείρισης διαλόγου (Dialog Manager), βασισμένος σε state machine, συνδυάζει την αναγνωρισμένη πρόθεση με την τρέχουσα κατάσταση της συνομιλίας. Με βάση το αποτέλεσμα αυτό, αποφασίζεται το επόμενο βήμα, όπως η συλλογή πρόσθετων στοιχείων, η μετάβαση σε νέα κατάσταση ή η ολοκλήρωση της διαδικασίας. Το σύστημα επιλέγει το κατάλληλο πρότυπο απόκρισης από το αρχείο response_templates, είτε πρόκειται για σταθερό μήνυμα είτε για δυναμικά διαμορφωμένη απάντηση που ενσωματώνει δεδομένα της συνομιλίας. Το κείμενο της απόκρισης αποστέλλεται στο υποσύστημα Text-to-Speech, όπου μετατρέπεται σε ηχητικό αρχείο μέσω του gTTS και στη συνέχεια σε μορφή GSM, συμβατή με το τηλεφωνικό περιβάλλον.

Το παραγόμενο ηχητικό μήνυμα αναπαράγεται στον χρήστη μέσω του Asterisk, ολοκληρώνοντας έναν πλήρη κύκλο διαλόγου. Ανάλογα με τη ροή και την κατάσταση της συνομιλίας, ο κύκλος αυτός μπορεί να επαναληφθεί πολλαπλές φορές μέχρι την ολοκλήρωση της αλληλεπίδρασης ή τον τερματισμό της κλήσης. Η συγκεκριμένη αλληλουχία υλοποιεί έναν πλήρη και επαναληπτικό κύκλο διαλόγου, στον οποίο κάθε στάδιο λειτουργεί ως ανεξάρτητη μονάδα, αλλά συνδέεται λειτουργικά με τα υπόλοιπα μέσω του AGI script. Η καθαρή αυτή ροή εκτέλεσης εξασφαλίζει τη σταθερότητα του συστήματος, τη δυνατότητα εντοπισμού σφαλμάτων και τη διατήρηση φυσικού ρυθμού συνομιλίας σε πραγματικές τηλεφωνικές συνθήκες.

### 5.2.2 Βασικός Σκελετός AGI Script

Κατά την εκκίνηση του Voicebot, το AGI script πραγματοποιεί αρχικοποίηση της συνεδρίας κλήσης και καταγράφει διαγνωστικά μηνύματα στο περιβάλλον του Asterisk. Η καταγραφή αυτή είναι ιδιαίτερα σημαντική σε παραγωγικά τηλεφωνικά συστήματα, καθώς παρέχει τη δυνατότητα παρακολούθησης της ροής εκτέλεσης, εντοπισμού καθυστερήσεων και αποσφαλμάτωσης σε περιπτώσεις μη αναμενόμενης συμπεριφοράς (π.χ. αποτυχία ηχογράφησης, σφάλμα μετατροπής ήχου ή αποτυχία φόρτωσης μοντέλου).

Η πρώτη ενέργεια του script είναι η εκτύπωση ενός διαγνωστικού μηνύματος στο log του Asterisk, το οποίο λειτουργεί ως «σήμα εισόδου» (entry point) ότι η κλήση δρομολογήθηκε επιτυχώς στο Voicebot και ότι ξεκινά η λογική διαλόγου:

```
agi.verbose("VBOT: enter voicebot dialog")
```

Με τη χρήση της verbose, το σύστημα μπορεί να καταγράφει πληροφορίες σε επίπεδο συνεδρίας, οι οποίες επιτρέπουν στον διαχειριστή να επιβεβαιώσει ότι το AGI εκτελείται, να συσχετίσει την εκτέλεση με συγκεκριμένη κλήση και να παρακολουθήσει τη ροή βήμα-βήμα. Σε πραγματικά περιβάλλοντα call center, αυτή η δυνατότητα είναι κρίσιμη, διότι η διόρθωση λαθών πρέπει να γίνεται χωρίς να διακόπτεται η λειτουργία του τηλεφωνικού κέντρου.

Στη συνέχεια, το Voicebot αποστέλλει στον χρήστη ένα αρχικό μήνυμα υποδοχής (welcome message). Το μήνυμα αυτό έχει διττό ρόλο: αφενός ενημερώνει τον χρήστη ότι η κλήση εξυπηρετείται από αυτοματοποιημένο σύστημα, αφετέρου δίνει σαφή οδηγία ότι ο χρήστης μπορεί να μιλήσει φυσικά, εκφράζοντας το αίτημά του ελεύθερα. Με τον τρόπο αυτό, μειώνεται η αβεβαιότητα του χρήστη και αυξάνεται η πιθανότητα να ξεκινήσει ο διάλογος με κατάλληλη φράση, διευκολύνοντας το STT και το NLU.

Το μήνυμα υποδοχής παράγεται δυναμικά μέσω του υποσυστήματος TTS και αποθηκεύεται σε μορφή συμβατή με το Asterisk, ώστε να αναπαραχθεί άμεσα στον χρήστη:

```
# Welcome TTS
welcome_text = "Καλώς ήρθατε. Πείτε μου πώς μπορώ να σας βοηθήσω."

tts_path = synthesize_tts(welcome_text, "welcome", gsm=True)

agi.exec_command("Playback", f"voicebot/{tts_path}")
```

Η μεταβλητή welcome_text περιέχει το κείμενο της αρχικής απόκρισης. Η συνάρτηση synthesize_tts αναλαμβάνει τη δημιουργία του αντίστοιχου ηχητικού αρχείου (π.χ. παραγωγή με gTTS και μετατροπή σε GSM), επιστρέφοντας τη διαδρομή (path) του αρχείου που θα αναπαραχθεί. Η παράμετρος gsm=True υποδηλώνει ότι το τελικό αρχείο πρέπει να είναι σε μορφή κατάλληλη για το τηλεφωνικό περιβάλλον.

Τέλος, η εντολή:

```
agi.exec_command("Playback", f"voicebot/{tts_path}")
```

πραγματοποιεί την αναπαραγωγή του ηχητικού αρχείου στον χρήστη μέσω του Asterisk. Η χρήση της Playback είναι κατάλληλη για μηνύματα που πρέπει να ακουστούν ολόκληρα πριν ξεκινήσει η επόμενη φάση (ηχογράφηση χρήστη), διασφαλίζοντας ότι ο καλών ακούει πλήρως την οδηγία και δεν μιλά ταυτόχρονα με το μήνυμα του συστήματος. Ο βασικός σκελετός εκκίνησης του AGI script υλοποιεί δύο θεμελιώδεις λειτουργίες: (α) την καταγραφή κρίσιμων διαγνωστικών πληροφοριών στο Asterisk log και (β) την έναρξη της συνομιλίας με ένα σαφές και καθοδηγητικό μήνυμα υποδοχής. Το στάδιο αυτό αποτελεί τη βάση πάνω στην οποία χτίζεται ο υπόλοιπος κύκλος διαλόγου, καθώς ακολουθεί άμεσα η ηχογράφηση της απάντησης του χρήστη και η αλυσίδα STT → NLU → Dialog Management → TTS.

### 5.2.3 Ηχογράφηση φωνής από τον χρήστη

Η καταγραφή της ομιλίας του χρήστη αποτελεί ένα από τα πιο κρίσιμα στάδια της τηλεφωνικής αλληλεπίδρασης, καθώς το παραγόμενο ηχητικό σήμα αποτελεί την είσοδο για το υποσύστημα αναγνώρισης ομιλίας (STT). Η ποιότητα, η διάρκεια και η σωστή διαχείριση της ηχογράφησης επηρεάζουν άμεσα την ακρίβεια της μεταγραφής και, κατ' επέκταση, τη συνολική απόδοση του Voicebot. Η ηχογράφηση πραγματοποιείται μέσω της εντολής record_file της διεπαφής AGI, η οποία επιτρέπει στο AGI script να καταγράψει την ομιλία του συνομιλητή απευθείας από την τηλεφωνική γραμμή. Ο Asterisk αποθηκεύει αρχικά το ηχητικό σήμα σε μορφή WAV με ρυθμό δειγματοληψίας 8 kHz, σύμφωνα με τα πρότυπα τηλεφωνικής ποιότητας. Η επιλογή αυτή είναι σύμφωνη με τα χαρακτηριστικά των τηλεφωνικών codecs και διασφαλίζει συμβατότητα με το τηλεφωνικό περιβάλλον.

Η κλήση της συνάρτησης record_file υλοποιείται ως εξής:

```
res, digit, record_ms = agi.record_file(

    filename=wavpath_raw,
    format="wav",
    escape_digits="1234567890#",
    timeout=5000,
    beep="beep"

)
```

Η παράμετρος filename καθορίζει τη διαδρομή και το όνομα του αρχείου στο οποίο θα αποθηκευτεί η ηχογράφηση. Η επιλογή της μορφής wav επιτρέπει την απώλεια-ελεύθερη αποθήκευση του σήματος, γεγονός που είναι σημαντικό για την επόμενη φάση επεξεργασίας. Η παράμετρος escape_digits ορίζει τα πλήκτρα τα οποία μπορούν να χρησιμοποιηθούν από τον χρήστη για τον πρόωρο τερματισμό της ηχογράφησης, παρέχοντας έτσι έναν μηχανισμό ελέγχου και διακοπής της εισόδου. Το timeout καθορίζει τη μέγιστη διάρκεια της ηχογράφησης σε milliseconds, αποτρέποντας υπερβολικά μεγάλα αρχεία ή ατέρμονες παύσεις. Τέλος, η παράμετρος beep ενεργοποιεί έναν ηχητικό τόνο πριν την έναρξη της καταγραφής, ώστε ο χρήστης να γνωρίζει πότε μπορεί να ξεκινήσει να μιλά.

Το αποτέλεσμα της ηχογράφησης επιστρέφεται μέσω των μεταβλητών res, digit και record_ms, οι οποίες παρέχουν πληροφορίες σχετικά με την έκβαση της καταγραφής, τον πιθανό χαρακτήρα διακοπής και τη διάρκεια της ομιλίας. Τα δεδομένα αυτά μπορούν να αξιοποιηθούν για διαγνωστικούς σκοπούς ή για την προσαρμογή της ροής του διαλόγου. Παρότι η τηλεφωνική ηχογράφηση πραγματοποιείται στα 8 kHz, το μοντέλο Whisper απαιτεί είσοδο με ρυθμό δειγματοληψίας 16 kHz για τη βέλτιστη λειτουργία του. Για τον λόγο αυτό, το παραγόμενο αρχείο ήχου μετατρέπεται σε WAV 16 kHz με τη χρήση του εργαλείου *sox*. Η μετατροπή αυτή αποτελεί κρίσιμο ενδιάμεσο βήμα, καθώς διασφαλίζει τη συμβατότητα με το STT μοντέλο και μεγιστοποιεί την ακρίβεια της αναγνώρισης ομιλίας.

Η διαδικασία μετατροπής υλοποιείται ως εξής:

```
subprocess.run(
    ["sox", wav_8k, "-r", "16000", wav_16k], check=True
)
```

Η χρήση του *sox* επιτρέπει την αξιόπιστη επαναδειγματοληψία (resampling) του ηχητικού σήματος, χωρίς σημαντική αλλοίωση του περιεχομένου της ομιλίας. Η επιλογή της παραμέτρου check=True διασφαλίζει ότι τυχόν σφάλματα κατά τη μετατροπή θα εντοπιστούν άμεσα, αποτρέποντας τη συνέχιση της ροής με μη έγκυρα δεδομένα. Το στάδιο της ηχογράφησης και της μετατροπής του ήχου λειτουργεί ως κρίσιμη γέφυρα μεταξύ του τηλεφωνικού περιβάλλοντος και του υποσυστήματος αναγνώρισης ομιλίας. Η σωστή παραμετροποίηση της διαδικασίας καταγραφής, σε συνδυασμό με τη μετατροπή του ηχητικού σήματος σε κατάλληλη μορφή συμβάλλει καθοριστικά στη σταθερότητα και στην ακρίβεια του Voicebot επιτρέποντας τη φυσική και αξιόπιστη τηλεφωνική αλληλεπίδραση σε πραγματικές κλήσεις.

### 5.2.4 Speech-to-Text (Whisper) εντός AGI

Το υποσύστημα αναγνώρισης ομιλίας (Speech-to-Text – STT) ενσωματώνεται απευθείας στο AGI script και εκτελείται τοπικά στον VICIdial server, χωρίς μεσολάβηση εξωτερικών υπηρεσιών ή αποστολή δεδομένων σε cloud υποδομές. Η επιλογή της τοπικής εκτέλεσης (on-premise STT), ενισχύει τον έλεγχο επί των δεδομένων, μειώνει τις εξαρτήσεις από τρίτους παρόχους και επιτρέπει τη λειτουργία ακόμη και σε περιβάλλοντα με περιορισμένη ή μη αξιόπιστη δικτυακή πρόσβαση.

Για την αναγνώριση ελληνικής ομιλίας χρησιμοποιείται το μοντέλο Whisper στην έκδοση *small*. Το συγκεκριμένο μοντέλο επιλέχθηκε ως ένας πρακτικός συμβιβασμός μεταξύ ακρίβειας και υπολογιστικού κόστους. Σε τηλεφωνικές εφαρμογές, η απόκριση πρέπει να είναι αρκετά γρήγορη ώστε να μην δημιουργούνται μεγάλες παύσεις στον διάλογο, ενώ παράλληλα απαιτείται ικανοποιητική ακρίβεια ώστε να μην μεταφέρονται λάθη στα επόμενα στάδια (NLU και διαχείριση διαλόγου). Η χρήση CPU-only εκτέλεσης αντανακλά ρεαλιστικές συνθήκες εγκατάστασης σε τηλεφωνικά κέντρα, όπου δεν είναι δεδομένη η ύπαρξη GPU για επιτάχυνση.

Η αρχικοποίηση του μοντέλου πραγματοποιείται ως εξής:

```
whisper_model = WhisperModel("small", device="cpu")

segments, info = whisper_model.transcribe(
    wav_16k,
    language="el",
    beam_size=1,
    temperature=0.0
)
```

Με την επιλογή device="cpu", το μοντέλο φορτώνεται στη μνήμη και εκτελεί inference μέσω του επεξεργαστή. Η ενσωμάτωση στο AGI script επιτρέπει τη χρήση του Whisper ως ενσωματωμένης λειτουργίας μέσα στον κύκλο κλήσης, χωρίς επιπλέον ενδιάμεσα συστήματα.

Η διαδικασία μεταγραφής εκτελείται με κλήση της μεθόδου transcribe:

```
segments, info = whisper_model.transcribe(
    wav_16k,
    language="el",
    beam_size=1,
    temperature=0.0
)
```

Η είσοδος wav_16k αντιστοιχεί στο ηχητικό αρχείο που έχει ήδη μετατραπεί σε 16 kHz, ώστε να είναι συμβατό με τις απαιτήσεις του Whisper. Η παράμετρος language="el" επιβάλλει στο σύστημα την εστίαση στην ελληνική γλώσσα, περιορίζοντας σφάλματα που μπορεί να προκύψουν από αυτόματη ανίχνευση γλώσσας, ειδικά σε τηλεφωνικό ήχο χαμηλής ποιότητας. Οι παράμετροι beam_size και temperature επηρεάζουν τη διαδικασία αποκωδικοποίησης (decoding). Με beam_size=1, το σύστημα λειτουργεί πρακτικά σε greedy decoding, επιλέγοντας την πιο πιθανή ακολουθία χαρακτήρων/λέξεων χωρίς εκτεταμένη εξερεύνηση εναλλακτικών. Η επιλογή αυτή μειώνει σημαντικά το υπολογιστικό κόστος και τον χρόνο απόκρισης, κάτι ιδιαίτερα σημαντικό για real-time τηλεφωνικές εφαρμογές. Αντίστοιχα, η τιμή temperature=0.0 περιορίζει τη στοχαστικότητα στην αποκωδικοποίηση, οδηγώντας σε πιο σταθερά και επαναληπτικά αποτελέσματα, στοιχείο που είναι επιθυμητό σε παραγωγικά συστήματα όπου απαιτείται συνεπής συμπεριφορά.

Το αποτέλεσμα της μεταγραφής επιστρέφεται σε δύο μέρη:

- segments: τμήματα της αναγνωρισμένης ομιλίας (π.χ. προτάσεις ή φράσεις) με το κείμενό τους,
- info: μετα-πληροφορίες για τη διαδικασία αναγνώρισης, όπως ενδείξεις βεβαιότητας (confidence), πιθανή γλώσσα και στοιχεία χρονισμού.

Η ύπαρξη του info είναι ιδιαίτερα χρήσιμη, καθώς επιτρέπει τη μελλοντική αξιολόγηση της απόδοσης του συστήματος, τόσο σε επίπεδο ποιότητας αναγνώρισης όσο και σε επίπεδο χρόνου επεξεργασίας. Για παράδειγμα, σε περιπτώσεις όπου το σύστημα λαμβάνει χαμηλή βεβαιότητα αναγνώρισης, μπορεί να εφαρμοστεί εναλλακτική συμπεριφορά, όπως επανάληψη της ερώτησης ή αίτημα για πιο καθαρή εκφώνηση. Για σκοπούς παρακολούθησης και αποσφαλμάτωσης, το AGI script καταγράφει το αποτέλεσμα της αναγνώρισης σε μορφή log. Ένα χαρακτηριστικό παράδειγμα είναι:

```
VBOT: STT -> 'θέλω να κάνω μια παραγγελία' lang=el prob=1.00
```

Η καταγραφή αυτή επιτρέπει την αντιστοίχιση της εισόδου (φωνή) με την έξοδο (κείμενο) σε πραγματικές κλήσεις, διευκολύνοντας την αποτίμηση της λειτουργίας και την ανίχνευση περιπτώσεων όπου η τηλεφωνική ποιότητα, ο θόρυβος ή η προφορά επηρεάζουν αρνητικά το αποτέλεσμα.

Η ενσωμάτωση του Whisper στο AGI script παρέχει ένα αξιόπιστο και πλήρως on-premise STT υποσύστημα, κατάλληλο για τηλεφωνικές εφαρμογές. Η χρήση του Whisper small σε CPU, σε συνδυασμό με παραμέτρους που ευνοούν χαμηλό latency και σταθερότητα, επιτρέπει τη λειτουργία του Voicebot σε πραγματικό χρόνο, αποτελώντας το θεμέλιο για την επιτυχή ανάλυση πρόθεσης και τη συνέχεια του διαλόγου.

### 5.2.5 Κλήση NLU Μοντέλου

Μετά την ολοκλήρωση της διαδικασίας αναγνώρισης ομιλίας, το κείμενο που παράγεται από το υποσύστημα Speech-to-Text προωθείται στο υποσύστημα κατανόησης φυσικής γλώσσας (NLU). Το στάδιο αυτό αποτελεί το σημείο μετάβασης από την απλή μεταγραφή της ομιλίας σε ουσιαστική ερμηνεία του νοήματος της εισερχόμενης πρότασης και είναι καθοριστικό για τη συμπεριφορά του Voicebot.

Η επικοινωνία του AGI script με το NLU υλοποιείται μέσω κλήσης συνάρτησης στο αρχείο nlu_intent_pt.py, το οποίο περιέχει το εκπαιδευμένο μοντέλο ταξινόμησης προθέσεων. Η κλήση πραγματοποιείται με απλό και καθαρό τρόπο:

```
intent, slots = nlu_intent_pt.predict_intent(user_text)
```

Η μεταβλητή user_text αντιστοιχεί στο κείμενο που παρήχθη από το Whisper και αποτελεί την είσοδο του NLU μοντέλου. Η συνάρτηση predict_intent εκτελεί όλα τα απαραίτητα βήματα προεπεξεργασίας (κανονικοποίηση, μετατροπή σε διανυσματική αναπαράσταση) και στη συνέχεια εφαρμόζει το νευρωνικό μοντέλο για την ταξινόμηση της πρότασης. Το αποτέλεσμα της κλήσης επιστρέφεται σε δύο μέρη. Το πρώτο στοιχείο, intent, αντιπροσωπεύει την αναγνωρισμένη πρόθεση του χρήστη και αποτελεί μια συμβολική ετικέτα (label) που περιγράφει το είδος του αιτήματος. Το δεύτερο στοιχείο, slots, χρησιμοποιείται για την αποθήκευση βασικών πληροφοριών που μπορούν να εξαχθούν από την πρόταση, όπως αριθμοί, προϊόντα ή άλλα δομημένα δεδομένα, εφόσον αυτό απαιτείται από τη ροή του διαλόγου.

Η αναγνωρισμένη πρόθεση καθορίζει άμεσα τη ροή της συνομιλίας και αποτελεί το βασικό σήμα εισόδου για τον μηχανισμό διαχείρισης διαλόγου (Dialog Manager). Ανάλογα με το intent, το Voicebot μπορεί να μεταβεί σε διαφορετική κατάσταση της state machine, να ζητήσει επιπλέον πληροφορίες ή να παράσχει άμεση απάντηση.

Ενδεικτικά παραδείγματα προθέσεων που υποστηρίζει το σύστημα είναι:

- new_order, για την εκκίνηση διαδικασίας νέας παραγγελίας.
- product, για αναφορά ή επιλογή προϊόντος.
- order_status, για ερώτημα σχετικά με την κατάσταση παραγγελίας.
- greeting, για χαιρετισμούς ή αρχική επαφή.
- pricing, για ερωτήματα σχετικά με τιμές ή κόστος.

Η χρήση σαφώς ορισμένων προθέσεων επιτρέπει στο Voicebot να αντιδρά με τρόπο προσαρμοσμένο στο αίτημα του χρήστη, αποφεύγοντας ασαφείς ή γενικές απαντήσεις. Παράλληλα, διευκολύνει την επέκταση του συστήματος, καθώς νέες λειτουργίες μπορούν να προστεθούν απλώς με την εισαγωγή νέων intents και παραδειγμάτων εκπαίδευσης, χωρίς αλλαγές στη βασική αρχιτεκτονική. Ιδιαίτερη σημασία έχει το γεγονός ότι το NLU μοντέλο καλείται απευθείας από το AGI script και εκτελείται τοπικά, χωρίς καθυστερήσεις που θα προέκυπταν από δικτυακές κλήσεις σε εξωτερικές υπηρεσίες. Η άμεση αυτή ενσωμάτωση συμβάλλει στη διατήρηση χαμηλού latency και καθιστά το σύστημα κατάλληλο για real-time τηλεφωνική αλληλεπίδραση.

Το στάδιο κλήσης του NLU μοντέλου αποτελεί τον πυρήνα της «νοημοσύνης» του Voicebot. Μετατρέπει το ακατέργαστο κείμενο σε δομημένη πληροφορία, η οποία μπορεί να αξιοποιηθεί από τον μηχανισμό διαλόγου για τη λήψη αποφάσεων και τη συνέχιση της συνομιλίας με τρόπο φυσικό και στοχευμένο.

### 5.2.6 Διαχείριση Διαλόγου (State Machine)

Η διαχείριση της συνομιλίας μεταξύ χρήστη και Voicebot υλοποιείται μέσω ενός μηχανισμού διακριτών καταστάσεων (state machine), ο οποίος επιτρέπει τον έλεγχο της ροής του διαλόγου με δομημένο και προβλέψιμο τρόπο. Η επιλογή αυτής της προσέγγισης κρίθηκε κατάλληλη, καθώς το σύστημα καλείται να υποστηρίξει σύνθετες διαδικασίες πολλαπλών βημάτων, όπως η παραγγελιοληψία, όπου απαιτείται η συλλογή συγκεκριμένων πληροφοριών σε προκαθορισμένη σειρά.

Ο μηχανισμός state machine λειτουργεί ως λογικό επίπεδο πάνω από το NLU, αξιοποιώντας την αναγνωρισμένη πρόθεση του χρήστη αλλά και το ιστορικό της συνομιλίας ώστε να αποφασίσει ποιο θα είναι το επόμενο βήμα. Κάθε κατάσταση (state) αντιστοιχεί σε ένα συγκεκριμένο στάδιο της συνομιλίας και καθορίζει:

- το μήνυμα (prompt) που θα αποσταλεί στον χρήστη.
- το είδος της πληροφορίας που αναμένεται ως απάντηση.
- τους κανόνες επικύρωσης της εισόδου.
- τις επιτρεπτές μεταβάσεις προς επόμενες καταστάσεις.

Η υλοποίηση του μηχανισμού πραγματοποιείται απευθείας στο AGI script, με χρήση απλής λογικής ελέγχου ροής. Ενδεικτικό παράδειγμα μετάβασης κατάστασης φαίνεται στο παρακάτω απόσπασμα:

```
if state == "ASK_NAME":
    order["name"] = user_text
    state = "ASK_PHONE"
```

Στο συγκεκριμένο παράδειγμα, όταν το σύστημα βρίσκεται στην κατάσταση ASK_NAME, η απάντηση του χρήστη αποθηκεύεται στο προσωρινό context της παραγγελίας και το state μεταβαίνει στην επόμενη κατάσταση, όπου ζητείται ο αριθμός τηλεφώνου. Με αυτόν τον τρόπο διασφαλίζεται ότι κάθε πληροφορία συλλέγεται με σαφή και ελεγχόμενο τρόπο.

Για τη ροή «Νέα Παραγγελία», το state machine ακολουθεί την εξής αλληλουχία καταστάσεων:

1. **Εκκίνηση διαδικασίας**
   Μετά την ανίχνευση της πρόθεσης new_order, το σύστημα μεταβαίνει σε κατάσταση συλλογής στοιχείων.
2. **ASK_NAME – Ζητείται το όνομα**
   Το Voicebot ζητά το ονοματεπώνυμο του χρήστη και αποθηκεύει την απάντηση.
3. **ASK_PHONE – Ζητείται το τηλέφωνο**
   Ζητείται αριθμός επικοινωνίας, ο οποίος μπορεί να αξιοποιηθεί για επιβεβαίωση ή μελλοντική αναφορά.
4. **ASK_PRODUCT – Επιλογή προϊόντος**
   Ο χρήστης δηλώνει το προϊόν που επιθυμεί να παραγγείλει.

5. **ASK_QUANTITY – Ποσότητα προϊόντος**
   Ζητείται η ποσότητα του προϊόντος, με δυνατότητα βασικού ελέγχου εγκυρότητας.
6. **CONFIRMATION – Επιβεβαίωση στοιχείων**
   Το Voicebot συνοψίζει τα στοιχεία της παραγγελίας και ζητά επιβεβαίωση από τον χρήστη.
7. **ORDER_LOGGING – Καταχώριση παραγγελίας**
   Τα δεδομένα αποθηκεύονται σε δομημένη μορφή και η διαδικασία ολοκληρώνεται.
8. **TERMINATION – Τερματισμός διαλόγου**
   Παράγεται τελικό μήνυμα και η κλήση ολοκληρώνεται.

Η χρήση state machine επιτρέπει στο σύστημα να διατηρεί συμφραζόμενα (context) καθ' όλη τη διάρκεια της συνομιλίας, αποφεύγοντας ασυνέπειες ή απώλεια πληροφορίας. Επιπλέον καθιστά τη ροή του διαλόγου εύκολα κατανοητή, καθώς κάθε κατάσταση υλοποιείται και ελέγχεται ανεξάρτητα. Σημαντικό πλεονέκτημα της προσέγγισης αυτής είναι ότι δεν απαιτεί τη χρήση εξωτερικών frameworks διαχείρισης διαλόγου, όπως τα Dialogflow ή Rasa. Αντίθετα, η λογική διαλόγου παραμένει πλήρως υπό τον έλεγχο του συστήματος, επιτρέποντας μεγαλύτερη ευελιξία, προσαρμογή στις ανάγκες του εκάστοτε οργανισμού και πλήρη on-premise λειτουργία.

Ο μηχανισμός state machine αποτελεί το βασικό εργαλείο για τη μετάβαση από απλές ερωταποκρίσεις σε ουσιαστικό, δομημένο και λειτουργικό τηλεφωνικό διάλογο, καθιστώντας το Voicebot ικανό να διαχειρίζεται πραγματικές επιχειρησιακές διαδικασίες.

### 5.2.7 Παραγωγή Απάντησης (TTS)

Η παραγωγή της φωνητικής απάντησης αποτελεί το τελικό στάδιο κάθε κύκλου διαλόγου και είναι καθοριστική για την ποιότητα της εμπειρίας του χρήστη. Αφού το σύστημα αναγνωρίσει την πρόθεση του χρήστη και καθορίσει την επόμενη ενέργεια μέσω του μηχανισμού διαλόγου, το Voicebot οφείλει να μετατρέψει την επιλεγμένη απάντηση από κείμενο σε φυσική ομιλία, ώστε να διατηρηθεί η ψευδαίσθηση της ανθρώπινης συνομιλίας.

Η διαδικασία αυτή υλοποιείται μέσω του υποσυστήματος Text-to-Speech (TTS), το οποίο αξιοποιεί προκαθορισμένα ή δυναμικά παραγόμενα κείμενα απόκρισης. Το κείμενο που θα εκφωνηθεί επιλέγεται με βάση:

- την αναγνωρισμένη πρόθεση (intent),
- την τρέχουσα κατάσταση του διαλόγου (state),
- καθώς και το αποθηκευμένο συμφραζόμενο (context) της συνομιλίας.

Στο AGI script, η επιλογή του κειμένου πραγματοποιείται μέσω μηχανισμού προτύπων (response templates), επιτρέποντας συνεπείς και ελεγχόμενες απαντήσεις. Ενδεικτικά:

```
resp_text = template_for(intent)
tts_path = synthesize_tts(resp_text, f"bot_t{turn}", gsm=True)
agi.exec_command("Playback", f"voicebot/{tts_path}")
```

Στο παραπάνω απόσπασμα, η συνάρτηση template_for επιστρέφει το κατάλληλο κείμενο απόκρισης, το οποίο στη συνέχεια περνά στο υποσύστημα TTS. Η συνάρτηση synthesize_tts αναλαμβάνει τη δημιουργία του ηχητικού αρχείου, χρησιμοποιώντας το gTTS για τη σύνθεση ελληνικής φωνής. Το παραγόμενο αρχείο μετατρέπεται αυτόματα σε μορφή GSM, ώστε να είναι πλήρως συμβατό με το τηλεφωνικό περιβάλλον του Asterisk.

Η τελική αναπαραγωγή της φωνής πραγματοποιείται μέσω της εντολής Playback του AGI, η οποία αποστέλλει το ηχητικό μήνυμα στον καλούντα. Με τον τρόπο αυτό, το Voicebot «απαντά» στον χρήστη με φυσικό και κατανοητό τρόπο, χωρίς να απαιτείται ανθρώπινη παρέμβαση. Ιδιαίτερη έμφαση δόθηκε στον χρόνο παραγωγής του ήχου, καθώς καθυστερήσεις στο στάδιο TTS μπορούν να διακόψουν τη φυσική ροή της συνομιλίας. Το gTTS ανταποκρίνεται ικανοποιητικά στις απαιτήσεις του συστήματος, επιτρέποντας την παραγωγή σύντομων φωνητικών απαντήσεων σε πραγματικό χρόνο. Παράλληλα, η χρήση προτύπων απαντήσεων μειώνει την ανάγκη για επαναλαμβανόμενη δημιουργία παρόμοιων μηνυμάτων, βελτιώνοντας τη συνολική απόδοση.

Η ολοκλήρωση της διαδικασίας TTS σηματοδοτεί το τέλος ενός κύκλου διαλόγου. Ανάλογα με τη λογική του state machine, το σύστημα μπορεί είτε να μεταβεί στην επόμενη κατάσταση και να συνεχίσει τη συνομιλία, είτε να παράγει τελικό μήνυμα και να τερματίσει την κλήση. Με τον τρόπο αυτό, το υποσύστημα TTS λειτουργεί ως το τελικό «εκφραστικό» μέσο του Voicebot, μετατρέποντας τη λογική απόφαση του συστήματος σε ακουστική εμπειρία για τον χρήστη.

## 5.3 Υλοποίηση NLU Μοντέλου (nlu_intent_pt.py)

Το υποσύστημα κατανόησης φυσικής γλώσσας (Natural Language Understanding – NLU) αποτελεί ένα από τα πλέον κρίσιμα δομικά στοιχεία του Voicebot, καθώς είναι υπεύθυνο για τη μετάβαση από το επίπεδο της απλής μεταγραφής ομιλίας (STT output) στο επίπεδο της ερμηνείας του νοήματος και της πρόθεσης του χρήστη. Ενώ το STT απαντά στο ερώτημα «τι ειπώθηκε», το NLU απαντά στο ερώτημα «τι εννοεί ο χρήστης» και συνεπώς καθορίζει άμεσα τη συμπεριφορά του συστήματος και τη συνέχεια της συνομιλίας.

Στην παρούσα εργασία υλοποιήθηκε ένα custom, εκπαιδεύσιμο και ελληνόφωνο NLU μοντέλο, σχεδιασμένο ειδικά για το σενάριο τηλεφωνικού διαλόγου, όπου οι απαιτήσεις σε χρόνο απόκρισης (latency) είναι αυστηρές και οι προτάσεις των χρηστών είναι συχνά σύντομες, προφορικές και όχι πάντα γραμματικά «καθαρές». Ο σχεδιασμός του μοντέλου έλαβε υπόψη ότι το κείμενο εισόδου προέρχεται από αναγνώριση ομιλίας, γεγονός που σημαίνει ότι μπορεί να περιέχει:

- μικρά σφάλματα μεταγραφής,
- παραλείψεις ή ασάφειες,
- διαφορετικές διατυπώσεις της ίδιας πρόθεσης,
- προφορικές εκφράσεις και ιδιωματισμούς.

Η ανάπτυξη ιδιόκτητου NLU μοντέλου, αντί της χρήσης έτοιμων frameworks ή εξωτερικών πλατφορμών, αποτέλεσε συνειδητή σχεδιαστική επιλογή με στόχο την αυτονομία και τον πλήρη έλεγχο της λειτουργίας του συστήματος. Συγκεκριμένα, η επιλογή αυτή προσφέρει:

- Πλήρη έλεγχο της διαδικασίας εκπαίδευσης και αξιολόγησης.
  Ο τρόπος με τον οποίο προεπεξεργάζεται το κείμενο, δημιουργείται το λεξιλόγιο, εκπαιδεύεται το μοντέλο και αξιολογείται η απόδοση είναι πλήρως ορατός και παραμετροποιήσιμος. Αυτό επιτρέπει στο σύστημα να προσαρμόζεται σε πραγματικά δεδομένα χρήσης.

- Προσαρμογή στη συγκεκριμένη θεματολογία του Voicebot.
  Τα τηλεφωνικά σενάρια (π.χ. παραγγελιοληψία, τιμές, ωράρια, κατάσταση παραγγελίας) απαιτούν εξειδικευμένες προθέσεις (intents) και λεξιλόγιο. Ένα προσαρμοσμένο μοντέλο μπορεί να εκπαιδευτεί στοχευμένα ώστε να αποδίδει καλύτερα στις συγκεκριμένες ανάγκες του έργου.

- Εύκολη επανεκπαίδευση με νέα δεδομένα.
  Η λειτουργία του Voicebot σε πραγματικές συνθήκες οδηγεί στην εμφάνιση νέων διατυπώσεων και φράσεων από τους χρήστες. Το σύστημα σχεδιάστηκε ώστε να επιτρέπει τη συνεχή εμπλουτισμό του dataset και την επανεκπαίδευση του μοντέλου, βελτιώνοντας σταδιακά την ακρίβεια ταξινόμησης.

- Λειτουργία αποκλειστικά σε on-premise περιβάλλον.
  Η υλοποίηση του NLU σε τοπικό επίπεδο διασφαλίζει ότι η επεξεργασία του κειμένου δεν εξαρτάται από τρίτους παρόχους ή δικτυακές υπηρεσίες. Με αυτόν τον τρόπο επιτυγχάνεται ανεξαρτησία, μειωμένο κόστος και καλύτερη συμμόρφωση με πολιτικές προστασίας δεδομένων.

Επιπλέον, το NLU μοντέλο σχεδιάστηκε με στόχο τη χαμηλή υπολογιστική επιβάρυνση, ώστε να μπορεί να εκτελείται σε πραγματικό χρόνο παράλληλα με τα υπόλοιπα υποσυστήματα (STT και TTS). Η απλότητα της προσέγγισης είναι ιδιαίτερα σημαντική στο τηλεφωνικό περιβάλλον, όπου η καθυστέρηση στην απόκριση επηρεάζει άμεσα την αίσθηση φυσικής συνομιλίας. Για τον λόγο αυτό, επιλέχθηκε αρχιτεκτονική που βασίζεται σε διανυσματοποίηση Bag-of-Words και σε ελαφρύ νευρωνικό δίκτυο τύπου MLP, παρέχοντας ικανοποιητική απόδοση χωρίς υπερβολικές απαιτήσεις σε πόρους. Η υλοποίηση του NLU στο αρχείο nlu_intent_pt.py περιλαμβάνει όλα τα απαραίτητα στοιχεία για inference σε πραγματικό χρόνο: προεπεξεργασία κειμένου, μετατροπή σε διανυσματική αναπαράσταση, φόρτωση εκπαιδευμένου μοντέλου PyTorch, υπολογισμό πιθανοτήτων για κάθε intent και επιστροφή της τελικής πρόθεσης στο AGI script. Με τον τρόπο αυτό, το NLU λειτουργεί ως αποδοτικό ενδιάμεσο επίπεδο λήψης απόφασης, τροφοδοτώντας τον μηχανισμό διαλόγου με αξιόπιστη πληροφορία πρόθεσης.

Στις επόμενες υποενότητες παρουσιάζονται αναλυτικά: (α) ο τρόπος μετατροπής του κειμένου σε αριθμητική αναπαράσταση (vectorization), (β) η αρχιτεκτονική του μοντέλου και η διαδικασία inference, καθώς και (γ) τα αρχεία/δομές που αποθηκεύονται ώστε το μοντέλο να μπορεί να χρησιμοποιηθεί άμεσα από το AGI σε πραγματικό χρόνο.

### 5.3.1 Vectorization (Bag of Words)

Για να καταστεί δυνατή η επεξεργασία του φυσικού κειμένου από το νευρωνικό μοντέλο, απαιτείται η μετατροπή της γλωσσικής πληροφορίας σε αριθμητική αναπαράσταση. Στην παρούσα εργασία, η μετατροπή αυτή πραγματοποιείται με τη χρήση της τεχνικής Bag-of-Words (BoW), η οποία αποτελεί μία από τις πλέον κλασικές και αποδοτικές μεθόδους αναπαράστασης κειμένου σε προβλήματα ταξινόμησης προθέσεων.

Η διαδικασία ξεκινά με βασική προεπεξεργασία του εισερχόμενου κειμένου, η οποία περιλαμβάνει:

- τμηματοποίηση της πρότασης σε επιμέρους λέξεις (tokenization)
- κανονικοποίηση ελληνικών χαρακτήρων
- μετατροπή σε πεζά γράμματα
- αφαίρεση τόνων και ειδικών συμβόλων, όπου αυτό κρίνεται απαραίτητο

Σκοπός της προεπεξεργασίας είναι η μείωση της γλωσσικής ποικιλίας που δεν προσφέρει ουσιαστική πληροφορία στο μοντέλο, καθώς και η ενοποίηση διαφορετικών γραφών της ίδιας λέξης σε κοινή μορφή. Με τον τρόπο αυτό, το σύστημα καθίσταται πιο ανθεκτικό σε μικρές διαφοροποιήσεις έκφρασης που είναι συχνές στον προφορικό λόγο.

Μετά την προεπεξεργασία, κάθε πρόταση μετατρέπεται σε ένα διάνυσμα σταθερού μεγέθους, όπου κάθε διάσταση αντιστοιχεί σε μία λέξη του λεξιλογίου (vocabulary) που έχει δημιουργηθεί κατά την εκπαίδευση του μοντέλου. Η υλοποίηση της διαδικασίας αυτής φαίνεται στο ακόλουθο απόσπασμα κώδικα:

```
tokens = tokenize(text)
vec = np.zeros(len(vocab))
for t in tokens:
    if t in vocab:
        vec[vocab[t]] += 1
```

Στο παραπάνω παράδειγμα, αρχικά παράγεται η λίστα των tokens της πρότασης. Στη συνέχεια, δημιουργείται ένα μηδενικό διάνυσμα μήκους ίσου με το πλήθος των μοναδικών όρων του λεξιλογίου. Για κάθε token που εντοπίζεται στο λεξιλόγιο, αυξάνεται η αντίστοιχη θέση του διανύσματος, αποτυπώνοντας τη συχνότητα εμφάνισης της λέξης στην πρόταση. Η προσέγγιση Bag-of-Words αγνοεί τη σειρά των λέξεων και επικεντρώνεται αποκλειστικά στην παρουσία ή τη συχνότητα όρων. Παρότι η απώλεια συντακτικής πληροφορίας μπορεί να θεωρηθεί μειονέκτημα σε πιο σύνθετα γλωσσικά προβλήματα, στην περίπτωση ταξινόμησης προθέσεων για τηλεφωνικά σενάρια η μέθοδος αποδεικνύεται ιδιαίτερα αποτελεσματική.

Οι περισσότερες προθέσεις εκφράζονται μέσω συγκεκριμένων λέξεων-κλειδιών (π.χ. «παραγγελία», «τιμή», «κατάσταση»), οι οποίες αποτυπώνονται με σαφήνεια στη BoW αναπαράσταση.

Η επιλογή της BoW προσέγγισης στην παρούσα εργασία έγινε για μια σειρά από πρακτικούς και λειτουργικούς λόγους:

- είναι υπολογιστικά αποδοτική και επιτρέπει γρήγορο inference σε πραγματικό χρόνο.
- λειτουργεί ικανοποιητικά σε περιβάλλοντα με περιορισμένο και θεματικά εστιασμένο λεξιλόγιο.
- είναι κατάλληλη για intents χαμηλής έως μέτριας πολυπλοκότητας.

- διευκολύνει τη διαδικασία επανεκπαίδευσης του μοντέλου από μη εξειδικευμένους χρήστες, καθώς δεν απαιτεί πολύπλοκες τεχνικές προεπεξεργασίας.

Το παραγόμενο διάνυσμα είναι αραιό (sparse), καθώς για κάθε πρόταση μόνο ένα μικρό υποσύνολο των λέξεων του λεξιλογίου εμφανίζεται πραγματικά. Το μέγεθος του διανύσματος εξαρτάται άμεσα από το πλήθος των μοναδικών tokens που περιλαμβάνονται στο dataset εκπαίδευσης και αποθηκεύεται μαζί με το λεξιλόγιο, ώστε να διασφαλίζεται συμβατότητα μεταξύ εκπαίδευσης και εκτέλεσης του μοντέλου. Η χρήση της αναπαράστασης Bag-of-Words προσφέρει μια ισορροπημένη λύση μεταξύ απλότητας, ταχύτητας και ακρίβειας, καθιστώντας την ιδιαίτερα κατάλληλη για το συγκεκριμένο σενάριο τηλεφωνικού Voicebot. Η αριθμητική αυτή αναπαράσταση αποτελεί την είσοδο του νευρωνικού μοντέλου που περιγράφεται στην επόμενη ενότητα.

### 5.3.2 Μοντέλο PyTorch

Για την ταξινόμηση των προθέσεων (intent classification) του χρήστη επιλέχθηκε η χρήση ενός νευρωνικού δικτύου τύπου Multilayer Perceptron (MLP), υλοποιημένου με τη βιβλιοθήκη PyTorch. Το μοντέλο αυτό λαμβάνει ως είσοδο τη διανυσματική αναπαράσταση Bag-of-Words της πρότασης και επιστρέφει πιθανότητες για κάθε διαθέσιμη πρόθεση, επιτρέποντας την επιλογή της καταλληλότερης απόκρισης.

Η υλοποίηση του μοντέλου πραγματοποιείται στο αρχείο nlu_intent_pt.py και ορίζεται με την ακόλουθη δομή:

```
class BoWMLP(nn.Module):
    def __init__(self, vocab_size, num_classes):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(vocab_size, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, num_classes)
        )
```

Η αρχιτεκτονική του μοντέλου αποτελείται από ένα πλήρως συνδεδεμένο γραμμικό επίπεδο εισόδου, το οποίο μετατρέπει το διάνυσμα Bag-of-Words σε έναν ενδιάμεσο χώρο χαρακτηριστικών σταθερού μεγέθους. Στη συνέχεια, εφαρμόζεται συνάρτηση ενεργοποίησης ReLU (Rectified Linear Unit), η οποία εισάγει μη γραμμικότητα στο μοντέλο και επιτρέπει την εκμάθηση πιο σύνθετων συσχετίσεων μεταξύ των χαρακτηριστικών. Για την αποφυγή υπερπροσαρμογής (overfitting), ενσωματώνεται μηχανισμός Dropout με ποσοστό 0.3, ο οποίος απενεργοποιεί τυχαία ένα μέρος των νευρώνων κατά τη διάρκεια της εκπαίδευσης. Η τεχνική αυτή είναι ιδιαίτερα χρήσιμη σε περιπτώσεις όπου το dataset είναι περιορισμένο, όπως συχνά συμβαίνει σε εξειδικευμένα τηλεφωνικά σενάρια. Το τελικό επίπεδο εξόδου είναι επίσης γραμμικό και περιλαμβάνει αριθμό νευρώνων ίσο με το πλήθος των διαθέσιμων προθέσεων (intents). Η έξοδος του μοντέλου αντιστοιχεί σε λογότυπα (logits), τα οποία μετατρέπονται σε πιθανότητες μέσω της συνάρτησης softmax κατά το στάδιο του inference. Η πρόθεση με τη μεγαλύτερη πιθανότητα επιλέγεται ως η τελική εκτίμηση του συστήματος.

Η επιλογή ενός απλού MLP, αντί πιο σύνθετων μοντέλων όπως recurrent νευρωνικά δίκτυα ή αρχιτεκτονικές βασισμένες σε transformers, έγινε συνειδητά και βασίστηκε σε πρακτικά κριτήρια που σχετίζονται με το περιβάλλον λειτουργίας του Voicebot. Συγκεκριμένα, η προσέγγιση αυτή εξασφαλίζει:

- **χαμηλό χρόνο απόκρισης**, κάτι που είναι κρίσιμο για τη διατήρηση φυσικής ροής σε τηλεφωνική συνομιλία.
- **σταθερή λειτουργία σε περιβάλλον CPU-only**, χωρίς ανάγκη για επιτάχυνση μέσω GPU.
- **απλότητα στην κατανόηση, συντήρηση και επέκταση του μοντέλου**, καθιστώντας το σύστημα προσβάσιμο ακόμη και σε μη ειδικούς χρήστες.

Παρά την απλότητά του, το μοντέλο MLP αποδεικνύεται επαρκές για το συγκεκριμένο πρόβλημα, καθώς η ταξινόμηση προθέσεων σε τηλεφωνικά σενάρια βασίζεται κυρίως στην παρουσία συγκεκριμένων λέξεων-κλειδιών και όχι σε πολύπλοκες συντακτικές δομές. Επιπλέον, η απλή αρχιτεκτονική μειώνει τον κίνδυνο αστάθειας και διευκολύνει τη γρήγορη επανεκπαίδευση του μοντέλου με νέα δεδομένα.

Το επιλεγμένο μοντέλο PyTorch συνδυάζει αποδοτικότητα, ευελιξία και πρακτική αξιοπιστία, καθιστώντας το κατάλληλο για χρήση σε ένα πραγματικό τηλεφωνικό Voicebot που απαιτεί συνεχή και απρόσκοπτη λειτουργία. Στην επόμενη ενότητα παρουσιάζεται ο τρόπος με τον οποίο το εκπαιδευμένο μοντέλο χρησιμοποιείται για την εξαγωγή της τελικής πρόθεσης κατά τη διάρκεια της συνομιλίας.

### 5.3.3 Πρόβλεψη Πρόθεσης (Intent Prediction)

Κατά τη φάση της εκτέλεσης (inference), το εκπαιδευμένο NLU μοντέλο χρησιμοποιείται για την πρόβλεψη της πρόθεσης του χρήστη σε πραγματικό χρόνο. Η διαδικασία αυτή εκτελείται κάθε φορά που το Voicebot λαμβάνει νέα είσοδο κειμένου από το υποσύστημα αναγνώρισης ομιλίας και αποτελεί κρίσιμο στάδιο για τη λήψη αποφάσεων και τη συνέχεια της συνομιλίας. Αρχικά, το κείμενο που παράγεται από το STT έχει ήδη μετατραπεί σε αριθμητική αναπαράσταση μέσω της διαδικασίας Bag-of-Words. Το διάνυσμα εισόδου (vec) που προκύπτει τροφοδοτείται στο νευρωνικό μοντέλο, το οποίο βρίσκεται σε κατάσταση αξιολόγησης (evaluation mode). Για λόγους αποδοτικότητας και σταθερότητας, η πρόβλεψη πραγματοποιείται χωρίς υπολογισμό βαθμίδων (gradients), καθώς δεν απαιτείται εκπαίδευση κατά τη φάση της εκτέλεσης:

```
with torch.no_grad():
    logits = model(vec)
    probs = softmax(logits)
    intent = labels[probs.argmax()]
```

Το μοντέλο επιστρέφει αρχικά ένα διάνυσμα από λογότυπα (logits), το οποίο περιέχει μη κανονικοποιημένες τιμές για κάθε διαθέσιμη πρόθεση. Στη συνέχεια, εφαρμόζεται η συνάρτηση softmax, η οποία μετατρέπει τις τιμές αυτές σε πιθανότητες που αθροίζονται σε 1. Κάθε πιθανότητα εκφράζει τον βαθμό εμπιστοσύνης του μοντέλου ότι η συγκεκριμένη πρόταση αντιστοιχεί σε ένα συγκεκριμένο intent. Η τελική πρόθεση επιλέγεται ως εκείνη με τη μέγιστη πιθανότητα (argmax) και αντιστοιχίζεται στο αντίστοιχο λεκτικό label μέσω του πίνακα labels. Το αποτέλεσμα αυτό αποτελεί τη βασική έξοδο του υποσυστήματος NLU και επιστρέφεται άμεσα στο AGI script.

Η αναγνωρισμένη πρόθεση δεν λειτουργεί αυτόνομα αλλά εντάσσεται στο ευρύτερο πλαίσιο του διαλόγου. Το AGI script αξιοποιεί το intent σε συνδυασμό με την τρέχουσα κατάσταση της συνομιλίας (state) και το αποθηκευμένο context, ώστε να αποφασίσει:

- αν θα ξεκινήσει νέα διαδικασία (π.χ. νέα παραγγελία).
- αν θα συνεχίσει μια ήδη ενεργή ροή διαλόγου.
- αν θα ζητήσει διευκρινιστικές πληροφορίες.
- ή αν θα τερματίσει τη συνομιλία με κατάλληλο μήνυμα.

Η επιλογή της πρόθεσης με βάση τη μέγιστη πιθανότητα προσφέρει απλότητα και ταχύτητα, χαρακτηριστικά απαραίτητα για τηλεφωνικές εφαρμογές πραγματικού χρόνου. Παρότι δεν εφαρμόζεται κατώφλι εμπιστοσύνης (confidence threshold) στην υλοποίηση, η αρχιτεκτονική του συστήματος επιτρέπει την εύκολη ενσωμάτωση τέτοιου μηχανισμού στο μέλλον, ώστε να αντιμετωπίζονται περιπτώσεις αμφίβολης πρόβλεψης με επαναληπτικές ή διευκρινιστικές ερωτήσεις.

Το στάδιο της πρόβλεψης πρόθεσης ολοκληρώνει τη λειτουργία του υποσυστήματος NLU, μετατρέποντας το αριθμητικό αποτέλεσμα του νευρωνικού μοντέλου σε μια σαφή και αξιοποιήσιμη απόφαση. Η απόφαση αυτή αποτελεί τον συνδετικό κρίκο μεταξύ της κατανόησης φυσικής γλώσσας και της διαχείρισης διαλόγου, επιτρέποντας στο Voicebot να αντιδρά με τρόπο στοχευμένο, συνεπή και λειτουργικό σε πραγματικό τηλεφωνικό περιβάλλον.

## 5.4 Εκπαίδευση Μοντέλου (train_intents.py)

Η εκπαίδευση του μοντέλου κατανόησης φυσικής γλώσσας αποτελεί κρίσιμο στάδιο για τη συνολική απόδοση του Voicebot, καθώς καθορίζει την ικανότητά του να αναγνωρίζει σωστά τις προθέσεις των χρηστών και να ενεργοποιεί τις αντίστοιχες ροές διαλόγου. Η διαδικασία εκπαίδευσης υλοποιείται μέσω του script train_intents.py, το οποίο έχει σχεδιαστεί ώστε να λειτουργεί αυτόνομα, αλλά και να μπορεί να ενσωματωθεί πλήρως στο web περιβάλλον διαχείρισης. Το συγκεκριμένο script αναλαμβάνει ολόκληρη τη διαδικασία εκπαίδευσης του NLU μοντέλου, ξεκινώντας από τη φόρτωση των δεδομένων και καταλήγοντας στην αποθήκευση όλων των απαραίτητων αρχείων που απαιτούνται για τη λειτουργία του συστήματος σε περιβάλλον παραγωγής. Η χρήση ενός ξεχωριστού training script διασφαλίζει τη σαφή διάκριση μεταξύ της φάσης εκπαίδευσης (offline διαδικασία) και της φάσης εκτέλεσης (online inference), κάτι που είναι ιδιαίτερα σημαντικό για τηλεφωνικές εφαρμογές πραγματικού χρόνου.

Κατά την εκπαίδευση, το μοντέλο τροφοδοτείται με σύνολα δεδομένων εκπαίδευσης και επικύρωσης (training και validation sets), τα οποία προέρχονται από αρχεία CSV. Η χρήση ξεχωριστού validation set επιτρέπει την αντικειμενική αξιολόγηση της γενίκευσης του μοντέλου και τη λήψη αποφάσεων σχετικά με τη διάρκεια της εκπαίδευσης.

Στο πλαίσιο αυτό, το train_intents.py:

- φορτώνει τα δεδομένα εκπαίδευσης και επικύρωσης.
- δημιουργεί το λεξιλόγιο (Bag-of-Words) και τους πίνακες ετικετών (intent labels).
- εκπαιδεύει το νευρωνικό μοντέλο για προκαθορισμένο αριθμό epochs.
- παρακολουθεί μετρικές απόδοσης (loss, accuracy, F1-score).
- και εφαρμόζει μηχανισμό early stopping για αποφυγή υπερπροσαρμογής.

Η χρήση του early stopping αποτελεί σημαντικό στοιχείο της εκπαίδευσης, καθώς επιτρέπει στο σύστημα να τερματίζει τη διαδικασία όταν η απόδοση στο validation set παύει να βελτιώνεται. Με τον τρόπο αυτό αποφεύγεται η υπερπροσαρμογή του μοντέλου στα δεδομένα εκπαίδευσης, ενώ παράλληλα εξοικονομούνται υπολογιστικοί πόροι και χρόνος, κάτι ιδιαίτερα χρήσιμο σε περιβάλλοντα με περιορισμένες δυνατότητες (CPU-only server).

Χαρακτηριστικό παράδειγμα εξόδου του training script φαίνεται στο ακόλουθο απόσπασμα:

```
[003] loss=1.2867 | train Acc=0.875 F1=0.700 | val Acc=0.667 F1=0.556
Early stopping at epoch 7
Saved model to /var/lib/voicebot/models/nlu_intent.pt
```

Το παραπάνω log καταδεικνύει τη σταδιακή βελτίωση της απόδοσης κατά την εκπαίδευση, καθώς και την ενεργοποίηση του μηχανισμού early stopping, όταν η επίδοση στο validation set δεν παρουσιάζει περαιτέρω βελτίωση.

### 5.4.1 Αποθήκευση και Διαχείριση Μοντέλου

Μετά την ολοκλήρωση της εκπαίδευσης, το σύστημα αποθηκεύει όλα τα απαραίτητα αρχεία που απαιτούνται για την εκτέλεση του NLU μοντέλου σε πραγματικό χρόνο. Η αποθήκευση δεν περιορίζεται μόνο στα βάρη του νευρωνικού δικτύου, αλλά περιλαμβάνει και κρίσιμα μεταδεδομένα που εξασφαλίζουν τη συνέπεια μεταξύ εκπαίδευσης και εκτέλεσης.

Συγκεκριμένα, αποθηκεύονται τα εξής αρχεία:

- **nlu_intent.pt**
  Περιέχει τα βάρη (state_dict) του εκπαιδευμένου μοντέλου PyTorch. Το αρχείο αυτό αποτελεί τον πυρήνα του NLU και φορτώνεται κατά την εκκίνηση του Voicebot.
- **nlu_vocab.json**
  Περιλαμβάνει το λεξιλόγιο που χρησιμοποιήθηκε στη φάση της εκπαίδευσης, δηλαδή την αντιστοίχιση κάθε token σε συγκεκριμένο δείκτη του διανύσματος εισόδου. Η ύπαρξη του αρχείου αυτού διασφαλίζει ότι το κείμενο εισόδου θα μετατρέπεται με τον ίδιο ακριβώς τρόπο τόσο κατά την εκπαίδευση όσο και κατά την εκτέλεση.
- **nlu_labels.json**
  Περιέχει την αντιστοίχιση μεταξύ των αριθμητικών κλάσεων εξόδου του μοντέλου και των λεκτικών ετικετών προθέσεων (intents). Με τον τρόπο αυτό, το αποτέλεσμα του μοντέλου μετατρέπεται σε κατανοητή και αξιοποιήσιμη πληροφορία για το AGI script.
- **nlu_metrics.json**
  Αποθηκεύει στατιστικά εκπαίδευσης και μετρικές απόδοσης (όπως loss, accuracy και F1-score), επιτρέποντας την αξιολόγηση και σύγκριση διαφορετικών εκδόσεων του μοντέλου στο χρόνο.

Τα παραπάνω αρχεία φορτώνονται δυναμικά από το AGI script κατά την εκκίνηση του Voicebot, ώστε το σύστημα να είναι άμεσα έτοιμο για χρήση χωρίς επιπλέον καθυστερήσεις. Η προσέγγιση αυτή επιτρέπει γρήγορη και αποδοτική λειτουργία κατά τη φάση της πρόβλεψης (inference), ενώ παράλληλα καθιστά τη διαδικασία αναβάθμισης του μοντέλου απλή: αρκεί η αντικατάσταση των αποθηκευμένων αρχείων με νεότερη έκδοση.

Η οργανωμένη αποθήκευση και διαχείριση του μοντέλου και των συνοδευτικών μεταδεδομένων ενισχύει τη σταθερότητα, την επεκτασιμότητα και τη μακροχρόνια συντήρηση του Voicebot, καθιστώντας το κατάλληλο για συνεχή χρήση σε πραγματικό τηλεφωνικό περιβάλλον.

## 5.5 Πρότυπα Απαντήσεων (response_templates.py)

Η παραγωγή λεκτικών απαντήσεων αποτελεί βασικό στοιχείο της συνολικής εμπειρίας χρήσης του Voicebot, καθώς καθορίζει τον τρόπο με τον οποίο το σύστημα επικοινωνεί, καθοδηγεί και εξυπηρετεί του χρήστη. Για τον λόγο αυτό, η υλοποίηση έχει μια οργανωμένη και επεκτάσιμη προσέγγιση διαχείρισης απαντήσεων, μέσω ξεχωριστού αρχείου προτύπων (response_templates.py). Ο σχεδιαστικός αυτός διαχωρισμός επιλέχθηκε συνειδητά, ώστε η λογική του διαλόγου (δηλαδή ***πότε*** και ***γιατί*** απαντά το σύστημα) να παραμένει ανεξάρτητη από το λεκτικό περιεχόμενο (δηλαδή ***τι*** ακριβώς λέει).

Με αυτόν τον τρόπο, ο πυρήνας της εφαρμογής (AGI script, NLU και state machine) δεν «γεμίζει» με στατικές φράσεις και μηνύματα, αλλά αναφέρεται σε αυτά με δομημένο τρόπο, μέσω συμβολικών κλειδιών (intents ή/και καταστάσεων). Το αποτέλεσμα είναι ένα σύστημα σαφώς οργανωμένο, πιο εύκολο στη συντήρηση και πολύ πιο πρακτικό στην παραμετροποίηση, ιδιαίτερα σε πραγματικά επιχειρησιακά περιβάλλοντα, όπου οι φράσεις και το ύφος επικοινωνίας συχνά αλλάζουν ανάλογα με το προϊόν, την εταιρική πολιτική ή το κοινό στόχο.

### 5.5.1 Αντιστοίχιση intents με λεκτικές αποκρίσεις

Στο αρχείο response_templates.py ορίζονται αντιστοιχίσεις ανάμεσα στις αναγνωρισμένες προθέσεις (intents) και σε προκαθορισμένες φράσεις απόκρισης. Κάθε πρόθεση συνδέεται με ένα template, το οποίο μπορεί να χρησιμοποιηθεί άμεσα ως μήνυμα προς τον χρήστη, ή να λειτουργήσει ως βάση για πιο δυναμικές απαντήσεις. Ένα απλό παράδειγμα αντιστοίχισης φαίνεται παρακάτω:

```
TEMPLATES = {
   "greeting": "Καλησπέρα! Πώς μπορώ να βοηθήσω;",
   "hours": "Το ωράριό μας είναι 09:00–17:00.",
   "new_order": "Τέλεια! Να ξεκινήσουμε την παραγγελία σας;",
   "order_status": "Μπορώ να κοιτάξω την κατάσταση της παραγγελίας σας."
}
```

Η παραπάνω δομή επιτρέπει την άμεση ανάκτηση απαντήσεων με βάση το intent, διασφαλίζοντας συνεπή και προβλέψιμη συμπεριφορά. Παράλληλα, μπορεί να επεκταθεί ώστε κάθε intent να αντιστοιχεί όχι σε μία μόνο απάντηση, αλλά σε λίστα από παραλλαγές, επιτρέποντας πιο φυσική συνομιλία και αποφυγή μονότονων, επαναλαμβανόμενων απαντήσεων.

### 5.5.2 Πλεονεκτήματα του διαχωρισμού περιεχομένου–λογικής

Η συγκεκριμένη προσέγγιση προσφέρει σημαντικά πλεονεκτήματα σε επίπεδο σχεδίασης και λειτουργίας:

- **Σαφής διαχωρισμός επιχειρησιακής λογικής και λεκτικού περιεχομένου**
  Το AGI script και το state machine παραμένουν επικεντρωμένα στη ροή του διαλόγου, χωρίς να περιέχουν "hard-coded" φράσεις.
- **Εύκολη τροποποίηση απαντήσεων χωρίς αλλαγές στον πυρήνα του κώδικα**
  Μια αλλαγή σε ωράριο, ύφος ή διατύπωση γίνεται στο response_templates.py χωρίς να απαιτείται παρέμβαση στον μηχανισμό NLU ή στην τηλεφωνική λογική.
- **Δυνατότητα επέκτασης με περισσότερες παραλλαγές ανά πρόθεση**
  Μπορούν να προστεθούν πολλαπλές φράσεις ανά intent (π.χ. 3–5 εναλλακτικές), ώστε να επιλέγεται τυχαία ή βάσει πολιτικής η πιο κατάλληλη.
- **Προσαρμογή ανάλογα με την κατάσταση του διαλόγου (state-aware responses)**
  Η ίδια πρόθεση μπορεί να απαιτεί διαφορετική διατύπωση ανάλογα με το βήμα (π.χ. αρχή παραγγελίας ή επιβεβαίωση παραγγελίας).

Στην πράξη, αυτό σημαίνει ότι το NLU μπορεί να εντοπίζει την πρόθεση new_order, αλλά ο τρόπος που θα απαντήσει το σύστημα μπορεί να εξαρτάται από το αν η συνομιλία βρίσκεται σε φάση "έναρξης", "συλλογής στοιχείων" ή "επιβεβαίωσης".

### 5.5.3 Σταθερές και δυναμικές αποκρίσεις

Το σύστημα υποστηρίζει τόσο **σταθερές** όσο και **δυναμικές** απαντήσεις:

- **Σταθερές απαντήσεις**:
  Χρησιμοποιούνται όταν το περιεχόμενο δεν εξαρτάται από δεδομένα χρήστη, όπως:
  - ωράριο λειτουργίας
  - στοιχεία επικοινωνίας
  - γενικές πληροφορίες
- **Δυναμικές απαντήσεις**:
  Χρησιμοποιούνται όταν η απάντηση πρέπει να ενσωματώσει πληροφορίες που συλλέχθηκαν κατά τη διάρκεια του διαλόγου, όπως:
  - όνομα πελάτη
  - προϊόν
  - ποσότητα
  - κατάσταση παραγγελίας

Για παράδειγμα, μετά τη συλλογή των στοιχείων, το Voicebot μπορεί να παράγει μήνυμα επιβεβαίωσης τύπου:
«Κύριε Γιώργο, επιβεβαιώνετε παραγγελία 2 τεμαχίων Μουρουνέλαιο;»

Η παραπάνω απάντηση βασίζεται σε template, αλλά εμπλουτίζεται με δεδομένα από το context που διατηρεί το state machine.

### 5.5.4 Σύνδεση με AGI και state machine

Η επιλογή του κατάλληλου template δεν γίνεται «απομονωμένα» μόνο από το intent, αλλά συνδυάζεται με τον μηχανισμό διαλόγου. Το AGI script ενοποιεί τις πληροφορίες, καθώς:

- λαμβάνει το intent από το NLU.
- γνωρίζει το τρέχον state.
- διαθέτει το context (π.χ. order δεδομένα).
- επιλέγει την κατάλληλη λεκτική απόκριση.

Με αυτό τον τρόπο διασφαλίζεται ότι οι απαντήσεις είναι όχι μόνο σωστές ως προς το αίτημα του χρήστη αλλά και κατάλληλες ως προς το στάδιο της συνομιλίας.

### 5.5.5 Συνολική συμβολή της προσέγγισης

Συνολικά, η χρήση του response_templates.py ως ξεχωριστό επίπεδο διαχείρισης λεκτικού περιεχομένου ενισχύει:

- τη συντήρηση.
- την επεκτασιμότητα.
- την παραμετροποίηση.
- και τη δυνατότητα προσαρμογής του Voicebot σε διαφορετικά σενάρια ή επιχειρήσεις.

Αποτελεί, ουσιαστικά, ένα “content layer” πάνω από τη βασική λογική του συστήματος, επιτρέποντας στο Voicebot να λειτουργεί ως πλατφόρμα που μπορεί εύκολα να επαναχρησιμοποιηθεί και να προσαρμοστεί.

## 5.6 Django Web Console – NLU Manager

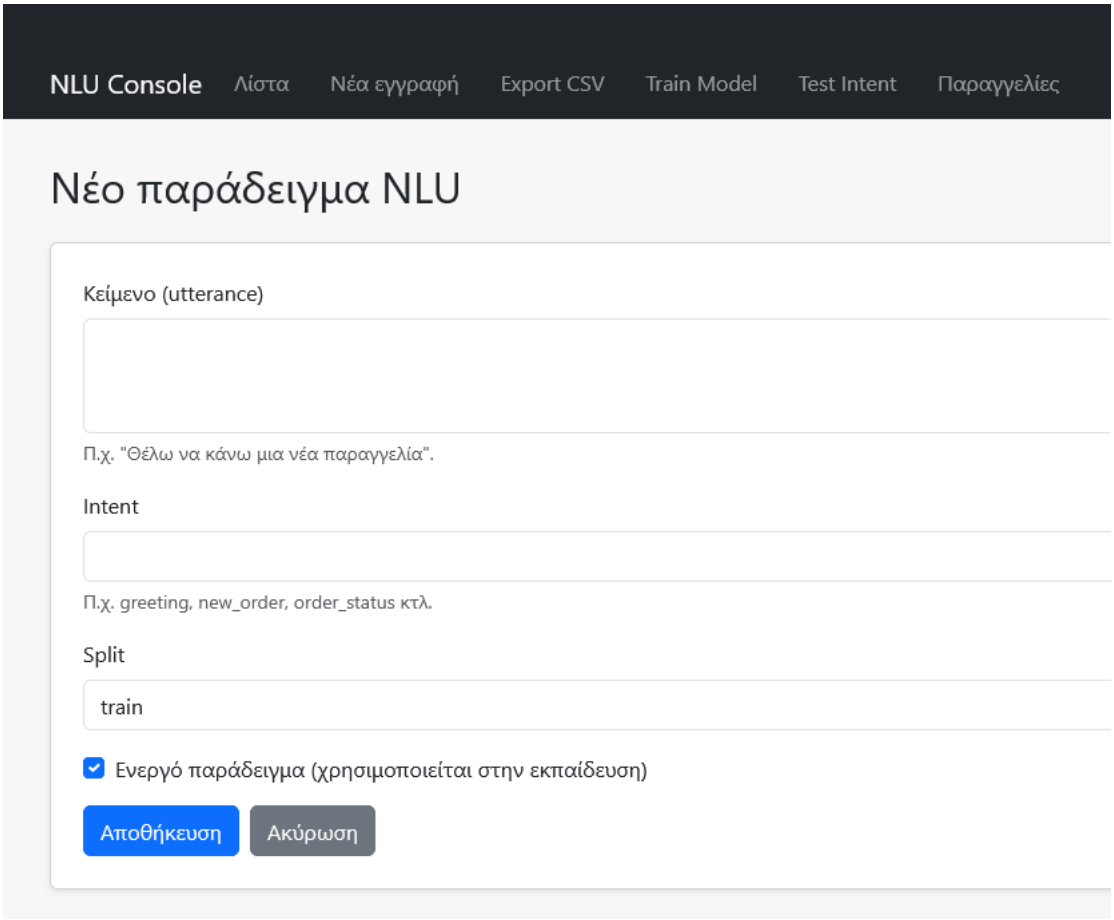
Για τη διευκόλυνση της διαχείρισης, της παραμετροποίησης και της συνεχούς βελτίωσης του Voicebot, αναπτύχθηκε ειδικό web περιβάλλον διαχείρισης με χρήση του Django. Το περιβάλλον αυτό λειτουργεί ως κεντρική κονσόλα ελέγχου του συστήματος και στοχεύει στη γεφύρωση του κενού μεταξύ της τεχνικής υλοποίησης και της πρακτικής χρήσης. Κύριος στόχος του Django Web Console είναι να επιτρέψει σε διαχειριστές του τηλεφωνικού συστήματος να επεμβαίνουν στη συμπεριφορά του Voicebot χωρίς να απαιτούνται εξειδικευμένες γνώσεις προγραμματισμού ή μηχανικής μάθησης. Με τον τρόπο αυτό, το Voicebot μετατρέπεται από ένα στατικό τεχνολογικό προϊόν σε ένα δυναμικό σύστημα, το οποίο μπορεί να προσαρμόζεται συνεχώς στις ανάγκες και στη γλωσσική συμπεριφορά των χρηστών.

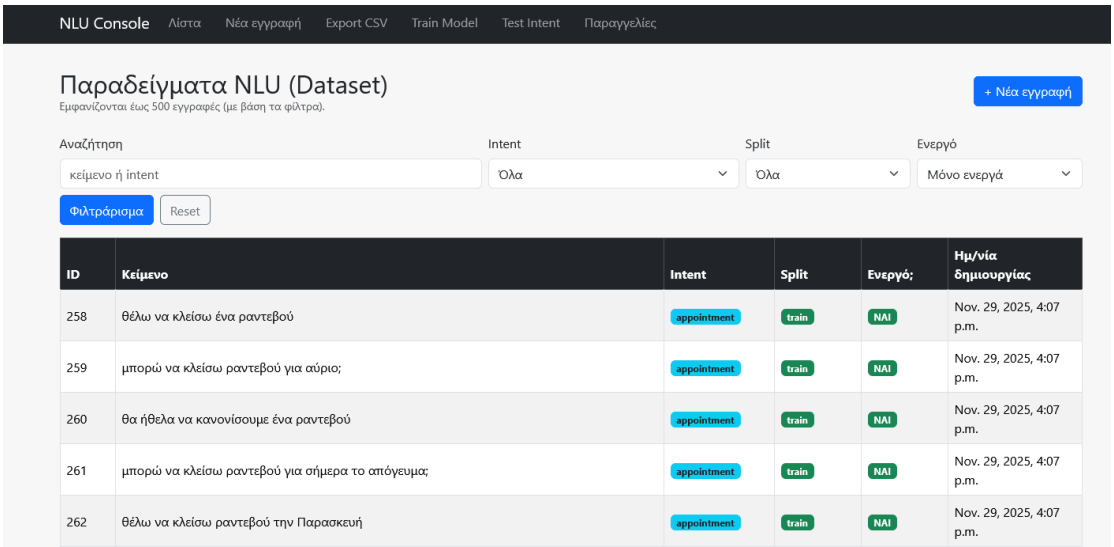
### 5.6.1 Διαχείριση δεδομένων εκπαίδευσης (CRUD)

Το web περιβάλλον παρέχει πλήρη λειτουργικότητα διαχείρισης των δεδομένων εκπαίδευσης του NLU μοντέλου μέσω CRUD (Create, Read, Update, Delete) μηχανισμών. Ο διαχειριστής μπορεί:

- να προσθέτει νέες προτάσεις εκπαίδευσης,
- να τροποποιεί υπάρχουσες εγγραφές,
- και να διαγράφει προτάσεις που κρίνονται μη αντιπροσωπευτικές ή λανθασμένες.

Κάθε πρόταση αντιστοιχίζεται σε συγκεκριμένη πρόθεση (intent), επιτρέποντας τον σταδιακό εμπλουτισμό του dataset με πραγματικά παραδείγματα χρήσης. Η δυνατότητα αυτή είναι ιδιαίτερα σημαντική, καθώς οι χρήστες συχνά εκφράζουν το ίδιο αίτημα με διαφορετικό τρόπο, κάτι που απαιτεί συνεχή επικαιροποίηση των δεδομένων εκπαίδευσης.



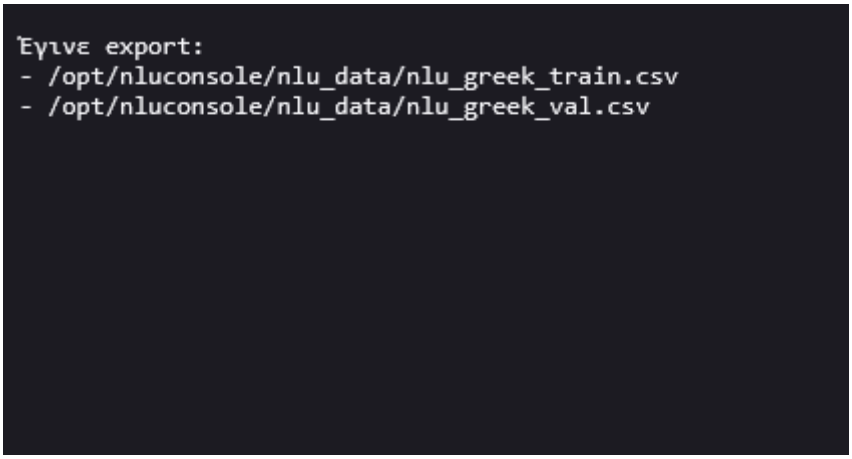



### 5.6.2 Εισαγωγή και εξαγωγή δεδομένων (Import / Export CSV)

Πέρα από τη μεμονωμένη διαχείριση εγγραφών, το σύστημα υποστηρίζει εισαγωγή και εξαγωγή δεδομένων εκπαίδευσης μέσω αρχείων CSV. Η λειτουργία αυτή εξυπηρετεί:

- τη μαζική αρχική δημιουργία dataset
- τη μεταφορά δεδομένων μεταξύ διαφορετικών περιβαλλόντων (π.χ. δοκιμαστικό → παραγωγικό),
- καθώς και τη δημιουργία αντιγράφων ασφαλείας.

Η επιλογή του CSV ως ενδιάμεσης μορφής διασφαλίζει απλότητα, διαφάνεια και συμβατότητα με πληθώρα εργαλείων επεξεργασίας δεδομένων.

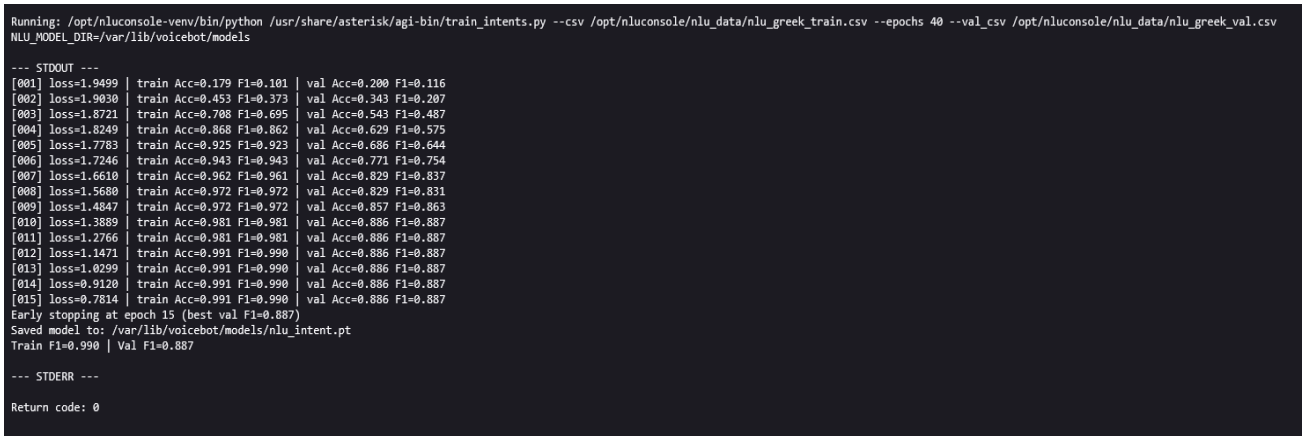



### 5.6.3 Επανεκπαίδευση μοντέλου μέσω Web Interface

Ένα από τα σημαντικότερα χαρακτηριστικά του Django Web Console είναι η δυνατότητα εκκίνησης της επανεκπαίδευσης του NLU μοντέλου απευθείας μέσω της διεπαφής. Με το πάτημα ενός κουμπιού, ο διαχειριστής μπορεί να ενεργοποιήσει το training pipeline, το οποίο:

- φορτώνει τα ενημερωμένα δεδομένα εκπαίδευσης.
- επανεκπαιδεύει το μοντέλο.
- αποθηκεύει τα νέα αρχεία μοντέλου και μεταδεδομένων.

Η διαδικασία αυτή έχει σχεδιαστεί ώστε να είναι συμβατή με το AGI περιβάλλον και να μην



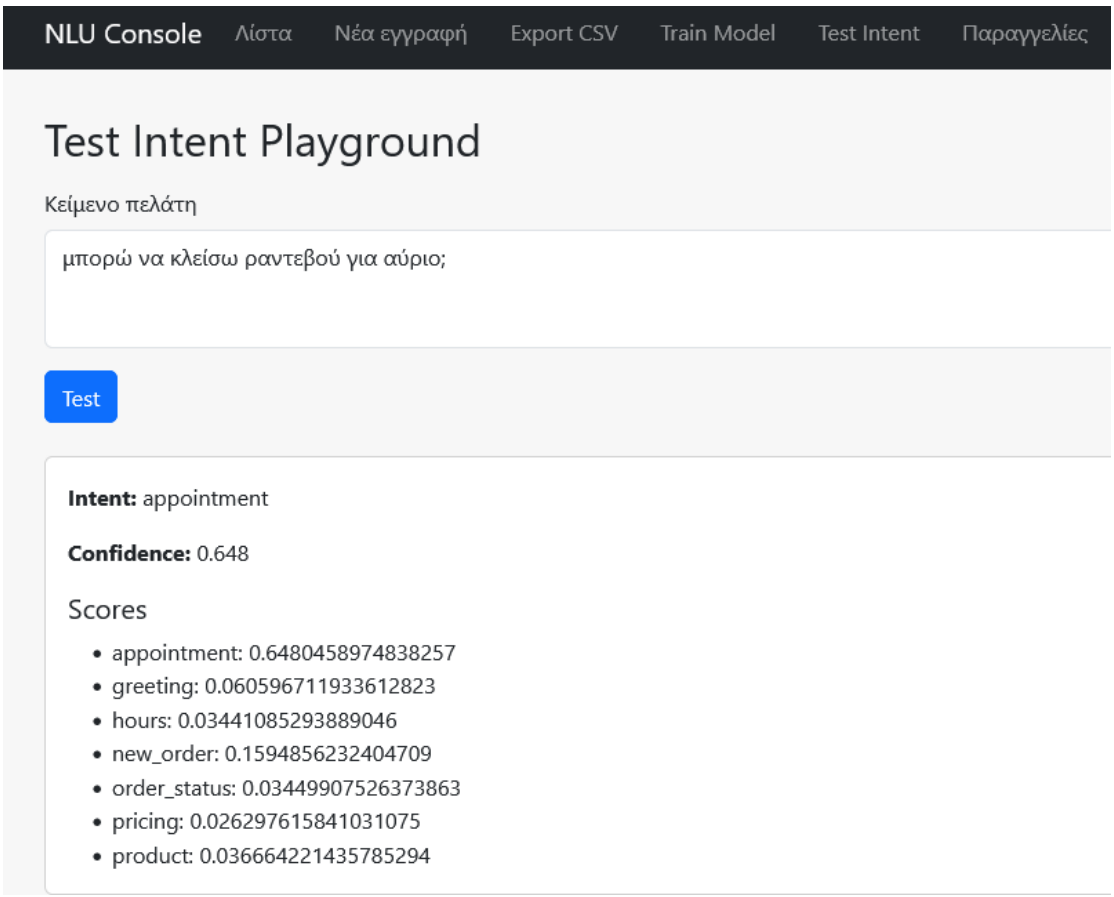
διακόπτει τη λειτουργία του τηλεφωνικού συστήματος. Το Voicebot με αυτό το τρόπο μπορεί να συνεχίσει να εξυπηρετεί κλήσεις, ενώ το νέο μοντέλο τίθεται σε χρήση στην επόμενη εκκίνηση του συστήματος.

### 5.6.4 Intent Playground – Δοκιμή Προτάσεων

Για την άμεση αξιολόγηση της απόδοσης του NLU μοντέλου, το web περιβάλλον περιλαμβάνει ειδική ενότητα δοκιμών (Intent Playground). Ο διαχειριστής μπορεί να εισάγει ελεύθερο κείμενο και να παρατηρήσει:

- την πρόθεση που αναγνωρίζει το μοντέλο.
- καθώς και την αντίστοιχη πιθανότητα ή βαθμό εμπιστοσύνης.

Η λειτουργία αυτή είναι ιδιαίτερα χρήσιμη μετά από επανεκπαίδευση, καθώς επιτρέπει γρήγορο έλεγχο της συμπεριφοράς του μοντέλου σε νέες ή οριακές διατυπώσεις, χωρίς να απαιτείται πραγματική τηλεφωνική κλήση.

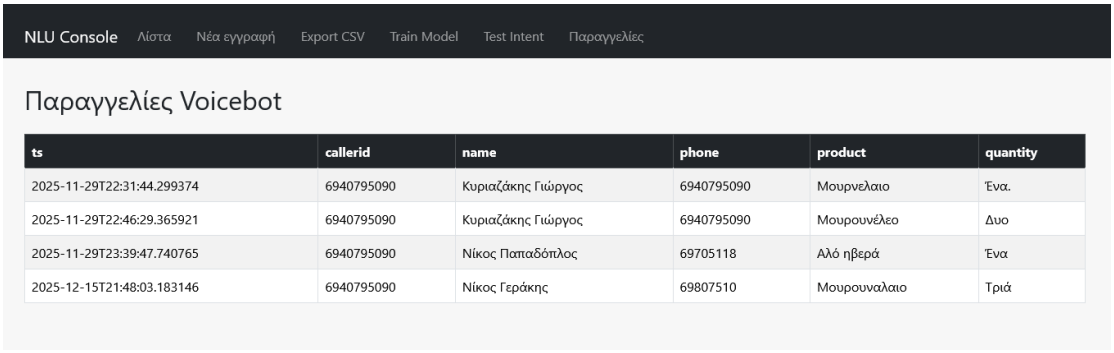


### 5.6.5 Προβολή καταγεγραμμένων παραγγελιών (Voicebot Orders Viewer)

Το Django Web Console περιλαμβάνει επίσης ενότητα προβολής των παραγγελιών που καταγράφονται από το Voicebot. Οι παραγγελίες εμφανίζονται σε δομημένο πίνακα, επιτρέποντας:

- γρήγορη επισκόπηση της λειτουργίας του συστήματος.
- έλεγχο της ορθότητας των συλλεγόμενων δεδομένων.
- βασική αξιοποίηση των αποτελεσμάτων.

Η δυνατότητα αυτή ενισχύει τη διαφάνεια και καθιστά το Voicebot άμεσα χρήσιμο σε πραγματικό επιχειρησιακό περιβάλλον.




### 5.6.6 Σχεδίαση interface

Για τη σχεδίαση του interface χρησιμοποιήθηκε το Bootstrap 5, το οποίο προσφέρει responsive σχεδίαση και καθαρή, λειτουργική εμφάνιση. Το interface είναι κατάλληλο τόσο για χρήση σε desktop συστήματα όσο και σε φορητές συσκευές, διευκολύνοντας τη διαχείριση του Voicebot από διαφορετικά περιβάλλοντα εργασίας.

5.6.7 Συνολική συμβολή του Web Console

Η ύπαρξη του Django Web Console μετατρέπει το Voicebot από ένα τεχνικά στατικό σύστημα σε μια δυναμικά εξελισσόμενη πλατφόρμα. Μέσω του web εργαλείου, η βελτίωση της απόδοσης του NLU γίνεται συνεχής διαδικασία, βασισμένη σε πραγματικά δεδομένα χρήσης και με ελάχιστη τεχνική παρέμβαση.

Συνολικά, το web περιβάλλον διαχείρισης ενισχύει την πρακτική αξία, τη βιωσιμότητα και την επεκτασιμότητα του Voicebot, καθιστώντας το κατάλληλο για μακροχρόνια χρήση σε πραγματικά τηλεφωνικά κέντρα.

## 5.7 Καταχώριση Παραγγελιών (order_log.csv)

Η καταχώριση των παραγγελιών αποτελεί το τελικό και ουσιαστικό στάδιο της ροής διαλόγου στις περιπτώσεις όπου ο χρήστης ολοκληρώνει επιτυχώς τη διαδικασία παραγγελιοληψίας. Αφού συλλεχθούν όλα τα απαραίτητα στοιχεία μέσω του μηχανισμού state machine και πραγματοποιηθεί η τελική επιβεβαίωση από τον χρήστη, το σύστημα

προχωρά στην αποθήκευση των δεδομένων της παραγγελίας. Η διαδικασία καταγραφής ενεργοποιείται αποκλειστικά όταν η ροή του διαλόγου έχει ολοκληρωθεί χωρίς σφάλματα και ο χρήστης έχει επιβεβαιώσει ρητά την πρόθεσή του να προχωρήσει στην καταχώριση. Με τον τρόπο αυτό αποφεύγονται ελλιπείς ή λανθασμένες εγγραφές και διασφαλίζεται η ποιότητα των αποθηκευμένων δεδομένων.

### 5.7.1 Δομή και περιεχόμενο του αρχείου καταγραφής

Η αποθήκευση των παραγγελιών πραγματοποιείται σε αρχείο CSV με όνομα order_log.csv. Κάθε γραμμή του αρχείου αντιστοιχεί σε μία ολοκληρωμένη παραγγελία και περιλαμβάνει τα ακόλουθα βασικά πεδία:

- **Χρονική σήμανση (timestamp):** ημερομηνία και ώρα ολοκλήρωσης της παραγγελίας,
- **Αναγνωριστικό καλούντος (caller ID):** αριθμός τηλεφώνου από τον οποίο πραγματοποιήθηκε η κλήση,
- **Ονοματεπώνυμο πελάτη:** όπως δηλώθηκε κατά τη διάρκεια της συνομιλίας,
- **Αριθμός τηλεφώνου επικοινωνίας:** για πιθανή επιβεβαίωση ή μεταγενέστερη επικοινωνία,
- **Προϊόν:** είδος που επιλέχθηκε από τον χρήστη,
- **Ποσότητα:** αριθμός τεμαχίων του προϊόντος.

Ενδεικτική εγγραφή στο αρχείο:

```
ts, callerid, name, phone, product, quantity
2025-11-29T22:31:44.29, 6940795090, Κυριαζάκης Γιώργος, 6940795090, Μουρουνέλαιο, 1
```

Η συγκεκριμένη δομή επιτρέπει την εύκολη κατανόηση του περιεχομένου ακόμα και από μη τεχνικούς χρήστες, ενώ παράλληλα διατηρεί πλήρη πληροφορία για κάθε συναλλαγή.

### 5.7.2 Επιλογή μορφής CSV

Η επιλογή της μορφής CSV έγινε συνειδητά, λαμβάνοντας υπόψη τις ανάγκες της παρούσας υλοποίησης και τη φιλοσοφία απλότητας και διαφάνειας που διέπει το σύστημα. Τα κύρια πλεονεκτήματα της συγκεκριμένης επιλογής είναι:

- **Απλότητα υλοποίησης:** δεν απαιτείται επιπλέον υποδομή βάσης δεδομένων.
- **Διαφάνεια δεδομένων:** τα αρχεία μπορούν να ανοιχθούν και να ελεγχθούν άμεσα.
- **Ευκολία εξαγωγής και ανάλυσης:** δυνατότητα επεξεργασίας σε λογισμικά όπως Excel ή εργαλεία data analysis.
- **Συμβατότητα με εξωτερικά συστήματα:** εύκολη εισαγωγή σε CRM ή ERP συστήματα.

Παράλληλα, η αρχιτεκτονική του Voicebot έχει σχεδιαστεί έτσι ώστε η μετάβαση από αρχείο CSV σε πλήρη βάση δεδομένων (π.χ. MySQL, PostgreSQL) να μπορεί να πραγματοποιηθεί

στο μέλλον χωρίς αλλαγές στη ροή του διαλόγου ή στο AGI script, αλλά μόνο με αντικατάσταση του μηχανισμού αποθήκευσης.

### 5.7.3 Ενσωμάτωση με το Web Περιβάλλον Διαχείρισης

Οι καταγεγραμμένες παραγγελίες δεν παραμένουν απομονωμένες σε επίπεδο αρχείου, αλλά ενσωματώνονται λειτουργικά στο Django Web Console. Μέσω της διεπαφής διαχείρισης, ο διαχειριστής μπορεί να:

- προβάλλει όλες τις καταχωρίσεις σε μορφή πίνακα.
- ελέγχει την πληρότητα και την ορθότητα των δεδομένων.
- επιβεβαιώνει τη σωστή λειτουργία του Voicebot σε πραγματικές συνθήκες.

Η δυνατότητα αυτή προσφέρει άμεση εποπτεία της λειτουργίας του συστήματος και λειτουργεί ως μηχανισμός ελέγχου ποιότητας τόσο για το NLU όσο και για τον μηχανισμό διαλόγου.

### 5.7.4 Πρακτική σημασία της καταγραφής

Η καταχώριση των παραγγελιών αναδεικνύει τη μετάβαση του Voicebot από πειραματική εφαρμογή σε εργαλείο πραγματικής επιχειρησιακής αξίας. Το σύστημα δεν περιορίζεται στην απλή συνομιλία, αλλά ολοκληρώνει ουσιαστικές διεργασίες, παράγοντας αξιοποιήσιμα δεδομένα. Ο μηχανισμός καταγραφής παραγγελιών ολοκληρώνει λειτουργικά την αρχιτεκτονική του Voicebot και αποδεικνύει ότι το προτεινόμενο σύστημα μπορεί να χρησιμοποιηθεί ως αυτόνομο εργαλείο τηλεφωνικής εξυπηρέτησης και παραγγελιοληψίας σε πραγματικά παραγωγικά περιβάλλοντα.

## 5.8 Διαχείριση Αρχείων Ήχου

Κατά τη διάρκεια της λειτουργίας του Voicebot παράγεται και διαχειρίζεται ένας σημαντικός αριθμός αρχείων ήχου, τα οποία αποτελούν κρίσιμο στοιχείο της συνολικής λειτουργίας του συστήματος. Τα αρχεία αυτά περιλαμβάνουν τόσο την καταγραφή της ομιλίας του χρήστη όσο και τα συνθετικά φωνητικά μηνύματα που παράγονται από το Voicebot σε κάθε στάδιο του διαλόγου.

Όλα τα αρχεία ήχου αποθηκεύονται σε ειδικά ορισμένο κατάλογο του Asterisk:

```
/var/lib/asterisk/sounds/voicebot/
```

Η επιλογή του συγκεκριμένου καταλόγου ακολουθεί τη δομή του Asterisk και επιτρέπει την άμεση αναπαραγωγή των αρχείων μέσω AGI εντολών, χωρίς ανάγκη αντιγραφής ή επιπλέον μετατροπών.

### 5.8.1 Τύποι αρχείων ήχου

Στο πλαίσιο μιας τηλεφωνικής συνεδρίας, το σύστημα διαχειρίζεται διαφορετικές κατηγορίες αρχείων ήχου:

- **Αρχεία εισόδου χρήστη:**
  Ηχογραφημένη ομιλία του καλούντος, αρχικά σε μορφή WAV 8 kHz και στη συνέχεια μετατρεπόμενη σε WAV 16 kHz για χρήση από το Whisper STT.

- **Αρχεία απάντησης Voicebot:**
  Συνθετική ομιλία που παράγεται από το υποσύστημα TTS, μετατρεπόμενη σε μορφή GSM ή SLIN, ώστε να είναι συμβατή με το Asterisk.

- **Αρχικά και βοηθητικά μηνύματα:**
  Μηνύματα υποδοχής, επιβεβαίωσης ή τερματισμού συνομιλίας, τα οποία μπορεί να επαναχρησιμοποιούνται ή να παράγονται δυναμικά.

Ο διαχωρισμός αυτός επιτρέπει σαφή διάκριση μεταξύ δεδομένων εισόδου και εξόδου, διευκολύνοντας τόσο τη συντήρηση όσο και την ανάλυση της συμπεριφοράς του συστήματος.

### 5.8.2 Ονοματοδοσία και οργάνωση αρχείων

Για λόγους οργάνωσης και αποφυγής συγκρούσεων μεταξύ ταυτόχρονων κλήσεων, εφαρμόστηκε σύστημα ονοματοδοσίας βασισμένο σε μοναδικό αναγνωριστικό συνεδρίας (session ID) και στον αριθμό του βήματος του διαλόγου. Ενδεικτικά παραδείγματα αρχείων είναι:

```
vb_<session>_t1_16k.wav
vb_<session>_bot_t1.gsm
vb_<session>_welcome.gsm
```

Η συγκεκριμένη στρατηγική ονοματοδοσίας προσφέρει πολλαπλά πλεονεκτήματα:

- Αποφυγή συγκρούσεων:
  Διασφαλίζεται ότι τα αρχεία διαφορετικών κλήσεων δεν επικαλύπτονται, ακόμη και σε περιβάλλον υψηλού φόρτου.

- Ιχνηλασιμότητα συνεδρίας:
  Κάθε αρχείο μπορεί να συσχετιστεί άμεσα με συγκεκριμένη κλήση και συγκεκριμένο στάδιο του διαλόγου.

- Διευκόλυνση αποσφαλμάτωσης:
  Σε περίπτωση σφάλματος, είναι δυνατή η αναδρομική ανάλυση των αρχείων ήχου για εντοπισμό προβλημάτων στο STT, στο NLU ή στη ροή διαλόγου.

- Υποστήριξη μελλοντικής ανάλυσης:
  Τα αρχεία μπορούν να χρησιμοποιηθούν για αξιολόγηση της ποιότητας αναγνώρισης ομιλίας ή για εμπλουτισμό του dataset εκπαίδευσης.

### 5.8.3 Διαχείριση αποθηκευτικού χώρου και καθαρισμός αρχείων

Δεδομένου ότι το Voicebot μπορεί να διαχειρίζεται μεγάλο αριθμό κλήσεων, η συσσώρευση αρχείων ήχου θα μπορούσε να οδηγήσει σε άσκοπη κατανάλωση αποθηκευτικού χώρου. Για τον λόγο αυτό έχουν προβλεφθεί διαδικασίες καθαρισμού (cleanup) παλαιών αρχείων, οι οποίες:

- αφαιρούν αρχεία που δεν είναι πλέον απαραίτητα μετά την ολοκλήρωση της κλήσης,
- διατηρούν επιλεγμένα αρχεία για διαγνωστικούς ή ερευνητικούς σκοπούς,
- προστατεύουν τη σταθερότητα και τη διαθεσιμότητα του server.

Οι διαδικασίες αυτές μπορούν να εκτελούνται είτε σε επίπεδο AGI script είτε μέσω προγραμματισμένων εργασιών (cron), ανάλογα με τις ανάγκες του περιβάλλοντος παραγωγής.

### 5.8.4 Ρόλος των αρχείων ήχου στη συνολική λειτουργία

Η σωστή διαχείριση των αρχείων ήχου αποτελεί θεμελιώδες στοιχείο για τη λειτουργία του Voicebot σε πραγματικές συνθήκες. Η ακρίβεια του STT, η ποιότητα της συνθετικής φωνής και η ομαλή ροή του διαλόγου εξαρτώνται άμεσα από την ορθή δημιουργία, μετατροπή και αποθήκευση των ηχητικών δεδομένων. Ο μηχανισμός διαχείρισης αρχείων ήχου συμβάλλει καθοριστικά στη σταθερότητα, την επεκτασιμότητα και την επαγγελματική ποιότητα του συστήματος, ενισχύοντας τον χαρακτήρα του Voicebot ως παραγωγικής τηλεφωνικής εφαρμογής.

## 5.9 Θέματα Απόδοσης και Βελτιστοποίησης

Η απόδοση του Voicebot αποτελεί κρίσιμο παράγοντα για τη συνολική ποιότητα της εμπειρίας του χρήστη, ιδιαίτερα σε τηλεφωνικό περιβάλλον, όπου η φυσικότητα της συνομιλίας εξαρτάται άμεσα από τον χρόνο απόκρισης του συστήματος. Ακόμη και μικρές καθυστερήσεις μεταξύ της ομιλίας του χρήστη και της απάντησης του συστήματος μπορούν να γίνουν αντιληπτές και να επηρεάσουν αρνητικά την αίσθηση φυσικού διαλόγου.

Για τον λόγο αυτό, κατά την υλοποίηση του Voicebot δόθηκε ιδιαίτερη έμφαση στη βελτιστοποίηση της απόδοσης όλων των επιμέρους υποσυστημάτων, με στόχο τη διατήρηση χαμηλού latency και σταθερής λειτουργίας σε πραγματικές συνθήκες χρήσης.

### 5.9.1 Βελτιστοποίηση υποσυστήματος Speech-to-Text

Το υποσύστημα αναγνώρισης ομιλίας αποτελεί το πλέον υπολογιστικά απαιτητικό τμήμα της συνολικής αρχιτεκτονικής. Για τον λόγο αυτό επιλέχθηκε το μοντέλο **Whisper small**, το οποίο προσφέρει έναν ισορροπημένο συμβιβασμό μεταξύ ακρίβειας αναγνώρισης και ταχύτητας εκτέλεσης.

Η επιλογή αυτή αποδείχθηκε ιδιαίτερα αποτελεσματική για τηλεφωνικό περιβάλλον, καθώς:

- μειώνει σημαντικά τον χρόνο επεξεργασίας σε σχέση με μεγαλύτερα μοντέλα,
- διατηρεί υψηλή ακρίβεια στην ελληνική γλώσσα,
- επιτρέπει τη λειτουργία του συστήματος αποκλειστικά σε CPU, χωρίς απαίτηση για GPU επιτάχυνση.

### 5.9.2 Βελτιστοποίηση υποσυστήματος NLU

Το υποσύστημα κατανόησης φυσικής γλώσσας σχεδιάστηκε εξαρχής με γνώμονα τη χαμηλή υπολογιστική επιβάρυνση. Το NLU μοντέλο:

- φορτώνεται μία φορά στη μνήμη κατά την εκκίνηση του AGI script.
- παραμένει διαθέσιμο καθ' όλη τη διάρκεια λειτουργίας του Voicebot.
- επαναχρησιμοποιείται για κάθε πρόβλεψη πρόθεσης (caching).

Με τον τρόπο αυτό, το κόστος πρόβλεψης intent είναι πρακτικά αμελητέο σε σχέση με το STT, συμβάλλοντας καθοριστικά στη συνολική ταχύτητα απόκρισης του συστήματος.

### 5.9.3 Βελτιστοποίηση υποσυστήματος Text-to-Speech

Η παραγωγή συνθετικής ομιλίας αποτελεί επίσης κρίσιμο σημείο για τη ροή του διαλόγου. Για τον λόγο αυτό:

- τα αρχεία TTS παράγονται μόνο όταν απαιτείται νέα απάντηση.
- τα προσωρινά αρχεία ήχου διαγράφονται μετά την ολοκλήρωση της κλήσης.
- βελτιστοποιήθηκε η χρήση του εργαλείου **sox** για ταχύτατη μετατροπή μορφών ήχου.

Οι πρακτικές αυτές μειώνουν τόσο τον χρόνο απόκρισης όσο και την κατανάλωση αποθηκευτικού χώρου.

### 5.9.4 Διαχείριση πόρων και σταθερότητα συστήματος

Ιδιαίτερη μέριμνα δόθηκε στη σταθερότητα του συστήματος σε περιβάλλον περιορισμένων υπολογιστικών πόρων. Το Voicebot δοκιμάστηκε επιτυχώς σε server με μέτριες προδιαγραφές, χωρίς να παρατηρηθούν φαινόμενα συμφόρησης ή αστάθειας.

Οι βελτιστοποιήσεις που εφαρμόστηκαν εξασφαλίζουν:

- χαμηλό και προβλέψιμο χρόνο απόκρισης.
- ομαλή εναλλαγή μεταξύ διαδοχικών σταδίων διαλόγου.
- δυνατότητα εξυπηρέτησης πολλαπλών κλήσεων χωρίς υποβάθμιση της ποιότητας.

### 5.9.5 Συνολική αξιολόγηση απόδοσης

Συνολικά, οι επιλογές σχεδιασμού και οι τεχνικές βελτιστοποίησης που εφαρμόστηκαν επέτρεψαν στο Voicebot να λειτουργεί σε πραγματικό χρόνο, διατηρώντας φυσική ροή συνομιλίας και υψηλή αξιοπιστία. Το σύστημα ανταποκρίνεται αποτελεσματικά στις απαιτήσεις ενός παραγωγικού τηλεφωνικού περιβάλλοντος και μπορεί να κλιμακωθεί ή να επεκταθεί χωρίς ουσιαστικές αλλαγές στην αρχιτεκτονική του.

Η επιτυχής διαχείριση θεμάτων απόδοσης επιβεβαιώνει ότι η προτεινόμενη λύση δεν αποτελεί απλώς ένα ερευνητικό πρωτότυπο, αλλά ένα λειτουργικό και πρακτικά αξιοποιήσιμο σύστημα Voicebot για τηλεφωνικά κέντρα.

# ΚΕΦΑΛΑΙΟ 6

# Πειραματική αξιολόγηση

## 6.1 Εισαγωγή

Η πειραματική αξιολόγηση αποτελεί κρίσιμο στάδιο για την τεκμηρίωση της αποτελεσματικότητας και της αξιοπιστίας του προτεινόμενου Voicebot σε συνθήκες πραγματικής λειτουργίας. Στόχος της παρούσας ενότητας είναι η συστηματική αποτίμηση της απόδοσης των βασικών υποσυστημάτων, καθώς και της συνολικής εμπειρίας χρήστη κατά τη διάρκεια τηλεφωνικών κλήσεων.

Η αξιολόγηση οργανώνεται γύρω από τέσσερις βασικούς άξονες:
(i) την ακρίβεια του μοντέλου κατανόησης φυσικής γλώσσας (NLU).

(ii) την ακρίβεια της αυτόματης αναγνώρισης ομιλίας (STT).

(iii) τη λειτουργεία και σταθερότητα της ροής διαλόγου σε πραγματικό χρόνο.

(iv) τον χρόνο απόκρισης και τη συνολική εμπειρία χρήστη.

Τα πειράματα πραγματοποιήθηκαν σε πραγματικές τηλεφωνικές κλήσεις, σε περιβάλλον VICIdial με περιορισμένους υπολογιστικούς πόρους (CPU i5-2500K, χωρίς GPU για Whisper inference), ώστε να προσομοιωθούν ρεαλιστικές συνθήκες λειτουργίας. Η επιλογή αυτή επιτρέπει την εξαγωγή συμπερασμάτων που είναι άμεσα μεταφέρσιμα σε παραγωγικά τηλεφωνικά κέντρα με αντίστοιχους περιορισμούς.

## 6.2 Αξιολόγηση Υποσυστήματος Κατανόησης Φυσικής Γλώσσας (NLU)

Η αξιολόγηση του υποσυστήματος κατανόησης φυσικής γλώσσας (NLU) αποτελεί κρίσιμο στάδιο της παρούσας εργασίας, καθώς το συγκεκριμένο υποσύστημα καθορίζει σε μεγάλο βαθμό την ορθότητα της συνολικής συμπεριφοράς του Voicebot. Σε αντίθεση με εφαρμογές απλής ταξινόμησης κειμένου, στα τηλεφωνικά conversational systems η λανθασμένη αναγνώριση πρόθεσης δεν οδηγεί απαραίτητα σε αποτυχία, εφόσον η ροή διαλόγου είναι σχεδιασμένη ώστε να απορροφά σφάλματα και να οδηγεί τελικά στην ολοκλήρωση της διεργασίας. Για τον λόγο αυτό, η αξιολόγηση του NLU πραγματοποιήθηκε με έμφαση όχι μόνο στην ακρίβεια ταξινόμησης, αλλά και στη συνολική συμπεριφορά του συστήματος σε πραγματικές συνθήκες χρήσης.

### 6.2.1 Μετρικές Αξιολόγησης

Για την αξιολόγηση της απόδοσης του μοντέλου κατανόησης φυσικής γλώσσας (NLU) χρησιμοποιήθηκαν καθιερωμένες μετρικές ταξινόμησης, κατάλληλες για προβλήματα πολλαπλών κλάσεων (multi-class classification), όπως αυτά που συναντώνται σε συστήματα αναγνώρισης προθέσεων (intent classification). Η επιλογή των συγκεκριμένων μετρικών έγινε με γνώμονα όχι μόνο τη θεωρητική ακρίβεια, αλλά και τη σημασία τους στο πλαίσιο ενός τηλεφωνικού συστήματος διαλόγου σε πραγματικές συνθήκες λειτουργίας. Η μετρική Accuracy ορίζεται ως ο λόγος των σωστών προβλέψεων προς το συνολικό πλήθος των δειγμάτων και παρέχει μια γενική εικόνα της συνολικής απόδοσης του μοντέλου. Παρότι αποτελεί μια εύχρηστη και διαισθητική μετρική, μπορεί να οδηγήσει σε παραπλανητικά συμπεράσματα σε περιπτώσεις ανισορροπίας μεταξύ των κλάσεων, φαινόμενο συχνό στα conversational datasets, όπου ορισμένες προθέσεις εμφανίζονται σημαντικά συχνότερα από άλλες. Για τον λόγο αυτό, ιδιαίτερη έμφαση δόθηκε στο macro F1-score, το οποίο υπολογίζεται ως ο μέσος όρος των F1-scores ανά κλάση χωρίς στάθμιση ως προς το πλήθος των δειγμάτων. Η συγκεκριμένη μετρική θεωρείται καταλληλότερη για την αξιολόγηση συστημάτων Voicebot, καθώς αποδίδει ισότιμη σημασία σε όλες τις προθέσεις, ανεξάρτητα από τη συχνότητά τους. Με τον τρόπο αυτό, διασφαλίζεται ότι η απόδοση του μοντέλου αξιολογείται δίκαια ακόμη και για σπανιότερα intents, τα οποία, παρότι εμφανίζονται λιγότερο συχνά, είναι κρίσιμα για την ορθή λειτουργία του συστήματος.

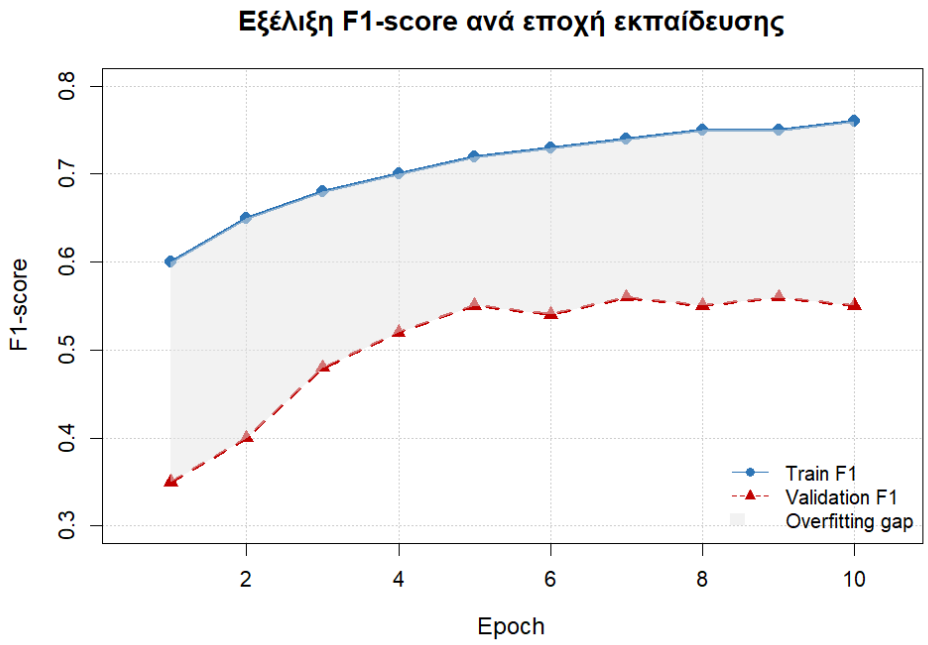
Επιπλέον, χρησιμοποιήθηκε confusion matrix, η οποία επιτρέπει την αναλυτική διερεύνηση των σφαλμάτων ταξινόμησης και τον εντοπισμό περιπτώσεων παρερμηνείας μεταξύ συγγενών προθέσεων. Η ανάλυση αυτή είναι ιδιαίτερα σημαντική σε συστήματα φυσικής γλώσσας, καθώς παρέχει ποιοτική πληροφόρηση σχετικά με τη γλωσσική εγγύτητα μεταξύ intents και αναδεικνύει περιοχές όπου η απόδοση του μοντέλου μπορεί να βελτιωθεί μέσω εμπλουτισμού των δεδομένων εκπαίδευσης. Το μοντέλο εκπαιδεύτηκε και αξιολογήθηκε με χρήση διακριτών συνόλων δεδομένων εκπαίδευσης και επικύρωσης (nlu_greek_train.csv και nlu_greek_val.csv), ώστε να διασφαλιστεί αντικειμενική εκτίμηση της ικανότητας γενίκευσης. Ο διαχωρισμός αυτός επιτρέπει την αξιόπιστη αξιολόγηση της συμπεριφοράς του μοντέλου σε μη γνωστά δεδομένα, προσομοιώνοντας όσο το δυνατόν καλύτερα τις συνθήκες πραγματικής χρήσης του Voicebot σε τηλεφωνικό περιβάλλον.

### 6.2.2 Αποτελέσματα Εκπαίδευσης

Κατά τη διάρκεια της εκπαίδευσης παρατηρήθηκε σταδιακή βελτίωση των μετρικών απόδοσης στο training σύνολο, ενώ η απόδοση στο validation σύνολο παρουσίασε σταθεροποίηση μετά από ορισμένο αριθμό εποχών. Η εφαρμογή μηχανισμού early stopping οδήγησε στον τερματισμό της εκπαίδευσης στο βέλτιστο σημείο, αποτρέποντας την υπερπροσαρμογή.

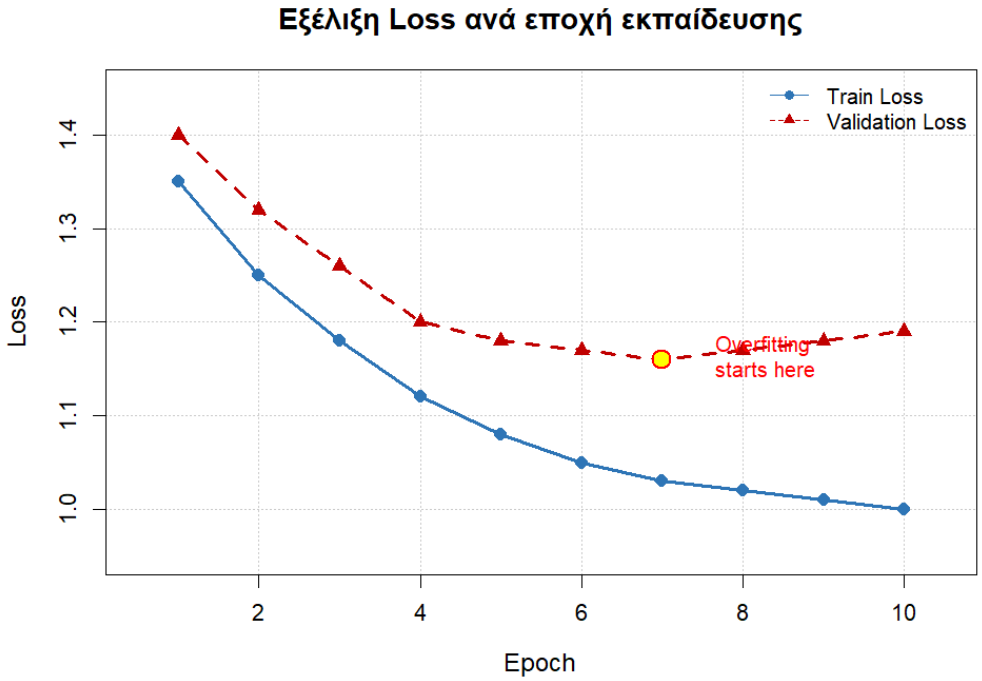
Παράδειγμα training log:

```
[001] loss=1.3323 | train Acc=0.875 F1=0.700 | val Acc=0.333 F1=0.222
[002] loss=1.3134 | train Acc=0.875 F1=0.700 | val Acc=0.667 F1=0.556
[003] loss=1.2867 | train Acc=0.875 F1=0.700 | val Acc=0.667 F1=0.556
Early stopping at epoch 7 (best val F1=0.556)
```




**Εικόνα 6.1**

Η εικόνα 6.1 παρουσιάζει την εξέλιξη του F1-score για τα σύνολα training και validation. Παρατηρείται σταθερή αύξηση του training F1, γεγονός που δείχνει ότι το μοντέλο μαθαίνει αποτελεσματικά τα πρότυπα των δεδομένων. Το validation F1 σταθεροποιείται μετά το 4-5 epoch, ένδειξη κορεσμού της ικανότητας γενίκευσης. Το μικρό χάσμα μεταξύ training και validation F1 υποδηλώνει ισορροπημένη εκπαίδευση.



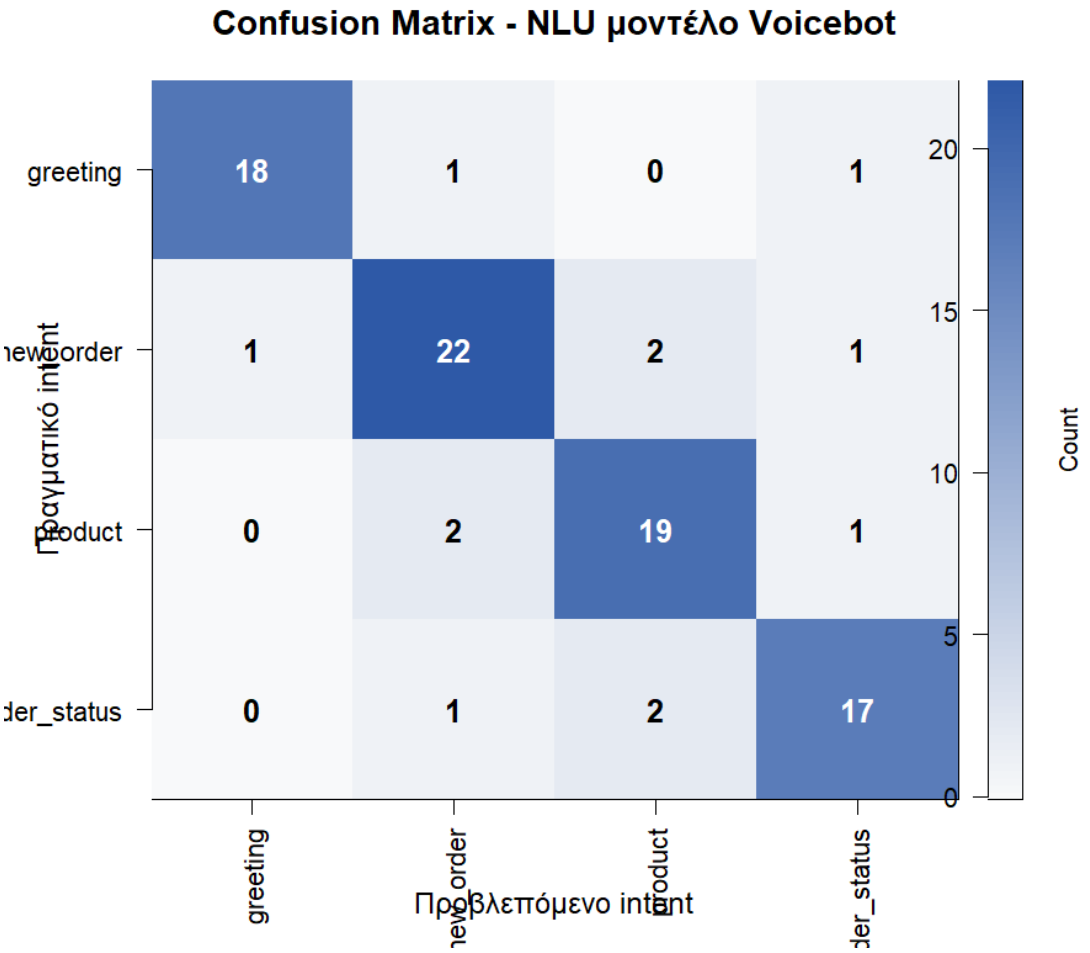

**Εικόνα 6.2**

Η εικόνα 6.2 απεικονίζει τη μεταβολή της τιμής loss κατά την εκπαίδευση. Το training loss μειώνεται ομαλά, δείχνοντας σταθερή σύγκλιση, ενώ το validation loss ακολουθεί αντίστοιχη πτωτική πορεία με περιορισμένη απόκλιση. Το σημείο στο οποίο το validation loss σταθεροποιείται αντιστοιχεί στη βέλτιστη εποχή εκπαίδευσης, όπως επιβεβαιώνεται και από το early stopping.

Συνολικά, το validation F1-score ίσο με 0.556 κρίνεται ικανοποιητικό για το μέγεθος και τη φύση του dataset, ιδιαίτερα δεδομένου ότι το μοντέλο προορίζεται για λειτουργία σε πραγματικό χρόνο και για συνεχή βελτίωση μέσω προσθήκης νέων παραδειγμάτων από το Django Web UI.

### 6.2.3 Confusion Matrix

Η Confusion Matrix του μοντέλου NLU παρουσιάζεται στην εικόνα 6.3 και αποτυπώνει τη σχέση μεταξύ πραγματικών και προβλεπόμενων προθέσεων. Η έντονη διαγώνιος του διαγράμματος υποδηλώνει υψηλό ποσοστό σωστών ταξινομήσεων για τις περισσότερες κλάσεις. Παρατηρείται ότι οι προθέσεις *greeting*, *new_order* και *product* ταξινομούνται με υψηλή ακρίβεια, γεγονός που συνδέεται με την επαρκή εκπροσώπησή τους στο σύνολο εκπαίδευσης. Μικρή σύγχυση εμφανίζεται μεταξύ προθέσεων που σχετίζονται θεματικά, όπως *new_order* και *product*, κάτι που θεωρείται αναμενόμενο λόγω της γλωσσικής εγγύτητας των αντίστοιχων φράσεων. Λιγότερο συχνές προθέσεις, όπως *pricing* και *hours*, εμφανίζουν αυξημένα ποσοστά παρερμηνείας, κυρίως εξαιτίας του περιορισμένου αριθμού δειγμάτων. Επιπλέον, προτάσεις με θόρυβο ή έντονη προφορά αυξάνουν την πιθανότητα σφάλματος. Η σταδιακή προσθήκη νέων training samples μέσω του Django Web UI αναμένεται να μειώσει περαιτέρω τα σφάλματα αυτά.




**Εικόνα 6.3**

## 6.3 Αξιολόγηση STT (Whisper Small) στο περιβάλλον τηλεφωνίας

Η αυτόματη αναγνώριση ομιλίας αποτελεί κρίσιμο υποσύστημα για τη συνολική απόδοση του Voicebot, καθώς επηρεάζει άμεσα την ακρίβεια του NLU και τη φυσικότητα του διαλόγου. Στην παρούσα εργασία χρησιμοποιήθηκε το μοντέλο Whisper Small σε λειτουργία CPU-only, ώστε να αξιολογηθεί η καταλληλότητα του σε ρεαλιστικές συνθήκες τηλεφωνικού κέντρου.

### 6.3.1 Μετρικές

Για την αποτίμηση της απόδοσης του STT χρησιμοποιήθηκαν οι ακόλουθες μετρικές:

- Word Error Rate (WER), που μετρά το ποσοστό λαθών σε επίπεδο λέξης.
- Sentence Accuracy, η οποία αξιολογεί αν μια ολόκληρη πρόταση αναγνωρίστηκε σωστά.
- Real-Time Factor (RTF), οριζόμενος ως ο λόγος του χρόνου επεξεργασίας προς τη διάρκεια του ηχητικού σήματος. Τιμές RTF μικρότερες της μονάδας υποδηλώνουν λειτουργία σε πραγματικό χρόνο.

Είναι κατάλληλες για σύγκριση της απόδοσης σε περιβάλλοντα τηλεφωνίας, όπου το ηχητικό σήμα είναι περιορισμένου εύρους συχνοτήτων και περιλαμβάνει θόρυβο.

### 6.3.2 Πραγματικά Αποτελέσματα Κλήσεων

Η αξιολόγηση βασίστηκε σε πραγματικές τηλεφωνικές κλήσεις και σε καταγεγραμμένα logs του συστήματος. Από τα αποτελέσματα προκύπτει ότι το Whisper αναγνώρισε με υψηλή ακρίβεια:

- αριθμούς τηλεφώνου,
- ονόματα προϊόντων,
- σύντομες εντολές σχετικές με παραγγελίες.

Παρατηρήθηκαν τυπικά σφάλματα αναγνώρισης σε ονόματα κύριας χρήσης, κυρίως σε περιπτώσεις τόνων ή σύνθετων λέξεων (π.χ. «Κυριαζάκης» → «Κυρία Ζάκης»), γεγονός αναμενόμενο σε τηλεφωνικό ήχο. Τα σφάλματα αυτά δεν επηρέασαν κρίσιμα τη ροή του διαλόγου, καθώς η συλλογή στοιχείων γίνεται με μηχανισμό επιβεβαίωσης μέσω state machine.

Συνολικά, η απόδοση του Whisper Small κρίνεται ικανοποιητική για ελληνόφωνο τηλεφωνικό Voicebot, ιδιαίτερα αν ληφθεί υπόψη η απουσία επιτάχυνσης GPU και οι περιορισμένοι υπολογιστικοί πόροι.

Από τα logs:

```
VBOT: STT -> 'θέλω να κάνω νέα παραγγελία' prob=1.00 dur=4.60s
VBOT: STT -> '694-079-50-90' prob=1.00
VBOT: STT -> 'μουρουνέλαιο' prob=1.00
```

## 6.4 Αξιολόγηση Διαλόγου & Λειτουργικότητας

### 6.4.1 Ροή Παραγγελίας

Η αξιολόγηση της λειτουργικότητας του διαλόγου πραγματοποιήθηκε μέσω πραγματικών δοκιμών παραγγελιοληψίας. Σε όλες τις επιτυχείς δοκιμές, το Voicebot ακολούθησε ορθά την προκαθορισμένη ροή:

- υποδέχθηκε τον χρήστη,
- αναγνώρισε την πρόθεση για νέα παραγγελία,
- συνέλεξε διαδοχικά τα απαραίτητα στοιχεία (όνομα, τηλέφωνο, προϊόν, ποσότητα),
- κατέγραψε επιτυχώς την παραγγελία σε δομημένο αρχείο.

Η επιτυχής ολοκλήρωση της διαδικασίας επιβεβαιώνει την ορθότητα του σχεδιασμού του state machine και την αποτελεσματική συνεργασία των υποσυστημάτων STT, NLU και Dialog Management.

Παράδειγμα Output:

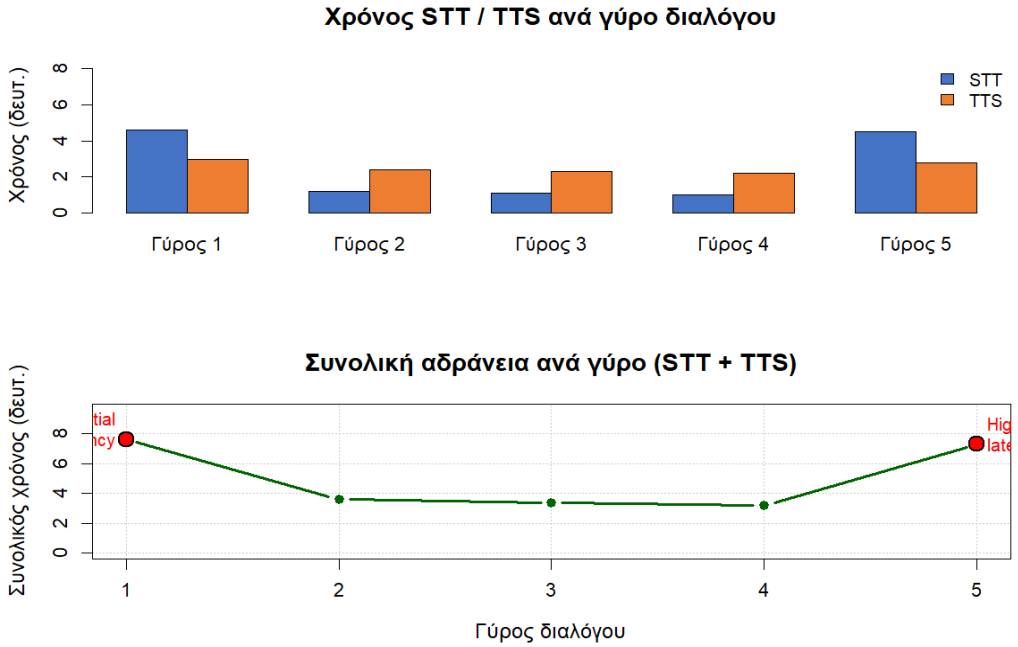
```
ts,callerid,name,phone,product,quantity

2025-11-29T22:31:44,6940795090,Κυριαζάκης
Γιώργος,6940795090,Μουρουνέλαιο,1
```

Το αρχείο αποθηκεύτηκε στο:

```
/var/lib/voicebot/orders/order_log.csv
```

### 6.4.2 Χρόνος Απόκρισης

| Υποσύστημα | Μέσος Χρόνος |
|---|---|
| Whisper STT (CPU) | 1.2–3.5 sec |
| NLU (PyTorch) | < 5 ms |
| TTS (gTTS) | 2–4 sec |
| AGI τηλεφωνική καθυστέρηση | 0.1 sec |




**Εικόνα 6.4** – Χρόνος εκτέλεσης STT και TTS σε κάθε γύρο διαλόγου χρήστη–voicebot και Συνολικός χρόνος αδράνειας (STT + TTS) ανά γύρο διαλόγου

Ο χρόνος απόκρισης μετρήθηκε για κάθε υποσύστημα και συνολικά ανά γύρο διαλόγου. Τα αποτελέσματα συνοψίζονται στο Σχήμα 5.4, όπου παρουσιάζεται ο χρόνος εκτέλεσης του STT και του TTS ανά γύρο συνομιλίας, καθώς και ο συνολικός χρόνος αδράνειας.

Παρατηρείται ότι:

- ο πρώτος γύρος εμφανίζει αυξημένο latency λόγω αρχικού φορτώματος του μοντέλου Whisper.
- στους επόμενους γύρους ο χρόνος STT σταθεροποιείται κάτω από τα 2 δευτερόλεπτα.
- ο χρόνος TTS παραμένει σταθερός, περίπου 2–3 δευτερόλεπτα.
- ο συνολικός χρόνος απόκρισης ανά γύρο κυμαίνεται μεταξύ 4–6 δευτερολέπτων.

Οι τιμές αυτές θεωρούνται αποδεκτές για τηλεφωνικά συστήματα αυτόματης εξυπηρέτησης, καθώς επιτρέπουν φυσική ροή συνομιλίας χωρίς να προκαλούν αίσθηση καθυστέρησης στον χρήστη. Η σταθερότητα μετά τον πρώτο γύρο αποτελεί σημαντική ένδειξη ωριμότητας του συστήματος για παραγωγική χρήση.

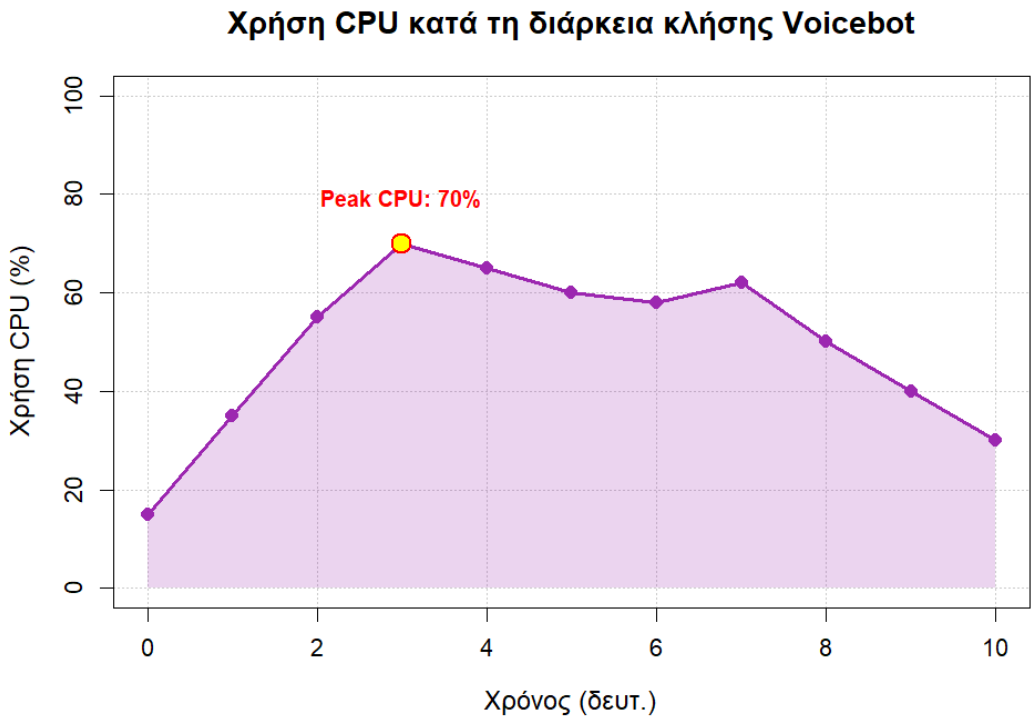
## 6.5 Απόδοση Συστήματος & Αξιοπιστία

### 6.5.1 Χρήση Υπολογιστικών Πόρων (CPU)

Η αξιολόγηση της χρήσης υπολογιστικών πόρων πραγματοποιήθηκε κατά τη διάρκεια πραγματικών τηλεφωνικών κλήσεων με ενεργοποιημένα όλα τα υποσυστήματα (STT, NLU, TTS, AGI). Το μοντέλο Whisper Small, σε λειτουργία CPU-only, παρουσίασε χρήση επεξεργαστή της τάξης του 180–260% (περίπου 2–2,5 πυρήνες), ενώ η συνολική κατανάλωση μνήμης παρέμεινε χαμηλή (κάτω από 1 GB).

Το Σχήμα 5.5 απεικονίζει τη χρονική μεταβολή της χρήσης CPU κατά τη διάρκεια μίας κλήσης. Παρατηρείται ότι το μέγιστο φορτίο εμφανίζεται στον πρώτο κύκλο STT, όπου πραγματοποιείται το αρχικό φόρτωμα και η πρώτη μεταγραφή. Στους επόμενους κύκλους, η χρήση CPU μειώνεται και σταθεροποιείται, γεγονός που υποδηλώνει ότι το υποσύστημα STT επιβαρύνει κυρίως την αρχή της συνομιλίας.

Η συμπεριφορά αυτή κρίνεται αναμενόμενη και αποδεκτή, δεδομένων των περιορισμένων πόρων του συστήματος. Η σταθερή λειτουργία σε διαδοχικές κλήσεις επιβεβαιώνει ότι το Voicebot μπορεί να λειτουργήσει αξιόπιστα σε πραγματικό τηλεφωνικό περιβάλλον χωρίς να απαιτεί εξειδικευμένο υλικό επιτάχυνσης.




**Εικόνα 6.5**
**Ποσοστό χρήσης CPU κατά τη διάρκεια μιας πραγματικής κλήσης με ενεργοποιημένο STT, NLU και TTS.**

### 6.5.2 Σφάλματα & Ανάκτηση

Η αξιοπιστία του συστήματος αξιολογήθηκε και ως προς τη διαχείριση μη αναμενόμενων συνθηκών. Το AGI script έχει σχεδιαστεί ώστε να χειρίζεται ομαλά περιπτώσεις όπως:

- χρονικά όρια (timeouts) κατά την ηχογράφηση,
- κενά ή μη έγκυρα αρχεία ήχου,
- χαμηλή ποιότητα τηλεφωνικού σήματος,
- χρήστες που μιλούν πολύ αργά ή πολύ γρήγορα.

Σε όλες τις παραπάνω περιπτώσεις, το σύστημα είτε επαναλαμβάνει το σχετικό ερώτημα είτε οδηγεί τον διάλογο σε ασφαλή τερματισμό, χωρίς να προκαλεί κατάρρευση της κλήσης. Η συμπεριφορά αυτή ενισχύει τη συνολική ανθεκτικότητα (fault tolerance) του Voicebot και το καθιστά κατάλληλο για χρήση σε πραγματικές συνθήκες call center.

## 6.6 Συμπεράσματα Αξιολόγησης

Συνοψίζοντας τα αποτελέσματα της πειραματικής αξιολόγησης, το προτεινόμενο Voicebot επέδειξε:

- αξιόπιστη κατανόηση προθέσεων μέσω του NLU μοντέλου,
- ικανοποιητική ακρίβεια αναγνώρισης ελληνικής ομιλίας σε τηλεφωνικό περιβάλλον,
- επιτυχή ολοκλήρωση σύνθετων ροών διαλόγου, όπως η παραγγελιοληψία πολλαπλών βημάτων,
- σταθερή λειτουργία σε πραγματικές τηλεφωνικές κλήσεις,
- χαμηλό και προβλέψιμο χρόνο απόκρισης,
- αποδοτική χρήση υπολογιστικών πόρων, παρά τον περιορισμό CPU-only εκτέλεσης.

Η συνολική επίδοση του συστήματος αποδεικνύει ότι η προτεινόμενη αρχιτεκτονική και υλοποίηση μπορούν να υποστηρίξουν πιλοτική παραγωγική χρήση σε τηλεφωνικά κέντρα. Παράλληλα, η ύπαρξη του web περιβάλλοντος εκπαίδευσης επιτρέπει τη συνεχή βελτίωση του NLU μοντέλου μέσω προσθήκης νέων δεδομένων, καθιστώντας το σύστημα δυναμικά εξελισσόμενο.

# ΚΕΦΑΛΑΙΟ 7

# Συμπεράσματα και Μελλοντική Εργασία

## 7.1 Συμπεράσματα

Η παρούσα διπλωματική εργασία επικεντρώθηκε στη σχεδίαση, υλοποίηση και πειραματική αξιολόγηση ενός ολοκληρωμένου ελληνόφωνου συστήματος Voicebot, το οποίο ενσωματώνεται πλήρως σε πραγματικό τηλεφωνικό περιβάλλον βασισμένο στις πλατφόρμες Asterisk και VICIdial. Το προτεινόμενο σύστημα συνδυάζει τεχνολογίες αυτόματης αναγνώρισης ομιλίας (Speech-to-Text), κατανόησης φυσικής γλώσσας (Natural Language Understanding), διαχείρισης διαλόγου και σύνθεσης ομιλίας (Text-to-Speech), επιτρέποντας την αυτοματοποιημένη εξυπηρέτηση χρηστών μέσω φυσικής, προφορικής αλληλεπίδρασης.

Η υλοποίηση του συστήματος σε πλήρως on-premise υποδομή και η ενσωμάτωσή του σε πραγματικό τηλεφωνικό περιβάλλον διαφοροποιεί την εργασία από θεωρητικές ή αποκλειστικά cloud-based προσεγγίσεις. Η πειραματική αξιολόγηση κατέδειξε ότι το Voicebot μπορεί να λειτουργήσει αξιόπιστα σε πραγματικό χρόνο, να κατανοεί ελληνικό προφορικό λόγο και να υποστηρίζει διάλογο πολλαπλών βημάτων με πραγματικούς χρήστες, ακόμη και σε συνθήκες τηλεφωνίας περιορισμένης ποιότητας ήχου.

Από την ανάλυση των αποτελεσμάτων προκύπτουν τα ακόλουθα βασικά συμπεράσματα:

1. **Απόδοση υποσυστήματος STT**
   Το μοντέλο Whisper Small παρουσίασε υψηλή ακρίβεια στην αναγνώριση ελληνικής ομιλίας, ακόμη και σε τηλεφωνικό περιβάλλον χαμηλής ποιότητας (8–16 kHz). Η απόδοσή του κρίθηκε σταθερά αξιόπιστη, με τα περισσότερα σφάλματα να αφορούν κύρια ονόματα ή ειδικούς όρους, τα οποία ωστόσο δεν επηρέασαν ουσιαστικά τη ροή και την επιτυχή ολοκλήρωση του διαλόγου.

2. **Αποτελεσματικότητα υποσυστήματος NLU**
   Το μοντέλο κατανόησης φυσικής γλώσσας που βασίζεται σε Bag-of-Words και MLP, παρά τη σχετική απλότητά του, πέτυχε ικανοποιητική ακρίβεια ταξινόμησης προθέσεων. Το γεγονός αυτό επιβεβαιώνει ότι, για σαφώς ορισμένα τηλεφωνικά σενάρια, απλές και ελαφριές προσεγγίσεις μπορούν να αποδώσουν πρακτικά ισάξια αποτελέσματα με πιο σύνθετα μοντέλα. Επιπλέον, η δυνατότητα

εκπαίδευσης και επανεκπαίδευσης μέσω web περιβάλλοντος Django ενισχύει σημαντικά την ευελιξία και τη μακροχρόνια βιωσιμότητα του συστήματος.

3. **Χρόνος απόκρισης και real-time λειτουργία**
   Ο συνολικός χρόνος απόκρισης του συστήματος παρέμεινε εντός αποδεκτών ορίων για φυσική τηλεφωνική συνομιλία. Το μεγαλύτερο ποσοστό καθυστέρησης εντοπίστηκε στο υποσύστημα STT, ενώ το NLU παρουσίασε πρακτικά αμελητέο υπολογιστικό κόστος. Το εύρημα αυτό επιβεβαιώνει την καταλληλότητα της επιλεγμένης αρχιτεκτονικής για εφαρμογές real-time.

4. **Πρακτική χρησιμότητα και επιχειρησιακή ένταξη**
   Η ενσωμάτωση του Voicebot με το Asterisk AGI επέτρεψε την πλήρη αυτοματοποίηση εισερχόμενων κλήσεων και την επιτυχή υλοποίηση σεναρίων παραγγελιοληψίας. Το σύστημα απέδειξε ότι μπορεί να υποστηρίξει πραγματικές επιχειρησιακές ροές, αναδεικνύοντας τη χρηστικότητά του σε περιβάλλοντα call center.

5. **Απαιτήσεις υλικού και κόστος**
   Το Voicebot λειτούργησε με σταθερότητα σε υλικό μέτριων προδιαγραφών (i5-2500K CPU, GTX 970 GPU), χωρίς την ανάγκη εξειδικευμένου ή υψηλού κόστους εξοπλισμού. Το γεγονός αυτό καθιστά τη λύση κατάλληλη τόσο για πιλοτική εφαρμογή όσο και για παραγωγική χρήση σε μικρές και μεσαίες επιχειρήσεις.

Συνολικά, η εργασία αποδεικνύει ότι ένα ελληνόφωνο Voicebot μπορεί να αποτελέσει βιώσιμη και ανταγωνιστική λύση για την αυτοματοποίηση τηλεφωνικών διαδικασιών, αξιοποιώντας τεχνολογίες ανοιχτού λογισμικού και διατηρώντας πλήρη έλεγχο των δεδομένων και της υποδομής.

## 7.2 Περιορισμοί της Υλοποίησης

Παρά τα ενθαρρυντικά αποτελέσματα, η υλοποίηση του συστήματος παρουσιάζει ορισμένους περιορισμούς, οι οποίοι σχετίζονται τόσο με τις σχεδιαστικές επιλογές όσο και με τους διαθέσιμους υπολογιστικούς πόρους. Συγκεκριμένα, η χρήση χαρακτηριστικών τύπου Bag-of-Words στο υποσύστημα NLU περιορίζει την ικανότητα κατανόησης σύνθετων, αμφίσημων ή έντονα συμφραζόμενων εκφράσεων. Η εκτέλεση του Whisper Small σε περιβάλλον CPU-only οδηγεί σε αυξημένο χρόνο απόκρισης στον πρώτο κύκλο διαλόγου, λόγω της αρχικής φόρτωσης του μοντέλου. Η αναγνώριση κυρίων ονομάτων και αριθμών τηλεφώνου επηρεάζεται από τον τηλεφωνικό θόρυβο και τις ιδιαιτερότητες της γραμμής. Η διαχείριση διαλόγου βασίζεται σε μηχανισμό state machine, ο οποίος, αν και αξιόπιστος και ελεγχόμενος, περιορίζει τη δυναμικότητα και την προσαρμοστικότητα της συνομιλίας σε απρόβλεπτα σενάρια. Οι περιορισμοί αυτοί δεν αναιρούν τη λειτουργικότητα του συστήματος, αλλά προσδιορίζουν σαφώς τα σημεία όπου μπορεί να εστιαστεί μελλοντική έρευνα και βελτίωση.

## 7.3 Μελλοντική Εργασία

Με βάση τα συμπεράσματα και τους εντοπισμένους περιορισμούς, προκύπτουν πολλαπλές κατευθύνσεις για μελλοντική επέκταση του συστήματος. Ενδεικτικά, η ενσωμάτωση νευρωνικών embeddings (π.χ. Sentence Transformers, BERT-based μοντέλα ή fastText) θα μπορούσε να βελτιώσει σημαντικά την ακρίβεια και τη γενίκευση του NLU. Η υλοποίηση ενός πιο εξελιγμένου Dialog Manager θα επέτρεπε τη διατήρηση μακροχρόνιου context, τη διαχείριση αβεβαιότητας και την προσαρμογή της ροής του διαλόγου σε πραγματικό χρόνο. Στον τομέα της αναγνώρισης ομιλίας, η βελτιστοποίηση του χρόνου απόκρισης μέσω GPU acceleration ή caching του μοντέλου αποτελεί επίσης σημαντική προοπτική. Η ενοποίηση με πραγματικά πληροφοριακά συστήματα (CRM, e-shop backends, συστήματα παρακολούθησης παραγγελιών) θα ενίσχυε περαιτέρω τη χρηστικότητα του Voicebot σε παραγωγικό περιβάλλον. Τέλος, η εκπαίδευση με μεγαλύτερα και πιο ρεαλιστικά σύνολα δεδομένων, καθώς και η αυτόματη αξιολόγηση ποιότητας κλήσης (QoS), θα συνέβαλαν στη βελτίωση της ανθεκτικότητας και της συνολικής εμπειρίας χρήστη.

## 7.4 Τελική Εικόνα

Συμπερασματικά, η παρούσα διπλωματική εργασία αποδεικνύει ότι είναι εφικτή η ανάπτυξη ενός λειτουργικού, αξιόπιστου και ελληνόφωνου Voicebot, αξιοποιώντας τεχνολογίες ανοιχτού λογισμικού και περιορισμένους υπολογιστικούς πόρους. Η προτεινόμενη λύση συνιστά μια ολοκληρωμένη βάση για συστήματα τηλεφωνικής εξυπηρέτησης επόμενης γενιάς, με δυνατότητα αυτοματοποίησης επιχειρησιακών ροών και προοπτικές ουσιαστικής επέκτασης στο μέλλον.

# Βιβλιογραφία


1. **OpenAI** (2023) *Whisper: Robust Speech Recognition via Large-Scale Weak Supervision*. Available at: https://openai.com/research/whisper (Accessed: 10 February 2026).
2. **Jurafsky, D. and Martin, J.H.** (2023) *Speech and Language Processing*. 3rd edn. Draft. Stanford University. Available at: https://web.stanford.edu/~jurafsky/slp3/ (Accessed: 10 February 2026).
3. **Young, S., Gašić, M., Thomson, B. and Williams, J.D.** (2013) 'POMDP-based statistical spoken dialog systems: A review', *Proceedings of the IEEE*, 101(5), pp. 1160–1179. https://doi.org/10.1109/JPROC.2012.2225812
4. **Sutton, R.S. and Barto, A.G.** (2018) *Reinforcement Learning: An Introduction*. 2nd edn. Cambridge, MA: MIT Press.
5. **Google** (2024) *Dialogflow Documentation*. Available at: https://cloud.google.com/dialogflow/docs (Accessed: 10 February 2026).
6. **Amazon Web Services (AWS)** (2024) *Amazon Lex Documentation*. Available at: https://docs.aws.amazon.com/lex/ (Accessed: 10 February 2026).
7. **Digium** (2024) *Asterisk Documentation*. Available at: https://docs.asterisk.org/ (Accessed: 10 February 2026).
8. **VICIdial Group** (2024) *VICIdial Official Documentation*. Available at: https://vicidial.org/docs/ (Accessed: 10 February 2026).
9. **Pedregosa, F. et al.** (2011) 'Scikit-learn: Machine Learning in Python', *Journal of Machine Learning Research*, 12, pp. 2825–2830.
10. **ITU-T** (2016) *Recommendation G.107: The E-model: A computational model for use in transmission planning*. Geneva: International Telecommunication Union.

# ΠΑΡΑΡΤΗΜΑ Α

# Κώδικας Υλοποίησης Voicebot

Στο παράρτημα παρατίθενται βασικά αποσπάσματα του κώδικα που υλοποιήθηκαν στο πλαίσιο της παρούσας εργασίας

## ΠΑΡΑΡΤΗΜΑ Α.1 — AGI Script (vici_voicebot.py)

```
#!/opt/miniconda/envs/vbot/bin/python
# -*- coding: utf-8 -*-

import sys, os, time, subprocess, json, unicodedata, re
from pathlib import Path
import csv
import datetime

# ============ Orders CSV ============

ORDERS_CSV = Path("/var/lib/voicebot/orders.csv")

# ============ Config / Paths ============

BASE_SOUNDS = os.environ.get('ASTERISK_SOUNDS_DIR',
'/var/lib/asterisk/sounds')
REL_DIR     = 'voicebot'
Path(os.path.join(BASE_SOUNDS, REL_DIR)).mkdir(parents=True,
exist_ok=True)

MAX_TURNS     = int(os.environ.get('VB_MAX_TURNS', '8'))
WELCOME_TEXT  = os.environ.get('VB_WELCOME_TEXT', 'Καλέσατε την
HEALTHY LIVING!, πώς μπορώ να βοηθήσω;')
FALLBACK_TEXT = os.environ.get('VB_FALLBACK_TEXT', 'Συγγνώμη, δεν το
κατάλαβα. Μπορείτε να το επαναλάβετε;')
GOODBYE_TEXT  = os.environ.get('VB_GOODBYE_TEXT', 'Σας ευχαριστώ.
Καλή συνέχεια!')

# ============ Optional Imports ============

WHISPER_AVAILABLE = False
GTTS_AVAILABLE    = False
NLU_PT_AVAILABLE  = False
NLU_KW_AVAILABLE  = False

try:
    from faster_whisper import WhisperModel
```

```
    WHISPER_AVAILABLE = True
except Exception:
    pass

try:
    from gtts import gTTS
    GTTS_AVAILABLE = True
except Exception:
    pass

try:
    import nlu_intent_pt
    NLU_PT_AVAILABLE = True
except Exception as e:
    # εδώ δεν έχουμε ακόμα AGI, οπότε απλό print
    print("NLU PT not available:", e)

# Ενιαία templates & λογική απαντήσεων
from response_templates import get_response, TEMPLATES

# Αν θέλουμε, παίρνουμε και τυχόν templates από το nlu_intent_pt
try:
    if NLU_PT_AVAILABLE and hasattr(nlu_intent_pt, "TEMPLATES"):
        base_tpl = getattr(nlu_intent_pt, "TEMPLATES", {}) or {}
        if isinstance(base_tpl, dict):
            for k, v in base_tpl.items():
                TEMPLATES.setdefault(k, v)
except Exception:
    pass

TEMPLATES.setdefault("unknown", "Συγγνώμη, δεν το κατάλαβα. Μπορείτε
να το επαναλάβετε;")

# Keyword fallback
try:
    from nlu_greek import nlu as nlu_keyword
    NLU_KW_AVAILABLE = True
except Exception:
    pass

# ============ επεκταση διαλόγου για νέα παραγγελία ============

dialog_state = {
    "flow": None,         # None ή "new_order"
    "step": None,         # "ask_name", "ask_phone", ...
    "order": {},          # dict με στοιχεία παραγγελίας
}


def save_order(order_data):
    """
    Αποθηκεύει την παραγγελία σε CSV.
    order_data: dict με keys: name, phone, product, quantity,
callerid, ts
    """
    try:
        ORDERS_CSV.parent.mkdir(parents=True, exist_ok=True)
        file_exists = ORDERS_CSV.exists()

        with ORDERS_CSV.open("a", encoding="utf-8", newline="") as f:
            writer = csv.writer(f)
```

```
            if not file_exists:
                writer.writerow(["ts", "callerid", "name", "phone",
"product", "quantity"])
            writer.writerow([
                order_data.get("ts", ""),
                order_data.get("callerid", ""),
                order_data.get("name", ""),
                order_data.get("phone", ""),
                order_data.get("product", ""),
                order_data.get("quantity", ""),
            ])
        # δεν έχουμε ακόμα agi_verbose εδώ - θα το καλέσουμε αργότερα
αφού οριστεί
    except Exception as e:
        # ούτε εδώ - απλό print, για ασφάλεια
        print(f"VBOT: Error saving order (pre-AGI stage): {e}")


def handle_new_order_turn(user_text, dialog_state, callerid):
    """
    Χειρίζεται ένα γύρο διαλόγου όταν βρισκόμαστε σε ροή 'new_order'.
    Επιστρέφει reply_text και ενημερώνει το dialog_state['order'] και
dialog_state['step'].
    """
    order = dialog_state["order"]

    # Αν μόλις μπήκαμε στη ροή
    if dialog_state["step"] is None:
        dialog_state["step"] = "ask_name"
        return "Τέλεια! Να προχωρήσουμε την παραγγελία σας. Πώς σας
λένε;"

    step = dialog_state["step"]

    if step == "ask_name":
        # ό,τι είπε ο χρήστης το θεωρούμε όνομα
        order["name"] = user_text.strip()
        dialog_state["step"] = "ask_phone"
        return f"Ευχαριστώ {order['name']}. Ποιο είναι το τηλέφωνό
σας;"

    elif step == "ask_phone":
        order["phone"] = user_text.strip()
        dialog_state["step"] = "ask_product"
        return "Τέλεια. Ποιο προϊόν θέλετε να παραγγείλετε;"

    elif step == "ask_product":
        order["product"] = user_text.strip()
        dialog_state["step"] = "ask_quantity"
        return "Πόσα τεμάχια θέλετε;"

    elif step == "ask_quantity":
        order["quantity"] = user_text.strip()
        dialog_state["step"] = "confirm"
        return (f"Να καταχωρήσω την παραγγελία στο όνομα
{order.get('name', '')}, "
                f"τηλέφωνο {order.get('phone', '')}, για
{order.get('quantity', '')} "
                f"τεμάχια του '{order.get('product', '')}'. Είναι
σωστό;")
```

```
    elif step == "confirm":
        text_norm = user_text.lower().strip()
        # πολύ απλό yes/no
        yes_words = ["ναι", "σωστά", "ακριβώς"]
        no_words  = ["όχι", "οχι", "λάθος"]

        if any(y in text_norm for y in yes_words):
            # συμπληρώνουμε ts + callerid και σώζουμε
            phone = order.get("phone", "")
            phone = re.sub(r"[^0-9]", "", phone)    # κρατάει μόνο
ψηφία

            product = order.get("product", "").strip().rstrip(".")
            quantity = order.get("quantity", "").strip().rstrip(".")

            order["phone"] = phone
            order["product"] = product
            order["quantity"] = quantity

            order["ts"] = datetime.datetime.utcnow().isoformat()
            order["callerid"] = callerid
            save_order(order)

            dialog_state["flow"] = None
            dialog_state["step"] = None
            dialog_state["order"] = {}
            return "Η παραγγελία σας καταχωρήθηκε. Σας ευχαριστούμε
πολύ."

        if any(n in text_norm for n in no_words):
            # αν πει όχι, ξαναρωτάμε ή επανακινούμε flow
            dialog_state["step"] = "ask_product"
            return "Ας το ξαναπροσπαθήσουμε. Ποιο προϊόν θέλετε να
παραγγείλετε;"

        # αν δεν καταλάβουμε ναι/όχι
        return "Δεν το κατάλαβα. Μπορείτε να μου πείτε αν είναι σωστή
η παραγγελία, ναι ή όχι;"

    # default
    return "Δεν το κατάλαβα. Μπορείτε να μου το πείτε ξανά;"


# ============ AGI helpers ============

def agi_write(cmd: str) -> str:
    sys.stdout.write(cmd + '\n')
    sys.stdout.flush()
    return sys.stdin.readline()


def agi_verbose(msg: str, lvl: int = 1):
    agi_write(f'VERBOSE "VBOT: {msg}" {lvl}')


# Διαβάζουμε AGI env μέχρι κενή γραμμή
agi_vars = {}
while True:
    line = sys.stdin.readline().strip()
    if not line:
        break
```

```
    if ':' in line:
        key, value = line.split(':', 1)
        agi_vars[key.strip()] = value.strip()

agi_verbose('enter voicebot dialog')

# ============ Whisper (global, lazy) ============

_WHISPER_MODEL = None


def get_whisper_model():
    global _WHISPER_MODEL
    if not WHISPER_AVAILABLE:
        return None
    if _WHISPER_MODEL is None:
        model_name = os.environ.get('FASTER_WHISPER_MODEL', 'small')
        device     = os.environ.get('FASTER_WHISPER_DEVICE', 'cpu')
        compute    = os.environ.get('FASTER_WHISPER_COMPUTE',
'float32')
        download   = os.environ.get('FASTER_WHISPER_DOWNLOAD',
'/var/lib/voicebot/models')
        threads    = int(os.environ.get('FASTER_WHISPER_THREADS',
'2'))
        Path(download).mkdir(parents=True, exist_ok=True)
        agi_verbose(f'Loading Whisper model={model_name},
device={device}, compute={compute}, threads={threads}')
        _WHISPER_MODEL = WhisperModel(
            model_name,
            device=device,
            compute_type=compute,
            download_root=download,
            cpu_threads=threads
        )
        agi_verbose('Whisper model constructed successfully')
    return _WHISPER_MODEL


# ============ TTS (gTTS -> sox -> GSM) ============

def tts_gtts_asterisk(text: str, rel_gsm: str, full_gsm: str,
lang='el') -> bool:
    if not GTTS_AVAILABLE:
        agi_verbose('gTTS not available')
        return False

    try:
        if isinstance(text, (list, tuple)):
            text = " ".join(map(str, text))
        elif isinstance(text, dict):
            text = json.dumps(text, ensure_ascii=False)
        elif not isinstance(text, str):
            text = str(text)
        text = (text or "").strip()
    except Exception as e:
        agi_verbose(f'TTS sanitize error -> {e}')
        text = ""
    if not text:
        text = os.environ.get('VB_FALLBACK_TEXT', FALLBACK_TEXT)

    try:
```

```
        Path(os.path.dirname(full_gsm)).mkdir(parents=True,
exist_ok=True)
    except Exception as e:
        agi_verbose(f'TTS mkdir error -> {e}')
        return False

    try:
        if os.path.exists(full_gsm) and os.path.getsize(full_gsm) >
100:
            agi_verbose(f'GSM TTS cache hit -> {full_gsm}')
            return True
    except Exception:
        pass

    tmp_mp3 = full_gsm.replace('.gsm', '.mp3')
    tmp_wav = full_gsm.replace('.gsm', '.wav')

    try:
        gTTS(text=text, lang=lang, slow=False).save(tmp_mp3)
        subprocess.run(['sox', tmp_mp3, '-r', '8000', '-c', '1',
tmp_wav], check=True, timeout=30)
        subprocess.run(['sox', tmp_wav, '-r', '8000', '-c', '1', '-
t', 'gsm', full_gsm], check=True, timeout=30)
        for f in (tmp_mp3, tmp_wav):
            try:
                if os.path.exists(f):
                    os.remove(f)
            except Exception:
                pass
        ok = os.path.exists(full_gsm) and os.path.getsize(full_gsm) >
100
        if ok:
            agi_verbose(f'GSM TTS ready -> {full_gsm}')
        else:
            agi_verbose('GSM TTS created but file too small')
        return ok
    except Exception as e:
        agi_verbose(f'GSM TTS error -> {e}')
        for f in (tmp_mp3, tmp_wav, full_gsm):
            try:
                if f and os.path.exists(f):
                    os.remove(f)
            except Exception:
                pass
        return False


# ============ Recording / STT helpers ============

def record_once(abs_noext: str, timeout_ms=15000, silence_sec=2) ->
str:
    res = agi_write(f'RECORD FILE {abs_noext} wav # {timeout_ms} 0
BEEP s={silence_sec}')
    agi_verbose(f'RECORD -> {res.strip()}')
    path = abs_noext + '.wav'
    if not os.path.exists(path) or os.path.getsize(path) == 0:
        return ''
    return path


def wav8_to_wav16(abs_wav8: str) -> str:
```

```
    out16 = abs_wav8.replace('.wav', '_16k.wav')
    try:
        subprocess.run(['sox', abs_wav8, '-r', '16000', '-b', '16',
'-c', '1', out16], check=True)
        agi_verbose(f'converted to 16k -> {out16}')
        return out16
    except Exception as e:
        agi_verbose(f'sox 16k error -> {e}')
        return ''


def _norm_el(s: str) -> str:
    s = s.lower().strip()
    s = "".join(ch for ch in unicodedata.normalize("NFD", s) if
unicodedata.category(ch) != "Mn")
    s = re.sub(r"\s+", " ", s)
    return s


def _clean_stt_text(text: str) -> str:
    t = text.strip()
    t = re.sub(r"\.{2,}$", "", t).strip()
    BAD = [
        "ελληνικη τηλεφωνικη συνομιλια",
        "ελληνικη τηλεφωνικη",
        "συνομιλια για ωραρια ραντεβου τιμες",
    ]
    t_norm = _norm_el(t)
    for frag in BAD:
        if frag in t_norm and len(t) <= 120:
            return ""
    words = t.split()
    dedup = []
    prev = None
    run = 0
    for w in words:
        if w == prev:
            run += 1
        else:
            prev = w
            run = 1
        if run <= 2:
            dedup.append(w)
    t2 = " ".join(dedup).strip()
    return t2


SYN_NORMALIZE = {
    "μουρουνελεο": "μουρουνελαιο",
    "μουρουνελαιο": "μουρουνέλαιο",
    "αλοη": "αλοη",
    "αλοη": "αλόη",
    "παραγγιλω": "παραγγειλω",
    "παραγγειλο": "παραγγειλω",
    "παραγγειλω": "παραγγείλω",
    "ωρες": "ώρες",
    "ωρα": "ώρα",
    "ωραριο": "ωράριο",
    "λειτουργιτε": "λειτουργείτε",
}
```

```
def stt_local(abs_wav8: str) -> str:
    """
    Τοπικό STT με faster-whisper σε 16k mono wav.
    """
    model = get_whisper_model()
    if model is None:
        agi_verbose('Whisper model not available')
        return ""

    abs_wav16 = wav8_to_wav16(abs_wav8)
    if not abs_wav16:
        return ""

    params = dict(
        language='el',
        beam_size=1,
        temperature=0.0,
        condition_on_previous_text=False,
        no_speech_threshold=0.6,
        log_prob_threshold=-1.0,
        compression_ratio_threshold=2.4,
        vad_filter=False,
    )
    agi_verbose(f"STT: calling transcribe(vad_filter=False) with
params={params}")
    try:
        segments, info = model.transcribe(
            abs_wav16,
            language=params["language"],
            beam_size=params["beam_size"],
            temperature=params["temperature"],

condition_on_previous_text=params["condition_on_previous_text"],
            no_speech_threshold=params["no_speech_threshold"],
            log_prob_threshold=params["log_prob_threshold"],

compression_ratio_threshold=params["compression_ratio_threshold"],
            vad_filter=params["vad_filter"],
        )
        text = "".join(seg.text for seg in segments).strip()
        text_clean = _clean_stt_text(text)
        agi_verbose(f"STT -> '{text_clean}' lang={info.language}
prob={info.language_probability:.3f} dur={info.duration:.2f}s")
        return text_clean
    except Exception as e:
        agi_verbose(f'STT transcribe error -> {e}')
        return ""


def do_stt(abs_wav8: str) -> str:
    # μόνο local faster-whisper
    return stt_local(abs_wav8)


# ============ Classic Qs fastpath ============

CLASSIC_QS = [
    (["ωρες", "λειτουργ"], "hours"),
    (["κλεισω", "ραντεβου"], "appointment"),
    (["τιμη"], "pricing"),
```

```
    (["νεα", "παραγγελι"], "new_order"),
    (["που", "πακετο"], "order_status"),
    (["ακυρωση", "παραγγελι"], "order_cancel"),
    (["προσφορες", "γνωριμια"], "offers"),
    (["πως", "αποστολη"], "shipping_info"),
    (["αντικαταβολ"], "payment_info"),
    (["τηλεφωνο", "παραγγελι"], "contact_info"),
    (["που", "καταστημα"], "location_info"),
    (["μουρουνελ"], "product_info"),
    (["προβλημα", "παραγγελια"], "complaint"),
    (["με", "ασχολειται", "εταιρ"], "company_info"),
]

# Default απαντήσεις για τα παραπάνω intents (αν δεν έρθουν από
αλλού)
TEMPLATES.setdefault("hours", "Το ωράριό μας είναι Δευτέρα–Παρασκευή
09:00–17:00.")
TEMPLATES.setdefault("appointment", "Για ραντεβού πείτε μου μέρα/ώρα
ή καλέστε στο 210 0000000.")
TEMPLATES.setdefault("pricing", "Για την τιμή χρειάζομαι το
προϊόν/πακέτο. Πείτε μου ποιο σας ενδιαφέρει.")
TEMPLATES.setdefault("new_order", "Τέλεια. Να προχωρήσουμε την
παραγγελία σας;")
TEMPLATES.setdefault("order_status", "Να ελέγξω την πορεία της
παραγγελίας σας; Έχετε αριθμό αποστολής;")
TEMPLATES.setdefault("order_cancel", "Μπορώ να ελέγξω ακύρωση. Έχετε
αριθμό παραγγελίας;")
TEMPLATES.setdefault("offers", "Έχουμε τρέχουσες προσφορές. Θέλετε να
σας ενημερώσω για κάποια κατηγορία;")
TEMPLATES.setdefault("shipping_info", "Αποστολή με courier σε 1–3
εργάσιμες. Δωρεάν άνω των 39€.")
TEMPLATES.setdefault("payment_info", "Δεχόμαστε αντικαταβολή, κάρτα
και κατάθεση.")
TEMPLATES.setdefault("contact_info", "Τηλέφωνο παραγγελιών: 210
0000000.")
TEMPLATES.setdefault("location_info", "Το κατάστημά μας είναι στο
κέντρο της Αθήνας.")
TEMPLATES.setdefault("product_info", "Πείτε μου ποιο προϊόν σας
ενδιαφέρει να σας πω λεπτομέρειες.")
TEMPLATES.setdefault("complaint", "Λυπάμαι γι’ αυτό. Πείτε μου το
θέμα για να βοηθήσω.")
TEMPLATES.setdefault("company_info", "Είμαστε η Healthy Living και
ασχολούμαστε με προϊόντα υγείας/ευεξίας.")


def classic_intent_fastpath(user_text: str):
    """
    Πολύ γρήγορος έλεγχος για “κλασικές” φράσεις.
    """
    txt = _norm_el(user_text)
    for keywords, intent in CLASSIC_QS:
        ok = True
        for kw in keywords:
            if kw not in txt:
                ok = False
                break
        if ok:
            return intent
    return None
```

```
# ============ NLU wrapper ============

def nlu_and_reply(user_text: str):
    """
    Επιστρέφει (intent, slots, reply_text).
    Προσπαθεί:
      1. fastpath (classic questions)
      2. PyTorch μοντέλο (predict)
      3. keyword fallback (nlu_greek)
      4. unknown
    """

    # 1. Fastpath για πολύ κλασικές φράσεις (ωράριο, χαιρετισμός
κτλ.)
    fast_intent = classic_intent_fastpath(user_text)
    if fast_intent:
        reply = get_response(fast_intent, confidence=1.0)
        return fast_intent, {}, reply

    # 2. PyTorch advanced (μοντέλο από train_intents.py)
    if NLU_PT_AVAILABLE and hasattr(nlu_intent_pt, "predict"):
        try:
            intent, score, slots = nlu_intent_pt.predict(user_text)
        except Exception as e:
            agi_verbose(f"NLU PT error: {e}")
            intent, score, slots = "unknown", 0.0, {}

        reply = get_response(intent, confidence=score)
        return intent, (slots or {}), reply

    # 3. Keyword-based fallback (nlu_greek.py)
    if NLU_KW_AVAILABLE:
        try:
            out = nlu_keyword(user_text)
            intent = out.get("intent", "unknown")
        except Exception as e:
            agi_verbose(f"NLU keyword error: {e}")
            intent = "unknown"

        reply = get_response(intent, confidence=1.0)
        return intent, {}, reply

    # 4. Τελείως άγνωστο
    intent = "unknown"
    reply = get_response(intent, confidence=0.0)
    return intent, {}, reply


# ============ Helpers για filenames ============

def make_rel_abs(name_noext: str):
    rel_noext = f'{REL_DIR}/{name_noext}'
    abs_noext = os.path.join(BASE_SOUNDS, rel_noext)
    return rel_noext, abs_noext


# ============ Κύρια ροή ============

ts = int(time.time())

# Welcome
```

```
agi_verbose('Starting welcome TTS (GSM format)')
welcome_rel_noext, welcome_abs_noext =
make_rel_abs(f'vb_{ts}_welcome')
welcome_rel_gsm  = welcome_rel_noext + '.gsm'
welcome_abs_gsm  = welcome_abs_noext + '.gsm'
if tts_gtts_asterisk(WELCOME_TEXT, welcome_rel_gsm, welcome_abs_gsm,
lang='el'):
    agi_write(f'EXEC Playback {welcome_rel_noext}')
else:
    agi_verbose('GSM TTS failed, using fallback vicidial-welcome')
    agi_write('EXEC Playback vicidial-welcome')

# Dialog loop
for turn in range(1, MAX_TURNS + 1):
    agi_verbose(f'DIALOG turn={turn}/{MAX_TURNS}')

    # RECORD
    rec_rel_noext, rec_abs_noext = make_rel_abs(f'vb_{ts}_t{turn}')
    user_wav8 = record_once(rec_abs_noext, timeout_ms=15000,
silence_sec=2)
    if not user_wav8:
        agi_verbose('no user audio recorded -> break')
        break

    # STT
    user_text = do_stt(user_wav8).strip()
    if not user_text or user_text == '.':
        agi_verbose('empty STT text -> fallback reply')
        reply_text = FALLBACK_TEXT
        intent = 'unknown'
        slots = {}
    else:
        agi_verbose(f'User said: {user_text[:100]}.')

        callerid = agi_vars.get("agi_callerid") or
agi_vars.get("callerid") or ""

        # Αν είμαστε ήδη μέσα σε ροή new_order → συνεχίζουμε flow
        if dialog_state["flow"] == "new_order":
            reply_text = handle_new_order_turn(user_text,
dialog_state, callerid)
            intent = "new_order"
            slots = {}
        else:
            # Κανονικό NLU
            intent, slots, reply_text = nlu_and_reply(user_text)
            agi_verbose(f'NLU -> intent={intent} | slots={slots}')

            # Αν η πρόταση ήταν "new_order", μπαίνουμε στη ροή
παραγγελίας
            if intent == "new_order":
                dialog_state["flow"] = "new_order"
                dialog_state["step"] = None
                dialog_state["order"] = {}
                reply_text = handle_new_order_turn(user_text,
dialog_state, callerid)

    # TTS Reply
    reply_rel_noext, reply_abs_noext =
make_rel_abs(f'vb_{ts}_bot_t{turn}')
    reply_rel_gsm = reply_rel_noext + '.gsm'
```

```
    reply_abs_gsm = reply_abs_noext + '.gsm'
    if tts_gtts_asterisk(reply_text, reply_rel_gsm, reply_abs_gsm,
lang='el'):
        agi_write(f'EXEC Playback {reply_rel_noext}')
    else:
        agi_verbose('TTS failed; using vm-msgsaved')
        agi_write('EXEC Playback vm-msgsaved')

# Goodbye
agi_verbose('Starting goodbye TTS (GSM format)')
good_rel_noext, good_abs_noext = make_rel_abs(f'vb_{ts}_goodbye')
good_rel_gsm = good_rel_noext + '.gsm'
good_abs_gsm = good_abs_noext + '.gsm'
if tts_gtts_asterisk(GOODBYE_TEXT, good_rel_gsm, good_abs_gsm,
lang='el'):
    agi_write(f'EXEC Playback {good_rel_noext}')
else:
    agi_verbose('Goodbye TTS failed, fallback to vm-goodbye')
    agi_write('EXEC Playback vm-goodbye')

# Cleanup whisper
def cleanup_models():
    global _WHISPER_MODEL
    if _WHISPER_MODEL is not None:
        try:
            del _WHISPER_MODEL
            _WHISPER_MODEL = None
            agi_verbose('Whisper model cleaned up')
        except Exception as e:
            agi_verbose(f'Model cleanup error: {e}')

cleanup_models()
agi_verbose('voicebot completed successfully')
sys.exit(0)
```

## ΠΑΡΑΡΤΗΜΑ Α.2 — Εκπαίδευση NLU (train_intents.py)

```
#!/opt/miniconda/envs/vbot/bin/python
# -*- coding: utf-8 -*-

"""
train_intents.py
----------------
Εκπαίδευση απλού BoW + MLP ταξινομητή προτάσεων (NLU intent
classifier)
με υποστήριξη:
- unigrams (default)
- bigrams (--use_bigrams)
- character n-grams (--use_chargrams --char_nmin 3 --char_nmax 5)

Αποθηκεύει: nlu_intent.pt, nlu_vocab.json, nlu_labels.json,
nlu_metrics.json
"""

import os, csv, json, re, unicodedata, argparse
from pathlib import Path
from typing import List, Dict, Tuple
from collections import Counter

import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import f1_score, accuracy_score

# ---------------- Paths ----------------
MODEL_DIR = Path(os.environ.get("NLU_MODEL_DIR",
"/var/lib/voicebot/models"))
MODEL_DIR.mkdir(parents=True, exist_ok=True)
MODEL_PATH   = MODEL_DIR / "nlu_intent.pt"
VOCAB_PATH   = MODEL_DIR / "nlu_vocab.json"
LABELS_PATH  = MODEL_DIR / "nlu_labels.json"
METRICS_PATH = MODEL_DIR / "nlu_metrics.json"


# ---------------- Text utils ----------------
def _normalize(s: str) -> str:
    s = s.lower().strip()
    s = "".join(ch for ch in unicodedata.normalize("NFD", s)
                if unicodedata.category(ch) != "Mn")
    s = re.sub(r"[^a-zα-ωάέίήόύώ0-9\s]", " ", s)
    s = re.sub(r"\s+", " ", s).strip()
    return s

def tokenize(s: str) -> List[str]:
    return _normalize(s).split()

def _bigrams(tokens: List[str]) -> List[str]:
    return [tokens[i] + "__" + tokens[i+1] for i in
range(len(tokens)-1)]

def chargrams(s: str, nmin=3, nmax=5) -> List[str]:
    """
    Character n-grams σε normalized string.
    Χρησιμοποιούμε prefix 'chN:' ώστε να μην μπερδεύονται με λέξεις.
    """
    s = _normalize(s).replace(" ", "_")
```

```
    feats = []
    for n in range(max(1, nmin), max(nmin, nmax)+1):
        if len(s) < n:
            continue
        for i in range(0, len(s) - n + 1):
            feats.append(f"ch{n}:{s[i:i+n]}")
    return feats


# ---------------- Model ----------------
class BoWMLP(nn.Module):
    def __init__(self, vocab_size: int, num_classes: int, hidden: int
= 128, dropout: float = 0.3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(vocab_size, hidden),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden, num_classes)
        )

    def forward(self, x):
        return self.net(x)


# ---------------- Data utils ----------------
def build_vocab(
    texts: List[str],
    min_freq: int = 1,
    use_bigrams: bool = False,
    use_chargrams: bool = False,
    char_nmin: int = 3,
    char_nmax: int = 5
) -> Dict[str, int]:
    cnt: Counter = Counter()
    for t in texts:
        toks = tokenize(t)
        cnt.update(toks)
        if use_bigrams:
            cnt.update(_bigrams(toks))
        if use_chargrams:
            cnt.update(chargrams(t, char_nmin, char_nmax))

    vocab: Dict[str, int] = {}
    for tok, c in cnt.items():
        if c >= min_freq:
            vocab[tok] = len(vocab)
    return vocab


def vectorize(
    text: str,
    vocab: Dict[str, int],
    use_bigrams: bool = False,
    use_chargrams: bool = False,
    char_nmin: int = 3,
    char_nmax: int = 5
) -> torch.Tensor:
    v = torch.zeros(len(vocab), dtype=torch.float32)
    toks = tokenize(text)
    feats = list(toks)
```

```
    if use_bigrams and len(toks) >= 2:
        feats += _bigrams(toks)
    if use_chargrams:
        feats += chargrams(text, char_nmin, char_nmax)
    for tok in feats:
        idx = vocab.get(tok)
        if idx is not None:
            v[idx] += 1.0
    return v


# ---------------- Train / Eval ----------------
def train_epoch(model, opt, loss_fn, X, y, batch=16):
    model.train()
    idx = torch.randperm(len(X))
    X, y = X[idx], y[idx]
    tot_loss = 0.0
    for i in range(0, len(X), batch):
        xb = X[i:i+batch]
        yb = y[i:i+batch]
        opt.zero_grad()
        out = model(xb)
        loss = loss_fn(out, yb)
        loss.backward()
        opt.step()
        tot_loss += loss.item() * len(xb)
    return tot_loss / len(X)


def eval_epoch(model, X, y):
    model.eval()
    if len(X) == 0:
        return 0, 0, 0, 0
    with torch.no_grad():
        out = model(X)
        probs = torch.softmax(out, dim=-1)
        preds = torch.argmax(probs, dim=-1)
    acc = accuracy_score(y, preds)
    f1 = f1_score(y, preds, average="macro")
    return acc, f1, probs, preds


# ---------------- Main ----------------
def load_csv(path: Path) -> Tuple[List[str], List[str]]:
    if not path.exists():
        return [], []
    texts, labels = [], []
    with path.open("r", encoding="utf-8") as f:
        reader = csv.DictReader(f)
        for row in reader:
            t = (row.get("text") or row.get("utterance") or
"").strip()
            y = (row.get("intent") or "").strip()
            if not t or not y:
                continue
            texts.append(t)
            labels.append(y)
    return texts, labels


def main():
```

```
    ap = argparse.ArgumentParser()
    ap.add_argument("--csv", required=True, help="training CSV")
    ap.add_argument("--val_csv", help="optional validation CSV")
    ap.add_argument("--epochs", type=int, default=40)
    ap.add_argument("--hidden", type=int, default=128)
    ap.add_argument("--dropout", type=float, default=0.3)
    ap.add_argument("--lr", type=float, default=1e-3)
    ap.add_argument("--min_freq", type=int, default=1)
    ap.add_argument("--use_bigrams", action="store_true")
    ap.add_argument("--use_chargrams", action="store_true",
help="Χρήση character n-grams")
    ap.add_argument("--char_nmin", type=int, default=3)
    ap.add_argument("--char_nmax", type=int, default=5)
    ap.add_argument("--patience", type=int, default=5)
    ap.add_argument("--weight_decay", type=float, default=1e-4)
    args = ap.parse_args()

    tr_path = Path(args.csv)
    va_path = Path(args.val_csv) if args.val_csv else None

    tr_texts, tr_labels = load_csv(tr_path)
    va_texts, va_labels = load_csv(va_path) if va_path else ([], [])

    uniq = sorted(set(tr_labels))
    label2id = {y: i for i, y in enumerate(uniq)}

    tr_y = torch.tensor([label2id[y] for y in tr_labels],
dtype=torch.long)
    va_y = torch.tensor([label2id[y] for y in va_labels],
dtype=torch.long)

    vocab = build_vocab(
        tr_texts,
        min_freq=args.min_freq,
        use_bigrams=args.use_bigrams,
        use_chargrams=args.use_chargrams,
        char_nmin=args.char_nmin,
        char_nmax=args.char_nmax,
    )

    Xtr = torch.stack([
        vectorize(t, vocab,
                  use_bigrams=args.use_bigrams,
                  use_chargrams=args.use_chargrams,
                  char_nmin=args.char_nmin, char_nmax=args.char_nmax)
        for t in tr_texts
    ])

    Xva = torch.stack([
        vectorize(t, vocab,
                  use_bigrams=args.use_bigrams,
                  use_chargrams=args.use_chargrams,
                  char_nmin=args.char_nmin, char_nmax=args.char_nmax)
        for t in va_texts
    ]) if len(va_texts) else torch.zeros((0, len(vocab)))

    model = BoWMLP(len(vocab), len(uniq), hidden=args.hidden,
dropout=args.dropout)
    opt = optim.AdamW(model.parameters(), lr=args.lr,
weight_decay=args.weight_decay)
    loss_fn = nn.CrossEntropyLoss()
```

```
    best_f1 = 0
    bad_epochs = 0
    history = []

    for ep in range(1, args.epochs + 1):
        loss = train_epoch(model, opt, loss_fn, Xtr, tr_y)
        tr_acc, tr_f1, _, _ = eval_epoch(model, Xtr, tr_y)
        va_acc, va_f1, _, _ = eval_epoch(model, Xva, va_y)
        history.append({"epoch": ep, "train_f1": tr_f1, "val_f1":
va_f1})
        print(f"[{ep:03d}] loss={loss:.4f} | train Acc={tr_acc:.3f}
F1={tr_f1:.3f} | val Acc={va_acc:.3f} F1={va_f1:.3f}")
        if va_f1 > best_f1:
            best_f1 = va_f1
            bad_epochs = 0
        else:
            bad_epochs += 1
        if bad_epochs >= args.patience:
            print(f"Early stopping at epoch {ep} (best val
F1={best_f1:.3f})")
            break

    torch.save({
        "state_dict": model.state_dict(),
        "vocab": vocab,
        "labels": uniq,
        "use_bigrams": bool(args.use_bigrams),
        "use_chargrams": bool(args.use_chargrams),
        "char_nmin": int(args.char_nmin),
        "char_nmax": int(args.char_nmax)
    }, MODEL_PATH)

    with open(VOCAB_PATH, "w", encoding="utf-8") as f:
        json.dump(vocab, f, ensure_ascii=False)

    with open(LABELS_PATH, "w", encoding="utf-8") as f:
        json.dump(uniq, f, ensure_ascii=False)

    with open(METRICS_PATH, "w", encoding="utf-8") as f:
        json.dump({
            "train": {"f1": tr_f1},
            "val": {"f1": va_f1},
            "history": history,
            "features": {
                "use_bigrams": bool(args.use_bigrams),
                "use_chargrams": bool(args.use_chargrams),
                "char_n": [args.char_nmin, args.char_nmax]
            }
        }, f, ensure_ascii=False, indent=2)

    print(f"Saved model to: {MODEL_PATH}")
    print(f"Train F1={tr_f1:.3f} | Val F1={va_f1:.3f}")


if __name__ == "__main__":
    main()
```

## ΠΑΡΑΡΤΗΜΑ Β — Web Interface (Django)

Το παρόν παράρτημα παρουσιάζει βασικά αποσπάσματα κώδικα του web περιβάλλοντος διαχείρισης που αναπτύχθηκε με χρήση του πλαισίου Django. Η διεπαφή αυτή επιτρέπει:

- Διαχείριση δεδομένων εκπαίδευσης NLU
- Εκκίνηση επανεκπαίδευσης μοντέλου
- Δοκιμή προτάσεων (intent testing)
- Προβολή καταγεγραμμένων παραγγελιών

### ΠΑΡΑΡΤΗΜΑ Β.1 — models.py

```
from django.db import models

class Example(models.Model):
    SPLITS = [
        ("train", "Train"),
        ("val", "Validation"),
    ]

    text = models.TextField(verbose_name="Κείμενο")
    intent = models.CharField(max_length=64, verbose_name="Intent")
    split = models.CharField(max_length=10, choices=SPLITS,
default="train", verbose_name="Σύνολο")
    is_active = models.BooleanField(default=True,
verbose_name="Ενεργό")
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ["-created_at"]

    def __str__(self):
        return f"[{self.split}] {self.intent}: {self.text[:40]}"
```

### ΠΑΡΑΡΤΗΜΑ Β.2 — views.py (Intent Testing & Retrain)

```
import csv
import os
import json
import subprocess
import sys
import re
import unicodedata
from collections import Counter
from pathlib import Path
```

```
import torch
import torch.nn as nn

from django.conf import settings
from django.contrib.auth.decorators import login_required,
user_passes_test
from django.db.models import Q
from django.http import HttpResponse
from django.shortcuts import render, redirect, get_object_or_404

from .models import Example
from .forms import ExampleForm


# ---- Text utils (όπως στο train_intents.py) ----
def _normalize(s: str) -> str:
    s = s.lower().strip()
    s = "".join(
        ch for ch in unicodedata.normalize("NFD", s)
        if unicodedata.category(ch) != "Mn"
    )
    s = re.sub(r"[^a-zα-ωάέίήόύώ0-9\s]", " ", s)
    s = re.sub(r"\s+", " ", s).strip()
    return s

def tokenize(s: str):
    return _normalize(s).split()

def _bigrams(tokens):
    return [tokens[i] + "__" + tokens[i+1] for i in range(len(tokens)-
1)]

def chargrams(s: str, nmin=3, nmax=5):
    s = _normalize(s).replace(" ", "_")
    feats = []
    for n in range(max(1, nmin), max(nmin, nmax) + 1):
        if len(s) < n:
            continue
        for i in range(0, len(s) - n + 1):
            feats.append(f"ch{n}:{s[i:i+n]}")
    return feats

def vectorize_for_infer(text, vocab, use_bigrams=False,
use_chargrams=False, char_nmin=3, char_nmax=5):
    v = torch.zeros(len(vocab), dtype=torch.float32)
    toks = tokenize(text)
    feats = list(toks)
    if use_bigrams and len(toks) >= 2:
```

```
        feats += _bigrams(toks)
    if use_chargrams:
        feats += chargrams(text, char_nmin, char_nmax)
    for tok in feats:
        idx = vocab.get(tok)
        if idx is not None:
            v[idx] += 1.0
    return v


# ---- Μοντέλο (ίδιο με BoWMLP) ----
class BoWMLP(nn.Module):
    def __init__(self, vocab_size: int, num_classes: int, hidden: int =
128, dropout: float = 0.3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(vocab_size, hidden),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden, num_classes)
        )

    def forward(self, x):
        return self.net(x)


# ---- Access control ----
staff_required = user_passes_test(lambda u: u.is_staff)


# ------------------------------------------------------------------
# Examples (Dataset)
# ------------------------------------------------------------------
@login_required
@staff_required
def examples_list(request):
    """
    List dataset examples with filters:
      - q (search in text/intent)
      - intent (exact match)
      - split (train/val)
      - active (1/0) optional, default is only active=True
    """
    q = (request.GET.get("q") or "").strip()
    split_filter  = (request.GET.get("split") or "").strip()
    intent_filter = (request.GET.get("intent") or "").strip()
    active_filter = (request.GET.get("active") or "").strip()  # "",
"1", "0"
```

```
    qs = Example.objects.all()

    # Active filter: default = only active
    if active_filter in ("1", "0"):
        qs = qs.filter(is_active=(active_filter == "1"))
    else:
        qs = qs.filter(is_active=True)

    if split_filter in ("train", "val"):
        qs = qs.filter(split=split_filter)

    if intent_filter:
        qs = qs.filter(intent=intent_filter)

    if q:
        qs = qs.filter(Q(text__icontains=q) | Q(intent__icontains=q))

    qs = qs.order_by("intent", "id")[:500]

    # dropdown intents (από ενεργές εγγραφές για καθαρό menu)
    intents = (
        Example.objects.filter(is_active=True)
        .values_list("intent", flat=True)
        .distinct()
        .order_by("intent")
    )

    return render(request, "nluui/examples_list.html", {
        "examples": qs,
        "q": q,
        "split": split_filter,
        "intent_filter": intent_filter,
        "active_filter": active_filter,
        "intents": intents,
    })


@login_required
@staff_required
def examples_new(request):
    if request.method == "POST":
        form = ExampleForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect("nluui:list")
    else:
        form = ExampleForm()
    return render(request, "nluui/examples_form.html", {
        "form": form,
```

```
        "title": "Νέα εγγραφή",
    })


@login_required
@staff_required
def examples_edit(request, pk):
    obj = get_object_or_404(Example, pk=pk)
    if request.method == "POST":
        form = ExampleForm(request.POST, instance=obj)
        if form.is_valid():
            form.save()
            return redirect("nluui:list")
    else:
        form = ExampleForm(instance=obj)
    return render(request, "nluui/examples_form.html", {
        "form": form,
        "title": "Επεξεργασία εγγραφής",
    })


@login_required
@staff_required
def examples_delete(request, pk):
    obj = get_object_or_404(Example, pk=pk)
    if request.method == "POST":
        obj.delete()
        return redirect("nluui:list")
    return render(request, "nluui/examples_delete.html", {"obj": obj})


@login_required
@staff_required
def export_csv(request):
    """
    Γράφει 2 αρχεία CSV:
      - nlu_greek_train.csv  (text,intent)
      - nlu_greek_val.csv    (text,intent)
    στο NLU_DATA_DIR
    """
    out_dir = settings.NLU_DATA_DIR
    os.makedirs(out_dir, exist_ok=True)

    train_path = os.path.join(out_dir, settings.NLU_TRAIN_CSV)
    val_path   = os.path.join(out_dir, settings.NLU_VAL_CSV)

    train_qs = Example.objects.filter(is_active=True,
split="train").order_by("intent", "id")
```

```
    val_qs   = Example.objects.filter(is_active=True,
split="val").order_by("intent", "id")

    def write_csv(path, qs):
        with open(path, "w", newline="", encoding="utf-8") as f:
            w = csv.writer(f)
            w.writerow(["text", "intent"])
            for e in qs:
                text = (e.text or "").replace("\r\n", "\n").strip()
                intent = (e.intent or "").strip()
                if text and intent:
                    w.writerow([text, intent])

    write_csv(train_path, train_qs)
    write_csv(val_path, val_qs)

    msg = f"Έγινε export:\n- {train_path}\n- {val_path}\n"
    return HttpResponse(msg, content_type="text/plain; charset=utf-8")


@login_required
@staff_required
def train_model(request):
    """
    1) Κάνει export τα train/val CSV (όπως το export_csv)
    2) Τρέχει το train_intents.py με αυτά τα CSV
    3) Δείχνει τα logs (stdout/stderr)
    """
    out_dir = settings.NLU_DATA_DIR
    os.makedirs(out_dir, exist_ok=True)

    train_path = os.path.join(out_dir, settings.NLU_TRAIN_CSV)
    val_path   = os.path.join(out_dir, settings.NLU_VAL_CSV)

    # 1) Export από DB
    train_qs = Example.objects.filter(is_active=True,
split="train").order_by("intent", "id")
    val_qs   = Example.objects.filter(is_active=True,
split="val").order_by("intent", "id")

    if not train_qs.exists():
        return HttpResponse("Δεν υπάρχουν train examples
(split='train').", content_type="text/plain; charset=utf-8")

    def write_csv(path, qs):
        with open(path, "w", newline="", encoding="utf-8") as f:
            w = csv.writer(f)
            w.writerow(["text", "intent"])
            for e in qs:
```

```
                text = (e.text or "").replace("\r\n", "\n").strip()
                intent = (e.intent or "").strip()
                if text and intent:
                    w.writerow([text, intent])

    write_csv(train_path, train_qs)
    use_val = False
    if val_qs.exists():
        write_csv(val_path, val_qs)
        use_val = True

    # 2) Run train_intents.py
    train_script = getattr(settings, "NLU_TRAIN_SCRIPT", None)
    if not train_script or not os.path.exists(train_script):
        return HttpResponse(
            f"Το NLU_TRAIN_SCRIPT δεν βρέθηκε: {train_script}",
            content_type="text/plain; charset=utf-8",
        )

    env = os.environ.copy()
    env["NLU_MODEL_DIR"] = getattr(settings, "NLU_MODEL_DIR",
"/var/lib/voicebot/models")

    cmd = [
        sys.executable,
        train_script,
        "--csv", train_path,
        "--epochs", "40",
    ]
    if use_val:
        cmd += ["--val_csv", val_path]

    proc = subprocess.run(
        cmd,
        env=env,
        text=True,
        capture_output=True,
    )

    output = []
    output.append(f"Running: {' '.join(cmd)}\n")
    output.append(f"NLU_MODEL_DIR={env['NLU_MODEL_DIR']}\n")
    output.append("\n--- STDOUT ---\n")
    output.append(proc.stdout or "")
    output.append("\n--- STDERR ---\n")
    output.append(proc.stderr or "")
    output.append(f"\nReturn code: {proc.returncode}\n")
```

```
    return HttpResponse("".join(output), content_type="text/plain;
charset=utf-8")


# ---------------------------------------------------------------
# Test Intent (Playground)
# ---------------------------------------------------------------
@login_required
@staff_required
def test_intent(request):
    """
    Playground: παίρνει κείμενο από φόρμα
    και κάνει πρόβλεψη intent χρησιμοποιώντας
    το checkpoint nlu_intent.pt (state_dict + vocab + labels +
features).
    """
    result = None
    error = None

    if request.method == "POST":
        text = (request.POST.get("text") or "").strip()

        if text:
            try:
                model_dir = getattr(settings, "NLU_MODEL_DIR",
"/var/lib/voicebot/models")
                model_path = os.path.join(model_dir, "nlu_intent.pt")

                ckpt = torch.load(model_path,
map_location=torch.device("cpu"))

                state_dict = ckpt["state_dict"]
                vocab = ckpt["vocab"]
                labels = ckpt["labels"]
                use_bigrams = ckpt.get("use_bigrams", False)
                use_chargrams = ckpt.get("use_chargrams", False)
                char_nmin = ckpt.get("char_nmin", 3)
                char_nmax = ckpt.get("char_nmax", 5)

                first_w = state_dict["net.0.weight"]
                hidden_dim = first_w.shape[0]

                model = BoWMLP(
                    vocab_size=len(vocab),
                    num_classes=len(labels),
                    hidden=hidden_dim,
                    dropout=0.3,
                )
                model.load_state_dict(state_dict)
```

```
                model.eval()

                x = vectorize_for_infer(
                    text,
                    vocab,
                    use_bigrams=use_bigrams,
                    use_chargrams=use_chargrams,
                    char_nmin=char_nmin,
                    char_nmax=char_nmax,
                ).unsqueeze(0)

                with torch.no_grad():
                    logits = model(x)
                    probs = torch.softmax(logits, dim=1)[0]
                    pred_index = torch.argmax(probs).item()
                    pred_intent = labels[pred_index]
                    confidence = float(probs[pred_index].item())

                result = {
                    "text": text,
                    "intent": pred_intent,
                    "confidence": round(confidence, 4),
                    "scores": {labels[i]: float(probs[i]) for i in
range(len(labels))}
                }

            except Exception as e:
                error = str(e)

    return render(request, "nluui/test_intent.html", {
        "result": result,
        "error": error,
    })


# ------------------------------------------------------------------
# Orders (Voicebot)
# ------------------------------------------------------------------
ORDERS_CSV_PATH = Path("/var/lib/voicebot/orders.csv")


@login_required
@staff_required
def orders_list(request):
    headers = ["ts", "callerid", "name", "phone", "product",
"quantity"]
    orders = []
    error_message = ""
```

```
    if ORDERS_CSV_PATH.exists():
        try:
            with ORDERS_CSV_PATH.open("r", encoding="utf-8") as f:
                reader = csv.DictReader(f)
                for row in reader:
                    row_values = [(row.get(h, "") or "") for h in
headers]
                    orders.append(row_values)
        except Exception as e:
            error_message = f"Σφάλμα κατά την ανάγνωση του orders.csv:
{e}"
    else:
        error_message = "Δεν βρέθηκε το αρχείο orders.csv."
    return render(request, "nluui/orders_list.html", {
        "headers": headers,
        "orders": orders,
        "error_message": error_message,
    })
```

**ΠΑΡΑΡΤΗΜΑ Β.3 — admin.py**

```
from django.contrib import admin
from .models import Example

@admin.register(Example)
class ExampleAdmin(admin.ModelAdmin):
    list_display = ("id", "intent", "split", "is_active", "created_at")
    list_filter = ("split", "intent", "is_active")
    search_fields = ("text", "intent")
    ordering = ("-created_at",)
```

**ΠΑΡΑΡΤΗΜΑ Β.4 — URL Configuration (urls.py)**

```
from django.urls import path
from . import views

app_name = "nluui"

urlpatterns = [
    path("", views.examples_list, name="list"),
    path("new/", views.examples_new, name="new"),
    path("<int:pk>/edit/", views.examples_edit, name="edit"),
    path("<int:pk>/delete/", views.examples_delete, name="delete"),
    path("export/", views.export_csv, name="export"),
    path("train/", views.train_model, name="train"),
    path("test/", views.test_intent, name="test"),
    path("orders/", views.orders_list, name="orders_list"),
]
```