



Σχολή Θετικών Επιστημών και Τεχνολογίας
Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά
Συστήματα

Διπλωματική Εργασία

Αυτόματη Δημιουργία Ερωτήσεων από Εκπαιδευτικό Κείμενο με
Χρήση Τεχνητής Νοημοσύνης και Παραγωγή Διαδραστικών Κουίζ
H5P

Απόστολος Καλύβας

ΑΜ: 159283

Επιβλέπων: Γεώργιος Μαυρομμάτης

Συν-Επιβλέπων: Μιχαήλ Βασιλακόπουλος

Πάτρα, Ιούνιος 2026

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



Αυτόματη Δημιουργία Ερωτήσεων από Εκπαιδευτικό Κείμενο με
Χρήση Τεχνητής Νοημοσύνης και Παραγωγή Διαδραστικών Κουίζ
H5P

Απόστολος Καλύβας

Επιτροπή Επίβλεψης Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:

Γεώργιος Μαυρομμάτης

Μέλος ΣΕΠ ΕΑΠ

Συν-Επιβλέπων Καθηγητής:

Μιχαήλ Βασιλακόπουλος

Μέλος ΣΕΠ ΕΑΠ

Πάτρα, Ιούνιος 2026

Θα ήθελα να εκφράσω τις ιδιαίτερες μου ευχαριστίες στον καθηγητή κ. Γεώργιο Μαυρομμάτη για την καθοδήγηση και την υποστήριξη κατά την διάρκεια της εκπόνησης της παρούσας εργασίας.

Περίληψη

Η παρούσα Διπλωματική Εργασία (ΔΕ) επικεντρώνεται στην αυτόματη δημιουργία ερωτήσεων από εκπαιδευτικό κυρίως κείμενο με αξιοποίηση τεχνικών επεξεργασίας φυσικής γλώσσας NLP και μοντέλων βαθιάς μάθησης με απώτερο σκοπό την παραγωγή διαδραστικών κουίζ σε μορφή H5P η οποία είναι συμβατή με ευρέως χρησιμοποιούμενα συστήματα διαχείρισης μάθησης, όπως το Moodle.

Αναπτύχθηκε εφαρμογή σε γλώσσα προγραμματισμού Python που υλοποιεί ένα αυτοματοποιημένο αγωγό (pipeline) επεξεργασίας κειμένου με φόρτωση κειμένου txt και pdf, ανίχνευση γλώσσας, γλωσσολογική ανάλυση μέσω της βιβλιοθήκης Stanza, και ολοκληρώνεται με την παραγωγή quiz μορφής Single Choice και Fill in the blanks, χρησιμοποιώντας προεκπαιδευμένο γλωσσικό μοντέλο T5. Για εναλλακτικές απαντήσεις (distractors) στις Single Choice ερωτήσεις, χρησιμοποιούνται σημασιολογικά συναφείς όροι, που αντλούνται από το WordNet, Wikipedia API και συμπληρωματικά από υπάρχοντες όρους του κειμένου.

Αναλύονται οι βασικές έννοιες του NLP, η αρχιτεκτονική Transformer, το μοντέλο T5, οι σύγχρονες προσεγγίσεις AQG, οι λεξικοσημασιολογικοί πόροι που χρησιμοποιούνται για distractors και το πρότυπο H5P ως μηχανισμός διάθεσης επαναχρησιμοποιήσιμου εκπαιδευτικού περιεχομένου.

Για την αξιολόγηση της ποιότητας των παραγόμενων ερωτήσεων εφαρμόζεται μετρική ROUGE, ενώ η εφαρμογή διαθέτει γραφικό περιβάλλον χρήστη υλοποιημένο μέσω της βιβλιοθήκης Gradio της Python

Λέξεις – Κλειδιά

Αυτόματη Δημιουργία Ερωτήσεων, Επεξεργασία Φυσικής Γλώσσας, T5, Stanza. H5P, Moodle

Automatic Question Generation from Educational Texts with the use of Artificial Intelligence and Production of Interactive H5P Quizzes

Apostolos Kalyvas

Abstract

This Thesis focuses on the automatic generation of questions from primarily educational text by leveraging Natural Language Processing (NLP) techniques and deep learning models, with the ultimate goal of producing interactive quizzes in H5P format, which is compatible with widely used learning management systems such as Moodle.

An application was developed in the Python programming language that implements an automated text-processing pipeline, including the loading of .txt and .pdf files, language detection, linguistic analysis through the Stanza library and concluding in the production of Single Choice and Fill in the Blanks quizzes, using a pre-trained language model T5. For alternative answer options (distractors) in Single Choice quizzes, semantically related terms are used, drawn from WordNet, the Wikipedia API and from existing terms in the text itself.

The thesis analyzes the core concepts of NLP, the Transformer architecture, the T5 model, contemporary AQG approaches, the lexical-semantic resources used for distractor generation, and the H5P standard as a mechanism for delivering reusable educational content.

For the evaluation of the quality of the generated questions, the ROUGE metric is applied, while the application also includes a graphical user interface implemented through Python's Gradio library.

Keywords

Automatic Question Generation, Natural Language Processing, T5, Stanza, H5P, Moodle

Περιεχόμενα

Περίληψη.....	v
Abstract	vi
Περιεχόμενα	vii
Κατάλογος Εικόνων	x
Κατάλογος Πινάκων	xi
Κατάλογος Διαγραμμάτων.....	xii
Συντομογραφίες & Ακρωνύμια.....	xiii
1. Εισαγωγή.....	1
1.1 Παρουσίαση Ανάγκης	1
1.2 Κίνητρο και Σκοπός Υλοποίησης	1
1.3 Στόχοι της Εργασίας.....	2
2. Θεωρητικό Υπόβαθρο.....	3
2.1 Επεξεργασία Φυσικής Γλώσσας (NLP)	3
2.1.1 Tokenization και Χωρισμός Προτάσεων	4
2.1.2 Ανάλυση μερών του λόγου (POS tagging)	5
2.1.3 Λημματοποίηση (LEM)	5
2.1.4 Συντακτική Ανάλυση Εξαρτήσεων (Dependency Parsing)	6
2.1.5 Αναγνώριση Ονομαστικών Οντοτήτων (NER).....	7
2.1.6 Η Βιβλιοθήκη Stanza	7
2.1.7 Η Βιβλιοθήκη Stanza σε σύγκριση με την spaCy	8
2.2 Transformers	9
2.2.1 Μηχανισμός Self Attention	9
2.2.2 Encoder-Decoder Αρχιτεκτονική	9
2.2.3 Bidirectional Encoder Representations from Transformers (BERT).....	10
2.2.4 Bidirectional and Auto-Regressive Transformers (BART)	11
2.2.5 GreekBERT.....	11
2.2.6 Text-to-Text Transfer Transformer (T5).....	12
2.2.7 Το οικοσύστημα Hugging Face Transformers.....	12
2.3 Μοντέλα T5.....	13
2.3.1 mT5	13
2.3.2 iarfmoose/t5-base-question-generator	13
2.3.3 valhalla/t5-base-qg-hl.....	14
2.3.4 Αιτιολόγηση επιλογής μοντέλου T5	15
2.4 Αυτόματη Δημιουργία Ερωτήσεων (AQG)	15
2.4.1 Rule-based προσεγγίσεις.....	16
2.4.2 Neural προσεγγίσεις — seq2seq και fine-tuned Transformers	17
2.4.3 Υβριδικές προσεγγίσεις.....	18
2.4.4 Σύγκριση με την παρούσα εργασία	18
2.5 Παραγωγή Distractors	19
2.5.1 WordNet.....	20
2.5.2 Wikipedia API.....	20
2.5.3 Cross-sentence entity pooling	21
2.6 Νευρωνική Μηχανική Μετάφραση (NMT)	22

2.6.1 MarianMT	22
2.6.2 Μοντέλα Helsinki-NLP	23
2.7 Βελτίωση Παραγόμενου Κειμένου	23
2.7.1 LLM-based post-processing	24
2.7.2 Επιλογή LLM	24
2.7.3 Επιλογή μοντέλου Groq	25
2.8 Learning Management Systems (LMS)	25
2.8.1 Moodle	25
2.8.2 H5P	26
2.8.3 H5P.Single Choice Set	28
2.8.4 H5P.Fill In The Blanks	28
2.8.5 Lumi Desktop	29
2.9 Μετρικές Αξιολόγησης Παραγόμενου Κειμένου	29
2.9.1 Bilingual Evaluation Understudy (BLEU)	30
2.9.2 Recall-Oriented Understudy for Gisting Evaluation (ROUGE)	31
3. Σχεδίαση	31
3.1 Αρχιτεκτονική	31
3.1.1 Φιλοσοφία σχεδιασμού	32
3.1.2 Τα 4 Στρώματα Οργάνωσης	33
3.1.3 Σειριακή Ροή Εκτέλεσης	35
3.2 Είσοδος και Προετοιμασία	38
3.2.1 Σημεία Εισόδου	38
3.2.2 Ενορχήστρωση μέσω «run_pipeline»	40
3.2.3 Φόρτωση και καθαρισμός κειμένου	42
3.2.4 Ανίχνευση Γλώσσας	43
3.2.5 Μετάφραση Εισόδου (el→en)	44
3.2.6 Anchor Dictionary	46
3.3 Γλωσσολογική Ανάλυση και Επιλογή	49
3.3.1 Stanza Pipeline και Φιλτράρισμα Προτάσεων	49
3.3.2 Επιλογή Απαντήσεων	51
3.3.3 Σύνθεση Fill in the blanks items	54
3.3.4 Sliding Window – Προετοιμασία για Single Choice items	56
3.4 Παραγωγή Ερωτήσεων	58
3.4.1 T5 και Highlight Format	58
3.4.2 Παραγωγή Distractors	58
3.4.3 Τελική Σύνθεση Single Choice items	60
3.5 Ποιοτικός Έλεγχος με LLM	62
3.5.1 Επιλογή Groq(Llama-3.1)	62
3.5.2 Εφαρμογή Τεσσάρων Βημάτων Ελέγχου	62
3.5.3 Έλεγχος Κατόπιν Μετάφρασης	64
3.6 Έξοδος και παράδοση	65
3.6.1 Μετάφραση Εξόδου (en→el)	65
3.6.2 Έλεγχοι πριν την Εξαγωγή	68
3.6.3 Εξαγωγή H5P	69
3.7 Αξιολόγηση	71
3.7.1 Αρχιτεκτονική ROUGE	71

3.7.2 Αποτελέσματα και παρατηρήσεις	72
4. Υλοποίηση και Λειτουργία	74
4.1 Λειτουργία μέσω Web UI	74
4.2 Παραμετροποίηση	75
4.3 Παραγωγή και download H5P αρχείων	77
4.4 Λειτουργία μέσω CLI	80
4.5 Χρήση σε Lumi και Moodle	81
5. Συμπεράσματα	86
5.1 Σύνοψη	86
5.2 Περιορισμοί και αποτίμηση	87
5.3 Προτάσεις για μελλοντική εργασία	88
Βιβλιογραφία	89
Παράρτημα Α: «Κώδικας»	94

Κατάλογος Εικόνων

Εικόνα 1 Διάρθρωση του πυρήνα του συστήματος.....	33
Εικόνα 2 Η Gradio διεπαφή σε λειτουργία.....	39
Εικόνα 3 Το Command line σε λειτουργία.....	39
Εικόνα 4 Κεντρική συνάρτησης ενορχήστρωσης.....	40
Εικόνα 5 Entry points της app.py.....	41
Εικόνα 6 Entry points της main.py.....	42
Εικόνα 7 Φόρτωση .txt αρχείου.....	42
Εικόνα 8 Καθαρισμός σελίδων πριν την ανάγνωση αρχείου.....	43
Εικόνα 9 qg/lang_detect.py.....	44
Εικόνα 10 Lazy loading MarianMT.....	46
Εικόνα 11 Καθαρισμός με regex.....	46
Εικόνα 12 Χτίσιμο Anchor Dict.....	47
Εικόνα 13 Εφαρμογή anchor dictionary.....	48
Εικόνα 14 Lazy load Stanza.....	49
Εικόνα 15 Διαχωρισμός και φιλτράρισμα προτάσεων.....	50
Εικόνα 16 Προτεραιότητα επιλογής υποψήφιας απάντησης.....	52
Εικόνα 17 Χειρισμός πολυλεκτικών οντοτήτων.....	53
Εικόνα 18 Σύνθεση fill in the blanks.....	54
Εικόνα 19 Σύνθεση sliding window.....	57
Εικόνα 20 Global token pool.....	60
Εικόνα 21 Καθαρισμός artifacts.....	61
Εικόνα 22 Έλεγχος circular question.....	62
Εικόνα 23 Groq prompt βελτίωσης Single Choice.....	65
Εικόνα 24 Μετάφραση Single Choice (en->el).....	67
Εικόνα 25 Διόρθωση brackets.....	68
Εικόνα 26 Αρχείο h5p ως zip.....	69
Εικόνα 27 Content Single Choice.....	70
Εικόνα 28 Content Fill in the blanks.....	70
Εικόνα 29 Unicode aware tokenizer.....	71
Εικόνα 30 Επιλογές εισόδου αρχείου/κειμένου.....	75
Εικόνα 31 Παραμετροποίηση.....	76
Εικόνα 32 Ρυθμίσεις ερωτήσεων.....	77
Εικόνα 33 Εμφάνιση παραγωγής.....	78
Εικόνα 34 Εμφάνιση παραγωγής.....	79
Εικόνα 35 Εμφάνιση παραγωγής.....	80
Εικόνα 36 Έλεγχος config.py.....	80
Εικόνα 37 Preview αρχείου single choice.....	81
Εικόνα 38 Preview αρχείου fill in the blanks.....	82
Εικόνα 39 Επεξεργασία αρχείου .h5p.....	83
Εικόνα 40 Επιλογή activity σε Moodle course.....	84
Εικόνα 41 Προεπισκόπηση σε Moodle course.....	85

Κατάλογος Πινάκων

Πίνακας 1 Σειριακή Ροή Εκτέλεσης	38
Πίνακας 2 Ανάλυση Stanza.....	51
Πίνακας 3 Επιλογή απάντησης	51
Πίνακας 4 Μασκάρισμα επαναλαμβανόμενων εμφανίσεων.....	55
Πίνακας 5 Κείμενα εφαρμογής ROUGE	72
Πίνακας 6 Αποτελέσματα ROUGE L	73
Πίνακας 7 Μέσος όρος ανά γλώσσα	74

Κατάλογος Διαγραμμάτων

Διάγραμμα 1 Τα τέσσερα στρώματα του αγωγού.....	34
Διάγραμμα 2 Δύο σημεία εισόδου καλούν την ίδια συνάρτηση ενορχήστρωσης	41
Διάγραμμα 3 Προσέγγιση «sandwich»	45
Διάγραμμα 4 Ροή λειτουργίας anchor dictionary.....	48
Διάγραμμα 5 Εκμετάλλευση «answer».....	56
Διάγραμμα 6 Χρήση sliding window	57
Διάγραμμα 7 Επιλογή κατάλληλου distractor.....	59
Διάγραμμα 8 Έλεγχος Single Choice items(en).....	63
Διάγραμμα 9 Μετάφραση Fill in the blanks (en->el)	66
Διάγραμμα 10 Δομή αρχείου .h5p	69
Διάγραμμα 11 Γράφημα ROUGE L.....	73

Συντομογραφίες & Ακρωνύμια

AI	Artificial Intelligence
API	Application Programming Interface
AQG	Automatic Question Generation
BART	Bidirectional and Auto-Regressive Transformers
BERT	Bidirectional Encoder Representations from Transformers
BIOES	Begin, Inside, Outside, End, Single
BLEU	Bilingual Evaluation Understudy
CLI	Command Line Interface
CPU	Central Processing Unit
GPE	Geo-Political Entity
GPU	Graphics Processing Unit
H5P	HTML5 Package
HTML	HyperText Markup Language
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LLM	Large Language Model
LMS	Learning Management System
LOC	Location
LSTM	Long Short-Term Memory
ML	Machine Learning
MarianMT	Marian Machine Translation
MWT	Multi-Word Token
NER	Named Entity Recognition
NLP	Natural Language Processing
NMT	Neural Machine Translation
NORP	Nationalities, Religious or Political groups
ORG	Organization (κατηγορία NER)
POS	Part Of Speech (Μέρος του Λόγου)
RNN	Recurrent Neural Network (Αναδρομικό Νευρωνικό Δίκτυο)

ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SC	Single Choice
T5	Text-to-Text Transfer Transformer
UD	Universal Dependencies
UI	User Interface
UPOS	Universal Part-Of-Speech
UTF	Unicode Transformation Format

1. Εισαγωγή

1.1 Παρουσίαση Ανάγκης

Η αξιολόγηση της γνώσης αποτελεί αναπόσπαστο κομμάτι της εκπαιδευτικής διαδικασίας είτε πραγματοποιείται σε παραδοσιακό περιβάλλον είτε σε ψηφιακές πλατφόρμες. Ειδικά σε περιβάλλοντα ηλεκτρονικής μάθησης, οι ερωτήσεις πολλαπλής επιλογής και οι ασκήσεις συμπλήρωσης κενού χρησιμοποιούνται ευρέως, επειδή υποστηρίζουν ταχεία ανατροφοδότηση, εύκολη βαθμολόγηση και προσαρμογή σε διαφορετικά μαθησιακά σενάρια.

Παρά τη χρησιμότητά τους, η χειροκίνητη δημιουργία τέτοιων ερωτήσεων παραμένει επίπονη διαδικασία. Ο εκπαιδευτικός πρέπει να επιλέξει το κατάλληλο απόσπασμα, να εντοπίσει την πληροφορία-στόχο, να διατυπώσει φυσική ερώτηση και να κατασκευάσει πειστικές αλλά λανθασμένες εναλλακτικές απαντήσεις. Η διαδικασία αυτή απαιτεί χρόνο, γνωστική προσπάθεια και εμπειρία, ιδίως όταν ζητούνται πολλές ερωτήσεις για μεγάλο όγκο υλικού.

Η ανάπτυξη των εργαλείων αυτόματης δημιουργίας ερωτήσεων επιχειρεί να συνδράμει στην εν λόγω ανάγκη και μέσω της αξιοποίησης τεχνικών της Τεχνητής Νοημοσύνης για την αυτόματη παραγωγή ερωτήσεων από κείμενο περιορίζεται σημαντικά ο φόρτος εργασίας των εκπαιδευτικών.

Παράλληλα, η εμφάνιση ανοιχτών προτύπων για διαδραστικό εκπαιδευτικό περιεχόμενο, όπως το H5P, παρέχει ένα ισχυρό πλαίσιο για τη δημιουργία, κοινή χρήση και ενσωμάτωση δραστηριοτήτων αξιολόγησης σε Συστήματα Διαχείρισης Μάθησης.

1.2 Κίνητρο και Σκοπός Υλοποίησης

Το κίνητρο για την εκπόνηση της παρούσας εργασίας απορρέει από το ενδιαφέρον για τη διερεύνηση των δυνατοτήτων Τεχνητής Νοημοσύνης στην εκπαιδευτική τεχνολογία και ειδικότερα από την πρακτική μελέτη τεχνικών Επεξεργασίας Φυσικής Γλώσσας. Παρά την ύπαρξη εμπορικών πλατφόρμων που αξιοποιούν την τεχνητή νοημοσύνη για τη μετατροπή εκπαιδευτικού υλικού σε διαδραστικές δραστηριότητες, η παρούσα εργασία δεν αποσκοπεί στην ανάπτυξη ανταγωνιστικού εμπορικού προϊόντος.

Σκοπός είναι η ανάπτυξη μιας πρακτικής εφαρμογής που να είναι χρήσιμη τόσο για εκπαιδευτικούς χωρίς τεχνικές γνώσεις, μέσω του γραφικού περιβάλλοντος Gradio, όσο και για developers που επιθυμούν να ενσωματώσουν την τεχνολογία στα δικά τους συστήματα μέσω της διεπαφής γραμμής εντολών. Επιπλέον, η εφαρμογή σχεδιάστηκε με γνώμονα την επεκτασιμότητα, τόσο ως προς τις γλώσσες (με ήδη ενσωματωμένη υποδομή για ελληνικά) όσο και ως προς τους τύπους ερωτήσεων.

1.3 Στόχοι της Εργασίας

Οι βασικοί στόχοι της παρούσας εργασίας είναι οι εξής:

- Ανάπτυξη ικανού αυτοματοποιημένου αγωγού επεξεργασίας κειμένου (pipeline) για την εισαγωγή κειμένου και αρχείων txt και pdf για την παραγωγή ερωτήσεων συμπλήρωσης κενού (fill in the blank) και ερωτήσεις μονής επιλογής (single choice), καθώς αποτελούν ορισμένες από τις βασικές εκδοχές που υποστηρίζει το H5P.
- Ενσωμάτωση γλωσσολογικής ανάλυσης με χρήση Stanza ώστε να ορίζονται κατάλληλοι υποψήφιοι όροι ως απαντήσεις.
- Αξιοποίηση προεκπαιδευμένου μοντέλου T5 για παραγωγή ερωτήσεων φυσικής γλώσσας
- Σχεδίαση μηχανισμού παραγωγής συναφών distractors με αλγοριθμικά φίλτρα, χρήση WordNet και Wikipedia API .
- Μετατροπή των αποτελεσμάτων σε έγκυρα αρχεία H5P για άμεση ενσωμάτωση σε LMS όπως το Moodle
- Ανάπτυξη φιλικού γραφικού περιβάλλοντος μέσω της βιβλιοθήκης Gradio για άμεση χρήση χωρίς γνώσεις προγραμματισμού
- Αποτίμηση της ποιότητας του συστήματος με αυτόματες μετρικές και ποιοτική συζήτηση περιορισμών.

2. Θεωρητικό Υπόβαθρο

2.1 Επεξεργασία Φυσικής Γλώσσας (NLP)

Η Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing - NLP) αποτελεί κλάδο της Τεχνητής Νοημοσύνης που εξετάζει πώς τα υπολογιστικά συστήματα μπορούν να αναλύουν, αναπαριστούν και να παράγουν ανθρώπινη γλώσσα. Στο σύγχρονο περιβάλλον, το NLP δεν περιορίζεται σε απλές εργασίες προεπεξεργασίας, αλλά περιλαμβάνει πολυεπίπεδες διαδικασίες όπως η συντακτική ανάλυση, η σημασιολογική ερμηνεία, η αναγνώριση οντοτήτων και η παραγωγή φυσικής γλώσσας.

Η εξέλιξη του πεδίου έχει μετατοπιστεί από τα παραδοσιακά συστήματα που βασίζονταν σε αυστηρούς γλωσσικούς κανόνες προς τις προηγμένες αρχιτεκτονικές Βαθιάς Μάθησης (Deep Learning). Τα σύγχρονα μοντέλα NLP, όπως οι Transformers (Vaswani et al., 2017) βασίζονται στη στατιστική ανάλυση τεράστιων σωμάτων κειμένων για την εκμάθηση διανυσματικών αναπαραστάσεων των λέξεων. Μέσω αυτών των αναπαραστάσεων, οι λέξεις μετατρέπονται σε μαθηματικά μεγέθη σε έναν πολυδιάστατο χώρο, επιτρέποντας στο σύστημα να συλλαμβάνει περίπλοκες σημασιολογικές σχέσεις (Manning & Schütze, 1999).

Στην περίπτωση της αυτόματης δημιουργίας ερωτήσεων, το NLP λειτουργεί ως ο βασικός μηχανισμός που μετατρέπει το ακατέργαστο κείμενο σε δομημένη πληροφορία κατάλληλη για λήψη αποφάσεων. Το σύστημα πρέπει να εντοπίσει ποια τμήματα του κειμένου φέρουν εκπαιδευτική αξία, ποια μπορούν να αποτελέσουν απάντηση, πώς θα παραχθεί αντίστοιχη ερώτηση και πώς θα δημιουργηθούν πειστικές εναλλακτικές επιλογές. Επομένως, η ποιότητα της παραγόμενης δραστηριότητας αξιολόγησης εξαρτάται σε μεγάλο βαθμό από την ποιότητα της γλωσσολογικής ανάλυσης που προηγείται.

Η παρούσα εργασία ακολουθεί λογική αγωγού επεξεργασίας, όπου κάθε στάδιο του NLP τροφοδοτεί το επόμενο. Το tokenization και ο χωρισμός προτάσεων αποτελούν τη βάση της αναπαράστασης, το POS tagging και το NER προσθέτουν γραμματική και σημασιολογική πληροφορία, ενώ στη συνέχεια οι πληροφορίες αυτές αξιοποιούνται από τη λογική επιλογής απάντησης, τη δημιουργία distractors και το μοντέλο T5 για την τελική παραγωγή ερώτησης.

2.1.1 Tokenization και Χωρισμός Προτάσεων

Το tokenization αποτελεί τη διαδικασία διάσπασης του κειμένου σε υπολογιστικά διαχειρίσιμες μονάδες. Οι μονάδες αυτές ονομάζονται tokens και απαρτίζονται από λέξεις, σημεία στίξης, αριθμητικά σύμβολα, ειδικά σύμβολα και χαρακτήρες ανάλογα με το εργαλείο και το μοντέλο που τις χαρακτηρίζει. Η διαδικασία της τμηματοποίησης είναι ιδιαίτερα καθοριστική καθώς οι υπολογιστικοί αλγόριθμοι και τα νευρωνικά δίκτυα αδυνατούν να επεξεργαστούν ακατέργαστο κείμενο, απαιτώντας τη μετατροπή του σε μια δομημένη ακολουθία αναγνωριστικών IDs.

Σε ένα σύστημα αυτόματης δημιουργίας ερωτήσεων, το tokenization δεν λειτουργεί απλώς ως προπαρασκευαστικό στάδιο, αλλά ως κρίσιμο σημείο ελέγχου της ποιότητας. Εάν μια φράση τεμαχιστεί λανθασμένα, τότε η αναγνώριση ονομαστικών οντοτήτων, η ετικετοποίηση μερών (POS tagging) λόγου και η επιλογή υποψήφιων απαντήσεων είναι πιθανό να επηρεαστούν αρνητικά.

Ο χωρισμός του κειμένου σε προτάσεις (sentence splitting ή segmentation) είναι στενά συνδεδεμένος με το tokenization και ιδιαίτερα καθοριστικός διότι τα AQG χειρίζονται την πρόταση ως βασική μονάδα παραγωγής. Η σωστή ανίχνευση των ορίων κάθε πρότασης επιτρέπει στο σύστημα να απομονώνει συνεκτικές σημασιολογικές μονάδες, μέσα στις οποίες μπορεί να εντοπιστεί η πληροφορία-στόχος και να παραχθεί ερώτηση με επαρκή συνάφεια προς το αρχικό κείμενο.

Η σημασία του sentence splitting γίνεται ακόμη πιο εμφανής σε περιπτώσεις όπου η πληροφορία που απαιτείται για τη δημιουργία ερώτησης δεν περιορίζεται σε μία και μόνο πρόταση. Στην υλοποίηση της παρούσας εργασίας, παρότι η πρόταση αποτελεί την κύρια μονάδα επιλογής answer candidate, αξιοποιείται σε ορισμένες περιπτώσεις και γειτονικό συμφραζόμενο μέσω λογικής sliding window, ώστε το μοντέλο παραγωγής να έχει πρόσβαση σε πλουσιότερο πλαίσιο. Η επιλογή αυτή βασίζεται στην παρατήρηση ότι πολλές εκπαιδευτικές προτάσεις περιλαμβάνουν αντωνυμικές αναφορές, ελλειπτικές σχέσεις ή εξαρτήσεις που γίνονται κατανοητές μόνο αν ληφθεί υπόψη και η προηγούμενη ή επόμενη πρόταση. Συνεπώς, ο ακριβής χωρισμός προτάσεων δεν επηρεάζει μόνο την τοπική ανάλυση, αλλά και την ποιότητα του context που τελικά τροφοδοτεί το γλωσσικό μοντέλο παραγωγής.

Αξίζει να σημειωθεί και η έννοια του multi-word tokenization. Σε ορισμένες γλώσσες ή σε συγκεκριμένες επιφανειακές μορφές, μία φαινομενικά ενιαία ακολουθία μπορεί να αντιστοιχεί σε περισσότερες από μία γλωσσολογικές μονάδες. Η σωστή διαχείριση αυτών των περιπτώσεων είναι σημαντική για να παραμένουν συνεπή τα επόμενα στάδια του pipeline και να μη δημιουργούνται σφάλματα σε μορφοσυντακτικές ή σημασιολογικές αναλύσεις.

2.1.2 Ανάλυση μερών του λόγου (POS tagging)

Η ετικετοποίηση μερών του λόγου (Part of Speech tagging) αποτελεί την κρίσιμη διαδικασία κατά την οποία κάθε token στο κείμενο χαρακτηρίζεται με μια γραμματική κατηγορία όπως ουσιαστικό (NOUN), ρήμα (VERB), επίθετο (ADJ), αντωνυμία (PRON) ή άρθρα με (DET) και επίρρημα (ADV). Όπως προαναφέρθηκε τα tokens αφορούν και αριθμητικά δεδομένα, ειδικά σύμβολα ή χαρακτήρες και. Έτσι πέρα από τα μέρη του λόγου έχουμε τη διάκριση και κατηγοριοποίηση υπό την ετικέτα NUM (Numbers), PUNCT (Punctuation) ή SYM (Symbols) στο πλαίσιο των Universal Dependencies (UD) (Nivre et al., 2016). Αυτή η καθολική κάλυψη διασφαλίζει ότι κάθε μονάδα του κειμένου αποκτά μια σαφή λειτουργική ταυτότητα επιτρέποντας στο σύστημα να την διακρίνει.

Σε εκπαιδευτικά κείμενα, τα answer candidates που οδηγούν σε χρηστικές ερωτήσεις ανήκουν στην κατηγορία των ουσιαστικών ή των κύριων ονομάτων. Αντίθετα άρθρα, επίθετα, αντωνυμίες και επιρρήματα απορρίπτονται ή προτιμάται η διαδικασία συμπερίληψης τους και συνοδείας ενός ουσιαστικού. Για αυτό τον λόγο, το POS tagging δεν λειτουργεί μόνο περιγραφικά, αλλά και ως φίλτρο επιλογής υποψηφίων όρων.

Το POS tagging επομένως λειτουργεί ως ενδιάμεσο επίπεδο ανάμεσα στην επιφανειακή μορφή του κειμένου και στη λογική λήψης αποφάσεων. Ενισχύει την ανθεκτικότητα του συστήματος, επιτρέπει ευρετικά φίλτρα ποιότητας και συμβάλλει ώστε το downstream question generation να βασίζεται σε όρους με μεγαλύτερη πιθανότητα να παράξουν καθαρές, σαφείς και αξιοποιήσιμες ερωτήσεις.

2.1.3 Λημματοποίηση (LEM)

Η λημματοποίηση (Lemmatization) είναι η διαδικασία μετασχηματισμού μιας λέξης στη βασική ή λεξικογραφική της μορφή, δηλαδή στο λήμμα της. Σε αντίθεση με την απλή

αποκοπή καταλήξεων (stemming), η λημματοποίηση επιδιώκει μορφολογικά και γλωσσολογικά έγκυρο αποτέλεσμα. Για παράδειγμα, οι μορφές “running”, “ran” και “runs” μπορούν να αντιστοιχιστούν στο λήμμα “run”, ενώ τύποι ουσιαστικών σε πληθυντικό μπορούν να επανέλθουν στην ενική τους μορφή. Η λειτουργία αυτή είναι σημαντική σε πολλά συστήματα NLP, επειδή επιτρέπει την ενοποίηση διαφορετικών μορφολογικών παραλλαγών κάτω από κοινή αναπαράσταση.

Στο πλαίσιο της αυτόματης δημιουργίας ερωτήσεων, η λημματοποίηση έχει ιδιαίτερη αξία επειδή βοηθά στην πιο σταθερή αντιμετώπιση της πληροφορίας. Δύο λέξεις που εμφανίζονται σε διαφορετική κλιτική ή χρονική μορφή μπορεί να αναφέρονται στην ίδια έννοια, και άρα να πρέπει να θεωρηθούν συγγενείς κατά την ανάλυση ή το φιλτράρισμα. Χωρίς λημματοποίηση, το σύστημα θα αντιμετώπιζε κάθε επιφανειακή μορφή ως εντελώς ξεχωριστή μονάδα, γεγονός που θα αύξανε τον θόρυβο και θα δυσχέραινε τη γενίκευση.

2.1.4 Συντακτική Ανάλυση Εξαρτήσεων (Dependency Parsing)

Η Συντακτική Ανάλυση Εξαρτήσεων (Dependency Parsing) αποτελεί το στάδιο του pipeline όπου προσδιορίζεται η δομική οργάνωση μιας πρότασης μέσω των σχέσεων μεταξύ των λέξεων που τη συνθέτουν. Εστιάζει κυρίως στις ασύμμετρες δυαδικές σχέσεις εξάρτησης, όπου κάθε λέξη (εξαρτώμενη-dependent) συνδέεται με μια άλλη λέξη που λειτουργεί ως η κεφαλή της (head). Το αποτέλεσμα αυτής της διαδικασίας είναι η παραγωγή ενός κατευθυνόμενου δέντρου στο οποίο η ρίζα είναι συνήθως το κύριο ρήμα πρότασης και οι υπόλοιπες λέξεις συνδέονται ιεραρχικά μέσω επώνυμων σχέσεων όπως υποκείμενο (nsubj), αντικείμενο (obj) ή προσδιορισμός (amod)

Η εξαγωγή του συντακτικού δέντρου είναι καθοριστική για την κατανόηση του νοήματος, καθώς επιτρέπει τη διάκριση των ρόλων των δρώντων προσώπων και των αντικειμένων μέσα στην πρόταση. Τα δεδομένα αυτά αποθηκεύονται με το διεθνές πρότυπο - Universal Dependencies (UD) εξασφαλίζοντας ότι η συντακτική πληροφορία είναι δομημένη με τρόπο που επιτρέπει την περαιτέρω επεξεργασία, όπως η εξαγωγή σχέσεων (relation extraction). Η ακρίβεια σε αυτό το στάδιο επηρεάζει την ποιότητα της τελικής πληροφορίας καθώς ορίζει τον σκελετό όπου θα στηριχτεί η γλωσσική ερμηνεία του κειμένου.

2.1.5 Αναγνώριση Ονομαστικών Οντοτήτων (NER)

Η αναγνώριση Ονομαστικών Οντοτήτων (Named Entity Recognition -NER) αποσκοπεί στον εντοπισμό και την κατηγοριοποίηση λέξεων ή φράσεων που αντιστοιχούν σε πρόσωπα (PER), τοποθεσίες (LOC), οργανισμούς (ORG), ημερομηνίες (DATE), αριθμούς (NUM) και άλλες σημασιολογικά διακριτές κατηγορίες. Σε αντίθεση με το POS tagging που αποδίδει γραμμικές ετικέτες σε μεμονωμένα tokens, το NER επιχειρεί να εντοπίσει τμήματα του κειμένου με σημασιολογική αυτοτέλεια και υψηλή πληροφοριακή αξία.

Τα αποτελέσματα του NER αποθηκεύονται στο πεδίο ents του εγγράφου παρέχοντας μια δομημένη λίστα από οντότητες μαζί με τους χαρακτηρισμούς τους. Η πληροφορία αυτή είναι θεμελιώδης για εφαρμογές εξαγωγής γνώσης καθώς επιτρέπει τη μετάβαση από την απλή επεξεργασία λέξεων στην κατανόηση των υποκείμενων εννοιών και των σχέσεων με των δρώντων προσώπων ή οργανισμών στο κείμενο.

Επιπλέον, η NER συμβάλλει αποφασιστικά στη δημιουργία πειστικών επιλογών πολλαπλής επιλογής. Αν η σωστή απάντηση είναι πρόσωπο, τότε και οι λανθασμένες επιλογές θα πρέπει, κατά κανόνα, να είναι επίσης πρόσωπα· το ίδιο ισχύει για τοποθεσίες, οργανισμούς ή ημερομηνίες. Η διατήρηση αυτής της σημασιολογικής συνέπειας περιορίζει παράλογους συνδυασμούς και αυξάνει την παιδαγωγική εγκυρότητα των ερωτήσεων.

2.1.6 Η Βιβλιοθήκη Stanza

Η Stanza είναι βιβλιοθήκη ανοιχτού κώδικα για Επεξεργασία Φυσικής Γλώσσας, η οποία προσφέρει ενιαίο νευρωνικό pipeline για tokenization, sentence segmentation, multi-word token expansion, POS tagging, λημματοποίηση, μορφολογικά χαρακτηριστικά, dependency parsing και αναγνώριση ονομαστικών οντοτήτων (Qi et al., 2020). Η ιδιαίτερη αξία της έγκειται στο ότι συνδυάζει πολλά στάδια ανάλυσης μέσα σε ένα συνεκτικό περιβάλλον, επιτρέποντας την παραγωγή ομοιογενών γλωσσολογικών αναπαραστάσεων χωρίς τη χρήση πολλών ετερογενών εργαλείων.

Η Stanza οργανώνει την επεξεργασία του κειμένου ως ακολουθία processors, όπου κάθε στάδιο τροφοδοτεί το επόμενο. Το αρχικό κείμενο εισέρχεται πρώτα στον μηχανισμό tokenization και sentence splitting. Στη συνέχεια, όπου απαιτείται, ακολουθεί η ανάλυση multi-word tokens, ενώ πάνω σε αυτή τη βάση εφαρμόζονται POS tagging, lemmatization και dependency parsing. Τέλος, το στάδιο NER αποδίδει σημασιολογικές ετικέτες σε spans του κειμένου. Με αυτό τον τρόπο, κάθε token συνοδεύεται σταδιακά από πολλαπλά επίπεδα γλωσσολογικής πληροφορίας.

Η τεχνική σημασία της Stanza συνδέεται και με τη χρήση συνεπούς αναπαράστασης τύπου UD για μεγάλο μέρος του pipeline. Αυτό διευκολύνει την ομοιομορφία των γραμματικών και μορφοσυντακτικών annotations, ενώ παράλληλα ενισχύει τη δυνατότητα διαγλωσσικής επέκτασης. Η ιδιότητα αυτή είναι σημαντική για την παρούσα εργασία, καθώς, παρότι η τρέχουσα παραγωγή ερωτήσεων εστιάζει κυρίως στα αγγλικά, υπάρχει σχεδιαστική υλοποίηση για την υποστήριξη ελληνικού κειμένου.

Ιδιαίτερη τεχνική αξία έχει επίσης το γεγονός ότι η Stanza παρέχει τόσο token-level όσο και span-level πληροφορία. Στο επίπεδο του POS και της λημματοποίησης, κάθε token συνοδεύεται από τη γραμματική και μορφολογική του ταυτότητα. Στο επίπεδο του dependency parsing, κάθε λέξη αποκτά πληροφορία σχετικά με την κεφαλή της και τον τύπο της συντακτικής της σχέσης. Στο επίπεδο του NER, το σύστημα μπορεί να μεταβεί από την επιφανειακή ακολουθία tokens σε ολοκληρωμένες σημασιολογικές οντότητες.

2.1.7 Η Βιβλιοθήκη Stanza σε σύγκριση με την spaCy

Η spaCy αποτελεί μία από τις πιο γνωστές και ευρέως χρησιμοποιούμενες βιβλιοθήκες NLP στην Python (Explosion AI, n.d.). Υποστηρίζει tokenization, POS tagging, dependency parsing, NER, lemmatization και πλήθος επιπλέον δυνατοτήτων, ενώ έχει σχεδιαστεί με έντονο προσανατολισμό προς production εφαρμογές. Η ταχύτητα εκτέλεσης, η ευχρηστία του API και η δυνατότητα επεξεργασίας μεγάλων όγκων κειμένου καθιστούν τη spaCy εξαιρετικά ισχυρή επιλογή σε περιβάλλοντα βιομηχανικής αξιοποίησης.

Παρά τα πλεονεκτήματα αυτά, η Stanza κρίθηκε καταλληλότερη για την παρούσα εργασία. Ο βασικός λόγος είναι ότι η διπλωματική δεν στοχεύει μόνο σε ταχεία επεξεργασία κειμένου, αλλά κυρίως σε γλωσσολογικά συνεπή ανάλυση που να υποστηρίζει με σαφήνεια τα πολλαπλά στάδια του συστήματος AQG. Η Stanza παρέχει ενιαίο pipeline για tokenization, MWT, POS, lemma, dependency parsing και NER, με ιδιαίτερη έμφαση στη συνεκτική γλωσσολογική αναπαράσταση και στη σύνδεση με το Universal Dependencies.

Σημαντική είναι επίσης η τεχνική διάσταση του multiword tokenization. Η Stanza ενσωματώνει ρητά στο pipeline της τον χειρισμό multiword tokens, γεγονός που είναι χρήσιμο όταν το σύστημα πρέπει να περάσει από την επιφανειακή μορφή του κειμένου σε πιο ακριβή μορφοσυντακτική αναπαράσταση. Για ένα έργο που βασίζεται σε ακριβή answer spans, σε λημματοποίηση και σε σημασιολογικά συνεπή αναγνώριση οντοτήτων, η δυνατότητα αυτή αποτελεί ουσιαστικό πλεονέκτημα. Η επιλογή του εργαλείου επομένως

δεν βασίστηκε αποκλειστικά στην παρούσα απόδοση, αλλά και στη μελλοντική προσαρμοστικότητα του συστήματος.

2.2 Transformers

2.2.1 Μηχανισμός Self Attention

Πριν την επικράτηση των Transformers, η μοντελοποίηση ακολουθιών στηριζόταν σε αναδρομικά δίκτυα RNN (Recurrent Neural Network) και αργότερα LSTM (Long Short-Term Memory) (Hochreiter & Schmidhuber, 1997), τα οποία διάβαζαν το κείμενο γραμμικά, μία λέξη τη φορά, μεταφέροντας πληροφορία μέσα από διαδοχικές κρυφές καταστάσεις. Το βασικό τους μειονέκτημα ήταν πρακτικό, δηλαδή όσο απομακρύνονταν δύο σχετιζόμενες λέξεις μέσα στην πρόταση, τόσο δυσκολότερα διατηρούνταν η μεταξύ τους εξάρτηση, ενώ η σειριακή φύση τους εμπόδιζε την παράλληλη εκπαίδευση.

Ο μηχανισμός attention αντιστρέφει αυτή τη λογική. Αντί να μεταφέρεται η πληροφορία βήμα-βήμα, κάθε λέξη «κοιτάζει» ταυτόχρονα όλες τις υπόλοιπες και τους αποδίδει βάρη ανάλογα με το πόσο συνεισφέρουν στην ερμηνεία της. Στην ειδική περίπτωση του self-attention, η σύγκριση γίνεται ανάμεσα στα tokens της ίδιας ακολουθίας (Vaswani et al., 2017), παράγοντας μια αναπαράσταση όπου η σημασία κάθε λέξης διαμορφώνεται από ολόκληρη την πρόταση και όχι μόνο από τη γειτονιά της.

Για το πρόβλημα της αυτόματης δημιουργίας ερωτήσεων, αυτή η ιδιότητα είναι κρίσιμη. Η μετατροπή μιας δήλωσης σε ερώτηση απαιτεί το μοντέλο να αντιληφθεί ποιο τμήμα της πρότασης είναι ο πυρήνας της πληροφορίας και πώς αυτό συνδέεται με τα υπόλοιπα στοιχεία, ώστε η ερώτηση να βγει γραμματικά φυσική και σωστά στοχευμένη. Η ταυτόχρονη πρόσβαση σε όλη την ακολουθία είναι ακριβώς αυτό που επιτρέπει στο μοντέλο παραγωγής του συστήματός μου να χειρίζεται σωστά αντωνυμικές αναφορές και εξαρτήσεις μεγάλης εμβέλειας μέσα στην ίδια πρόταση.

2.2.2 Encoder-Decoder Αρχιτεκτονική

Η αρχιτεκτονική των Transformers μπορεί να υλοποιηθεί με διαφορετικές παραλλαγές, όμως η πιο χαρακτηριστική μορφή της για εργασίες παραγωγής κειμένου είναι η διάταξη encoder-decoder. Ο encoder αναλαμβάνει να επεξεργαστεί το κείμενο εισόδου και να παράγει μια πλούσια συμφραστική αναπαράσταση του νοήματός του. Ο decoder, στη

συνέχεια, χρησιμοποιεί αυτή την αναπαράσταση για να δημιουργήσει βήμα προς βήμα την έξοδο, δηλαδή μια νέα ακολουθία κειμένου.

Η λογική αυτή είναι ιδιαίτερα σημαντική σε εργασίες sequence-to-sequence, όπως η μηχανική μετάφραση, η περίληψη και η παραγωγή ερωτήσεων. Ο encoder δεν περιορίζεται σε απλή αποθήκευση των λέξεων της εισόδου, αλλά κατασκευάζει embeddings που ενσωματώνουν συμφοραζόμενη πληροφορία. Ο decoder, αντίστοιχα, δεν προβλέπει τυφλά την επόμενη λέξη, αλλά συνδυάζει ό,τι έχει ήδη παραχθεί με τη συμφοραστική γνώση που έχει προσφέρει ο encoder.

Στην παρούσα εργασία, η σημασία της encoder-decoder λογικής είναι καθοριστική, επειδή ο στόχος δεν είναι απλώς η αναγνώριση μιας πληροφορίας μέσα στο κείμενο, αλλά ο μετασχηματισμός της σε νέα γλωσσική μορφή, δηλαδή σε ερώτηση. Το σύστημα πρέπει να κατανοήσει το input κείμενο, να εστιάσει στον κατάλληλο answer span και στη συνέχεια να παραγάγει ερώτηση γραμματικά αποδεκτή και νοηματικά συνεπή. Για αυτόν τον λόγο, τα encoder-decoder μοντέλα συνδέονται φυσικά με την εργασία της Automatic Question Generation.

2.2.3 Bidirectional Encoder Representations from Transformers (BERT)

Το BERT (Bidirectional Encoder Representations from Transformers) σχεδιάστηκε κυρίως ως encoder-only αρχιτεκτονική (Devlin et al., 2019). Ένα από τα σημαντικά στοιχεία του είναι ότι λαμβάνει υπόψη τόσο το αριστερό όσο και το δεξί συμφοραζόμενο κάθε λέξης κατά την προεκπαίδευση. Αυτό το χαρακτηριστικό το καθιστά εξαιρετικά ισχυρό για εργασίες κατανόησης κειμένου, όπως classification, token labeling, NER, question answering τύπου extractive και γενικότερα semantic understanding.

Η δυναμική του BERT έγκειται κυρίως στην ικανότητά του να παράγει υψηλής ποιότητας contextual embeddings. Σε πολλές εργασίες NLP αποτέλεσε σημείο καμπής, επειδή έδωσε πολύ καλύτερα αποτελέσματα από παλαιότερες αρχιτεκτονικές. Ωστόσο, η ισχύς του αυτή αφορά κυρίως το επίπεδο της αναπαράστασης και της κατανόησης, όχι της ελεύθερης παραγωγής κειμένου. Επειδή το BERT δεν διαθέτει decoder, δεν είναι από τη φύση του κατάλληλο για να γράψει νέα ακολουθία κειμένου με τον τρόπο που απαιτείται στην παραγωγή ερωτήσεων.

2.2.4 Bidirectional and Auto-Regressive Transformers (BART)

Το BART (Bidirectional and Auto-Regressive Transformers) είναι μοντέλο encoder-decoder (Lewis et al., 2020), συνδυάζοντας στοιχεία από bidirectional encoders τύπου BERT και autoregressive decoders κατάλληλους για generation. Το μοντέλο εκπαιδεύεται να ανακατασκευάζει αρχικό κείμενο από αλλοιωμένη ή μερικώς κατεστραμμένη εκδοχή του. Αυτή η λογική το καθιστά ιδιαίτερα αποτελεσματικό σε εργασίες όπου απαιτείται αναδιατύπωση ή παραγωγή κειμένου, όπως περίληψη, παραφραστική αναδιατύπωση και μετάφραση.

Το BART συνδέεται άμεσα με το AQG, διότι ως encoder-decoder, μπορεί να δεχθεί ένα κείμενο εισόδου και να παράγει νέο κείμενο εξόδου. Σε αντίθεση με το BERT, δεν περιορίζεται στην κατανόηση αλλά υποστηρίζει ενεργά γλωσσική παραγωγή. Για αυτόν τον λόγο θα μπορούσε να αποτελέσει θεωρητικά κατάλληλη επιλογή για question generation, και στη βιβλιογραφία αναφέρεται συχνά μαζί με τα T5-like μοντέλα ως μία από τις βασικές seq2seq οικογένειες.

2.2.5 GreekBERT

Στη διερεύνηση της ελληνικής υποστήριξης εξετάστηκε και το GreekBERT, την προσαρμογή του BERT στην ελληνική γλώσσα (Koutsikakis et al., 2020). Η βασική του σημασία έγκειται στο ότι παρέχει ισχυρές contextual αναπαραστάσεις για ελληνικό κείμενο, καλύπτοντας ένα σημαντικό κενό στη διαθεσιμότητα ελληνόφωνων προεκπαιδευμένων μοντέλων. Από ερευνητική άποψη είναι ιδιαίτερα ελκυστικό, επειδή επιτρέπει την αξιοποίηση transformer-based αναπαραστάσεων σε εργασίες ελληνικού NLP, όπως classification, NER, POS-related downstream tasks και γενικότερα semantic understanding.

Ωστόσο, ο πυρήνας παραμένει τύπου BERT, δηλαδή encoder-only. Αυτό σημαίνει ότι, όπως και το αρχικό BERT, είναι περισσότερο κατάλληλο για εργασίες κατανόησης παρά για ανοιχτή παραγωγή νέου κειμένου. Κατά συνέπεια, παρότι η ύπαρξή του είναι πολύ σημαντική ως θεωρητική και τεχνολογική βάση για την ελληνική γλώσσα, δεν αποτελεί αυτονόητη λύση για question generation, όπου απαιτείται μοντέλο που να μπορεί να συνθέσει φυσική έξοδο και όχι μόνο να αναλύσει την είσοδο.

Αυτό εξηγεί και γιατί στην παρούσα εργασία δεν επιλέχθηκε GreekBERT ως βασικό μοντέλο παραγωγής, ακόμη και αν η παραγωγή ελληνικών ερωτήσεων πραγματοποιήθηκε

με εναλλακτική πρόβλεψη. Η στρατηγική που κρίθηκε πιο πρακτική ήταν η αξιοποίηση μοντέλου encoder-decoder για generation, σε συνδυασμό με translation-based ή πολυγλωσσικές προσεγγίσεις, αντί της χρήσης ενός encoder-only ελληνικού μοντέλου. Με άλλα λόγια, το GreekBERT παραμένει πολύτιμο ως αναφορά για ελληνικό NLP, αλλά όχι ως η πιο συμβατή αρχιτεκτονική για τον πυρήνα του AQG pipeline.

2.2.6 Text-to-Text Transfer Transformer (T5)

Το T5 (Text-to-Text Transfer Transformer) είναι επίσης μοντέλο encoder-decoder (Raffel et al., 2020), αλλά διαφοροποιείται ως προς τη φιλοσοφία του καθώς αντιμετωπίζει κάθε εργασία (μετάφραση, περίληψη, παραγωγή ερώτησης) ως μετασχηματισμό μιας ακολουθίας κειμένου σε μια άλλη. Αυτή η ενιαία αντιμετώπιση εισόδου και εξόδου ως φυσικής γλώσσας το κάνει ιδιαίτερα ευέλικτο για παραγωγικές εργασίες.

Για τη παρούσα υλοποίηση, η επιλογή του T5 στηρίχθηκε σε τρία πρακτικά κριτήρια. Πρώτον, η αρχιτεκτονική του είναι συμβατή με το AQG ως εργασία παραγωγής. Δεύτερον, η λογική text-to-text διευκολύνει την προσαρμογή του μοντέλου σε πολλές παραλλαγές της ίδιας εργασίας. Τρίτον, στο οικοσύστημα Hugging Face είναι διαθέσιμες fine-tuned εκδόσεις του T5 που έχουν εκπαιδευτεί ειδικά για question generation, κάτι που μειώνει σημαντικά το τεχνικό κόστος πειραματισμού και επιτρέπει πιο άμεση πρακτική αξιοποίηση.

2.2.7 Το οικοσύστημα Hugging Face Transformers

Η πρακτική αξιοποίηση των μοντέλων Transformer στην ερευνητική και εφαρμοσμένη ανάπτυξη επιταχύνθηκε σημαντικά μέσα από το οικοσύστημα του Hugging Face (Wolf et al., 2020· Hugging Face, n.d.). Η βιβλιοθήκη Transformers και το Hugging Face Hub προσφέρουν ενιαίο τρόπο φόρτωσης, παραμετροποίησης και χρήσης μεγάλου αριθμού προεκπαιδευμένων μοντέλων, μειώνοντας σημαντικά το τεχνικό κόστος πειραματισμού. Με τον τρόπο αυτό, ο ερευνητής δεν χρειάζεται να υλοποιήσει από την αρχή πολύπλοκες αρχιτεκτονικές, αλλά μπορεί να αξιοποιήσει έτοιμα checkpoints και τεκμηριωμένα interfaces.

Η σημασία του Hugging Face δεν περιορίζεται στην ευκολία χρήσης. Το οικοσύστημα αυτό λειτουργεί ως κοινός τόπος διαμοιρασμού μοντέλων, tokenizers, documentation και model cards (Mitchell et al., 2019). Στη παρούσα ΔΕ εξετάζονται εναλλακτικές λύσεις και

διαφορετικά μοντέλα generation, το Hugging Face επιτρέπει συγκριτικό πειραματισμό και ταχεία ενσωμάτωση διαφορετικών αρχιτεκτονικών μέσα στο ίδιο προγραμματιστικό πλαίσιο.

Στην πράξη, η χρήση του Hugging Face καθιστά εφικτή τη μετάβαση από τη θεωρία στην υλοποίηση. Το T5, το BART, το BERT και άλλες οικογένειες μοντέλων γίνονται άμεσα προσβάσιμα μέσω τυποποιημένων APIs. Αυτό διευκολύνει όχι μόνο τη φόρτωση του μοντέλου, αλλά και τη διαχείριση tokenization, generation parameters και αποθήκευσης checkpoints. Για τον λόγο αυτό, το Hugging Face στην παρούσα εργασία δεν αντιμετωπίζεται ως απλή βοηθητική βιβλιοθήκη, αλλά ως κρίσιμο τεχνολογικό υπόβαθρο της υλοποίησης.

2.3 Μοντέλα T5

2.3.1 mT5

Το mT5 πρόκειται για την πολυγλωσσική εκδοχή του T5 (Xue et al., 2021 Google, n.d.), σχεδιασμένη να καλύπτει πολλές γλώσσες μέσα στο ίδιο text-to-text πλαίσιο, διαχειριζόμενη διαφορετικά λεξιλόγια και μορφολογικά συστήματα. Για ένα σύστημα που φιλοδοξεί να επεκταθεί πέρα από τα αγγλικά, ένα τέτοιο μοντέλο είναι θεωρητικά ελκυστικό, αφού θα απέφευγε την ανάγκη χωριστής αρχιτεκτονικής ανά γλώσσα.

Παρά τα πλεονεκτήματά του, η πολυγλωσσική ικανότητα δεν συνεπάγεται αυτόματα και βέλτιστη απόδοση σε κάθε επιμέρους γλώσσα ή υποεργασία. Στην πράξη, ένα πολυγλωσσικό μοντέλο συχνά καλείται να ισορροπήσει ανάμεσα σε πολλές γλώσσες και μπορεί να υπολείπεται μιας πιο εξειδικευμένης παραλλαγής σε συγκεκριμένο σενάριο χρήσης. Για τον λόγο αυτό, το mT5 κρίθηκε ως εναλλακτική ή μελλοντική κατεύθυνση.

2.3.2 iarfmoose/t5-base-question-generator

Το iarfmoose/t5-base-question-generator αποτελεί επίσης γνωστή fine-tuned παραλλαγή της οικογένειας T5 (iarfmoose, n.d.) για question generation και χρησιμοποιήθηκε σε πρώιμα στάδια πειραματισμού ως πιθανή βάση του συστήματος. Η ύπαρξή του είναι σημαντική γιατί δείχνει ότι η οικογένεια T5 δεν αντιστοιχεί σε μία μόνο πρακτική υλοποίηση για AQG, αλλά σε σύνολο από προσαρμοσμένες εκδόσεις που

διαφοροποιούνται ως προς τα δεδομένα fine-tuning, τον τρόπο διατύπωσης της εισόδου και τη συμπεριφορά της εξόδου.

Παρά τη χρησιμότητά του ως εναλλακτικής, το συγκεκριμένο μοντέλο δεν συνδέεται τόσο καθαρά με τον μηχανισμό answer highlighting όσο το valhalla/t5-base-qg-hl. Αυτό έχει πρακτικές συνέπειες, επειδή σε ένα pipeline όπου η answer selection έχει ήδη επιλυθεί σε προηγούμενο στάδιο, είναι προτιμότερο το μοντέλο generation να συνεργάζεται όσο το δυνατόν πιο άμεσα με αυτή τη δομή. Όταν η είσοδος δεν εκφράζει με σαφήνεια τον answer span, αυξάνεται ο κίνδυνος η παραγόμενη ερώτηση να είναι λιγότερο εστιασμένη.

2.3.3 valhalla/t5-base-qg-hl

Η παραλλαγή valhalla/t5-base-qg-hl αποτελεί μία από τις πλέον γνωστές fine-tuned εκδόσεις του T5 (Valhalla, n.d.) για την παραγωγή ερωτήσεων. Το ιδιαίτερο χαρακτηριστικό της είναι ότι ακολουθεί τη λογική του highlight format, όπου το answer span επισημαίνεται μέσα στο context με ειδικά tokens. Με αυτόν τον τρόπο, το μοντέλο δεν χρειάζεται να υποθέσει μόνο του ποιο τμήμα του κειμένου είναι το επίκεντρο της ερώτησης, αλλά λαμβάνει σαφή και ρητή καθοδήγηση.

Αυτή η σχεδιαστική λογική είναι εξαιρετικά σημαντική για την AQG. Σε πολλές περιπτώσεις, μια πρόταση περιέχει περισσότερες από μία πιθανές πληροφορίες-στόχους και χωρίς σαφή ένδειξη το μοντέλο μπορεί να παραγάγει ερώτηση είτε υπερβολικά γενική είτε προσανατολισμένη σε λάθος σημείο. Το highlight format μειώνει αυτόν τον βαθμό αβεβαιότητας και βελτιώνει τη συνάφεια ανάμεσα στην επιλεγμένη απάντηση και στην παραγόμενη ερώτηση.

Στο πλαίσιο της παρούσας εφαρμογής, το valhalla/t5-base-qg-hl συνδέεται ιδανικά με τον τρόπο λειτουργίας του pipeline. Το σύστημα πρώτα επιλέγει answer candidate μέσω γλωσσολογικής ανάλυσης και στη συνέχεια ενσωματώνει αυτή την επιλογή στο input του μοντέλου με explicit highlighting. Με άλλα λόγια, το μοντέλο δεν καλείται να επιλύσει ταυτόχρονα και το πρόβλημα της answer selection και το πρόβλημα της question generation, αλλά επικεντρώνεται στη δεύτερη φάση, όπου είναι ισχυρότερο.

2.3.4 Αιτιολόγηση επιλογής μοντέλου T5

Η τελική επιλογή μοντέλου στην παρούσα εργασία δεν βασίστηκε μόνο στη γενική φήμη της οικογένειας T5, αλλά κυρίως στη συμβατότητά της με την αρχιτεκτονική του ίδιου του συστήματος. Το pipeline που υλοποιήθηκε στηρίζεται σε προηγούμενη επιλογή answer spans μέσω Stanza, σε γλωσσολογικό φιλτράρισμα και σε ανάγκη παραγωγής στοχευμένης ερώτησης πάνω σε ήδη προσδιορισμένη απάντηση. Για αυτόν τον λόγο, το valhalla/t5-base-qg-hl κρίθηκε ως η πλέον κατάλληλη λύση, επειδή αξιοποιεί άμεσα τη λογική του explicit answer highlighting.

Σε σύγκριση με άλλες εναλλακτικές, η συγκεκριμένη επιλογή εξυπηρετεί καλύτερα την υβριδική φιλοσοφία του έργου. Η answer selection δεν αφήνεται εξ ολοκλήρου στο μοντέλο, αλλά υποστηρίζεται από ανεξάρτητο γλωσσολογικό στάδιο. Το μοντέλο generation, επομένως, πρέπει να συνεργάζεται με αυτή τη ροή και όχι να την παρακάμπτει. Το valhalla/t5-base-qg-hl προσφέρει αυτή τη γέφυρα ανάμεσα στο rule-guided preprocessing και στη νευρωνική παραγωγή φυσικής γλώσσας.

2.4 Αυτόματη Δημιουργία Ερωτήσεων (AQG)

Η Αυτόματη Δημιουργία Ερωτήσεων (Automatic Question Generation - AQG) αποτελεί ερευνητικό πεδίο της Επεξεργασίας Φυσικής Γλώσσας που εστιάζει στην παραγωγή ερωτήσεων από κείμενο, με ελάχιστη ή και μηδενική ανθρώπινη παρέμβαση. Η σημασία του πεδίου είναι ιδιαίτερα αυξημένη στο εκπαιδευτικό πλαίσιο, όπου η ανάγκη για γρήγορη παραγωγή δραστηριοτήτων αξιολόγησης, επαναληπτικής μελέτης και διαδραστικού υλικού είναι συνεχώς αυξανόμενη. Η AQG δεν περιορίζεται στη μετατροπή μιας δήλωσης σε ερώτηση, αλλά σχετίζεται με βαθύτερα προβλήματα γλωσσικής κατανόησης, επιλογής πληροφορίας, φυσικής διατύπωσης και παιδαγωγικής καταλληλότητας.

Η πρόοδος του πεδίου έχει υπάρξει σημαντική τις τελευταίες δεκαετίες (Das et al., 2021· Kurdi et al., 2020). Οι πρώτες προσεγγίσεις στηρίζονταν κυρίως σε γλωσσικούς κανόνες, χειροποίητα πρότυπα και συντακτικούς μετασχηματισμούς. Αργότερα, η ανάπτυξη των νευρωνικών μοντέλων και των αρχιτεκτονικών Transformer μετέφερε το κέντρο βάρους σε προσεγγίσεις που μαθαίνουν από δεδομένα και μπορούν να παράγουν πιο φυσική γλώσσα. Παράλληλα, εμφανίστηκαν υβριδικές μέθοδοι, οι οποίες προσπαθούν να συνδυάσουν την

ερμηνευσιμότητα και τον έλεγχο των κανόνων με τη γλωσσική ρευστότητα των νευρωνικών μοντέλων.

Στο πλαίσιο της παρούσας εργασίας καθίσταται σαφές γιατί η προτεινόμενη προσέγγιση δεν είναι ούτε αμιγώς rule-based ούτε αμιγώς neural, αλλά εντάσσεται στις υβριδικές προσεγγίσεις. Η κατανόηση των βασικών κατηγοριών AQG συμβάλλει επομένως όχι μόνο στην αποτύπωση της πορείας του πεδίου, αλλά και στην καλύτερη τεκμηρίωση της αρχιτεκτονικής που υλοποιήθηκε στη συνέχεια.

2.4.1 Rule-based προσεγγίσεις

Οι rule-based προσεγγίσεις αποτέλεσαν την πρώτη οργανωμένη φάση ανάπτυξης συστημάτων αυτόματης δημιουργίας ερωτήσεων (Le et al., 2014). Η βασική τους λογική είναι ότι, εφόσον το σύστημα μπορεί να αναλύσει συντακτικά και γραμματικά μια πρόταση, είναι δυνατό να εφαρμόσει κανόνες μετασχηματισμού ώστε να παραχθεί μια ερώτηση. Για παράδειγμα, μια δηλωτική πρόταση μπορεί να μετασχηματιστεί σε ερώτηση αντικαθιστώντας ένα επιλεγμένο τμήμα της με ερωτηματική λέξη, όπως «who», «what», «where» ή «when», και αναδιατάσσοντας τα υπόλοιπα στοιχεία της πρότασης.

Η ελκυστικότητα αυτής της κατηγορίας έγκειται στον υψηλό βαθμό ελέγχου που προσφέρει. Ο ερευνητής γνωρίζει με σαφήνεια γιατί παρήχθη μια συγκεκριμένη ερώτηση, ποιος κανόνας εφαρμόστηκε και σε ποιο τμήμα της πρότασης. Αυτό είναι ιδιαίτερα χρήσιμο όταν ζητείται διαφάνεια, προβλεψιμότητα και εύκολη αποσφαλμάτωση. Επιπλέον, οι rule-based μέθοδοι είναι συχνά λιγότερο απαιτητικές σε δεδομένα εκπαίδευσης, καθώς στηρίζονται περισσότερο στη γλωσσολογική γνώση παρά σε μεγάλους όγκους επισημειωμένων δεδομένων.

Παρά τα πλεονεκτήματά τους, οι μέθοδοι αυτές εμφανίζουν σαφή όρια. Η ποιότητα της εξόδου εξαρτάται σε μεγάλο βαθμό από το πόσο καλά οι χειροποίητοι κανόνες καλύπτουν τις πιθανές γραμματικές δομές του κειμένου. Σε σύνθετες ή λιγότερο αναμενόμενες προτάσεις, τα συστήματα αυτά συχνά παράγουν άκαμπτες, αφύσικες ή ελλιπείς ερωτήσεις. Επιπλέον, η μεταφορά τους σε διαφορετικά γνωστικά πεδία ή γλώσσες δεν είναι πάντοτε εύκολη, επειδή απαιτείται νέα σειρά κανόνων και εκτενής χειροκίνητη προσαρμογή. Για τον λόγο αυτό, αν και οι rule-based προσεγγίσεις διατηρούν αξία ως βάση ή ως

υποστηρικτικό εργαλείο, από μόνες τους σπάνια επαρκούν για παραγωγή υψηλής ποιότητας ερωτήσεων σε ανοικτά και ετερογενή κείμενα.

2.4.2 Neural προσεγγίσεις — seq2seq και fine-tuned Transformers

Η δεύτερη μεγάλη κατηγορία προσεγγίσεων στο AQG είναι οι νευρωνικές μέθοδοι, οι οποίες βασίζονται στη μάθηση από δεδομένα αντί στη ρητή κωδικοποίηση γλωσσικών κανόνων. Αρχικά, η ανάπτυξη του πεδίου επηρεάστηκε από μοντέλα ακολουθίας προς ακολουθία (sequence-to-sequence) (Sutskever et al., 2014), τα οποία χρησιμοποιήθηκαν με επιτυχία σε εργασίες όπως η μετάφραση και η περίληψη. Στα μοντέλα αυτά, ένα δίκτυο λαμβάνει ως είσοδο ένα κείμενο και εκπαιδεύεται να παράγει ως έξοδο ένα δεύτερο κείμενο, στην προκειμένη περίπτωση μια ερώτηση.

Τα Transformers και τα προεκπαιδευμένα μοντέλα όπως τα T5 και BART μπορούν να αξιοποιήσουν πολύ πιο αποτελεσματικά το συμφραζόμενο, να μάθουν πιο φυσικές διατυπώσεις και να προσαρμοστούν σε εξειδικευμένα tasks μέσω fine-tuning. Ειδικά σε εργασίες παραγωγής φυσικής γλώσσας, η αρχιτεκτονική encoder-decoder και η λογική text-to-text καθιστούν τέτοια μοντέλα ιδιαίτερα κατάλληλα για τη σύνθεση ερωτήσεων που δεν είναι απλώς γραμματικά σωστές, αλλά και πιο κοντά στον τρόπο που θα ρωτούσε ένας άνθρωπος.

Η ελκυστικότητα των neural προσεγγίσεων βρίσκεται ακριβώς σε αυτή τη γλωσσική φυσικότητα. Σε πολλές περιπτώσεις, οι ερωτήσεις που παράγονται από fine-tuned Transformers εμφανίζονται πιο ρευστές, πιο συνοπτικές και πιο κατάλληλα διατυπωμένες από εκείνες που προκύπτουν από άκαμπτα templates. Παρ' όλα αυτά, οι μέθοδοι αυτές δεν είναι χωρίς αδυναμίες. Η ποιότητα της εξόδου εξαρτάται από τα δεδομένα εκπαίδευσης, την προεκπαίδευση, τη διατύπωση της εισόδου και τη γενικότερη καταλληλότητα του μοντέλου για το συγκεκριμένο task. Επιπλέον, τα νευρωνικά συστήματα είναι λιγότερο ερμηνεύσιμα, ενδέχεται να εμφανίσουν hallucinations ή να παράγουν ερωτήσεις που είναι γλωσσικά σωστές αλλά όχι απόλυτα πιστές στο αρχικό κείμενο. Κατά συνέπεια, παρότι αποτελούν σήμερα τον βασικό κορμό της σύγχρονης AQG, συχνά χρειάζονται συμπληρωματικά φίλτρα και μηχανισμούς ελέγχου.

2.4.3 Υβριδικές προσεγγίσεις

Οι υβριδικές προσεγγίσεις προέκυψαν ως προσπάθεια να συνδυαστούν τα πλεονεκτήματα των δύο προηγούμενων κατηγοριών (Mulla & Gharpure, 2023). Αντί να βασίζονται αποκλειστικά είτε σε κανόνες είτε σε end-to-end νευρωνική παραγωγή, επιχειρούν να αξιοποιήσουν γλωσσολογική ανάλυση, ευρετικά φίλτρα και δομημένους πόρους στα στάδια όπου ο έλεγχος και η ερμηνευσιμότητα είναι σημαντικά, ενώ ταυτόχρονα χρησιμοποιούν νευρωνικά μοντέλα εκεί όπου απαιτείται παραγωγή φυσικής γλώσσας με μεγαλύτερη ρευστότητα.

Στην πράξη, μια υβριδική αρχιτεκτονική μπορεί να περιλαμβάνει rule-based επιλογή answer candidates, αναγνώριση οντοτήτων, φίλτρα εγκυρότητας, semantic constraints και στη συνέχεια neural question generation. Παρόμοια, η δημιουργία distractors μπορεί να συνδυάζει γλωσσολογικούς ή λεξικοσημασιολογικούς πόρους με νευρωνικές στρατηγικές αναδιατύπωσης ή βελτίωσης. Το αποτέλεσμα είναι ένα σύστημα περισσότερο ελεγχόμενο από ένα αμιγώς νευρωνικό μοντέλο, αλλά και περισσότερο φυσικό από μια αμιγώς template-based λύση.

Η σημασία των υβριδικών προσεγγίσεων είναι ιδιαίτερα μεγάλη στο εκπαιδευτικό AQG. Εκεί δεν αρκεί μόνο να παραχθεί μια γραμματικά σωστή ερώτηση. Χρειάζεται επίσης να διασφαλιστεί ότι η απάντηση είναι εκπαιδευτικά χρήσιμη, ότι οι distractors είναι σημασιολογικά συνεπείς, ότι η ερώτηση δεν αποκαλύπτει άμεσα τη λύση και ότι το παραγόμενο αποτέλεσμα μπορεί να ενταχθεί σε πρακτικό μαθησιακό περιβάλλον. Για τέτοιες απαιτήσεις, οι υβριδικές μέθοδοι εμφανίζονται συχνά ως πιο ρεαλιστική επιλογή από αμιγώς άκρο-προς-άκρο προσεγγίσεις.

2.4.4 Σύγκριση με την παρούσα εργασία

Η παρούσα εργασία εντάσσεται σαφώς στις υβριδικές προσεγγίσεις AQG. Η κατηγοριοποίηση αυτή δεν είναι τυπική, αλλά προκύπτει από τον ίδιο τον σχεδιασμό και τη ροή του συστήματος. Η εφαρμογή δεν βασίζεται αποκλειστικά σε κανόνες, καθώς ο πυρήνας παραγωγής της ερώτησης είναι νευρωνικός και υλοποιείται μέσω fine-tuned μοντέλου T5. Ταυτόχρονα, όμως, δεν ακολουθεί ούτε αμιγώς end-to-end neural λογική, αφού η επιλογή των answer candidates, η αναγνώριση και ομαδοποίηση οντοτήτων, τα

φίλτρα εγκυρότητας, η παραγωγή distractors και το post-processing στηρίζονται σε γλωσσολογική ανάλυση, κανόνες και εξωτερικούς πόρους.

Πιο συγκεκριμένα, το σύστημα αξιοποιεί τη Stanza για tokenization, POS tagging, lemmatization και NER, χρησιμοποιεί ιεραρχικά κριτήρια για την επιλογή της πληροφορίας που θα αποτελέσει απάντηση, εφαρμόζει semantic type filtering για τη διατήρηση της συνοχής των distractors και καταφεύγει σε πόρους όπως το WordNet και το Wikipedia API για τη δημιουργία εναλλακτικών απαντήσεων. Η τελική διατύπωση των ερωτήσεων πραγματοποιείται από το μοντέλο T5, ενώ σε νεότερες εκδοχές του συστήματος προστίθεται και δεύτερο επίπεδο μεταγενέστερης βελτίωσης. Όλα αυτά συνθέτουν μια υβριδική αρχιτεκτονική στην οποία η γλωσσολογική γνώση και η ευρετική λογική συνεργάζονται με τη νευρωνική παραγωγή.

Με βάση τη βιβλιογραφία (Das et al., 2021· Guo et al., 2024) και τα πρακτικά προβλήματα που εμφανίζονται στην AQG, μια αμιγώς rule-based λύση θα ήταν υπερβολικά περιοριστική και δύσκολα θα απέδιδε φυσικές ερωτήσεις. Από την άλλη πλευρά, μια πλήρως νευρωνική προσέγγιση θα μείωνε τον έλεγχο στη διαδικασία επιλογής answer spans, στη δημιουργία distractors και στη γενικότερη εκπαιδευτική αξιοπιστία του αποτελέσματος. Η παρούσα εργασία, επομένως, τοποθετείται συνειδητά στην υβριδική κατηγορία, επιδιώκοντας να εξισορροπήσει τη φυσικότητα της γλωσσικής παραγωγής με την ερμηνευσιμότητα, τον έλεγχο και την πρακτική αξιοποίηση του τελικού συστήματος.

2.5 Παραγωγή Distractors

Η παραγωγή συναφών παραπλανητικών/λανθασμένων επιλογών (distractors) κατά την κατασκευή των single choice ερωτήσεων αποτελεί ένα από τα πιο απαιτητικά προβλήματα στο AQG (Kurdi et al., 2020). Σε αντίθεση με την ίδια την παραγωγή της ερώτησης, όπου ένα μοντέλο μπορεί να αξιοποιήσει το συμφραζόμενο κείμενο για να διατύπωσε μια συντακτικά αποδεκτή πρόταση, οι distractor οφείλουν να είναι αρκετά εύλογοι ώστε να μπορούν να αποκλειστούν με ορισμένα γλωσσολογικά ή εννοιολογικά κριτήρια, αλλά ταυτόχρονα να είναι σαφώς καθορισμένα λανθασμένα. Για την επίτευξη αυτού του εγχειρήματος η παρούσα εργασία υιοθετεί υβριδική στρατηγική και δεν αναθέτει τη δημιουργία distractors αποκλειστικά σε νευρωνικό μοντέλο, αλλά αξιοποιεί

λεξικοσημασιολογικούς πόρους, Named Entity πληροφορία και συμφραζόμενη δεξαμενή οντοτήτων από τις υπόλοιπες προτάσεις του ίδιου του κειμένου.

2.5.1 WordNet

Το WordNet συνιστά έναν από τους κλασικότερους λεξικοσημασιολογικούς πόρους της αγγλικής γλώσσας (Miller, 1995· Fellbaum, 2010). Οι λέξεις οργανώνονται σε synsets, δηλαδή ομάδες εννοιών που εκφράζουν παρόμοιο σημασιολογικό περιεχόμενο, ενώ ταυτόχρονα συνδέονται μεταξύ τους μέσω σχέσεων όπως η υπωνυμία και η υπερωνυμία. Αυτή η δομή καθιστά το WordNet χρήσιμο όχι μόνο για εργασίες κατανόησης γλώσσας, αλλά και για ελεγχόμενη παραγωγή εναλλακτικών όρων.

Στο πρόβλημα των distractors, η αξιοποίηση του WordNet δεν στηρίζεται στην άμεση αναζήτηση συνωνύμων της σωστής απάντησης. Μια τέτοια επιλογή θα ήταν προβληματική, επειδή τα συνώνυμα συχνά οδηγούν σε απαντήσεις πολύ κοντά στη σωστή και άρα παιδαγωγικά ακατάλληλες. Πιο χρήσιμη είναι η λογική των co-hyponyms: εντοπίζεται η ευρύτερη κατηγορία στην οποία ανήκει η απάντηση και στη συνέχεια αναζητούνται άλλοι όροι της ίδιας κατηγορίας. Για παράδειγμα, αν η απάντηση είναι “lion”, η αναζήτηση μπορεί να κινηθεί γύρω από το υπερώνυμο “big cat” ή “feline” και να αντλήσει άλλους όρους του ίδιου επιπέδου, όπως “tiger” ή “leopard”. Έτσι, οι distractors παραμένουν ομότιμοι εννοιολογικά χωρίς να ταυτίζονται με την ορθή απάντηση.

Ωστόσο, το WordNet έχει σαφείς περιορισμούς. Πρώτον, καλύπτει κυρίως την αγγλική γλώσσα και επομένως δεν αποτελεί απευθείας λύση για ελληνικά distractors. Δεύτερον, η κάλυψη κύριων ονομάτων, ιστορικών προσώπων, γεωγραφικών τοπωνυμίων ή πολιτισμικών οντοτήτων είναι περιορισμένη. Τρίτον, σε αρκετές περιπτώσεις η εννοιολογική ιεραρχία είναι υπερβολικά γενική και δεν αρκεί για να παραχθούν πραγματικά πειστικοί distractors σε εκπαιδευτικό πλαίσιο.

2.5.2 Wikipedia API

Εκεί όπου το WordNet είναι σχετικά αδύναμο, ιδιαίτερα σε named entities, η Wikipedia λειτουργεί ως πρακτικός εξωτερικός γνωσιακός πόρος. Η χρήση του MediaWiki API (Wikimedia Foundation, 2024) επιτρέπει την ανάκτηση άρθρων, περιλήψεων και σχετικών οντοτήτων που συνδέονται με ένα πρόσωπο, μια τοποθεσία, έναν οργανισμό ή ένα ιστορικό

γεγονός. Σε αντίθεση με ένα κλειστό λεξικό, η Wikipedia έχει πολύ ευρύτερη κάλυψη σε κύρια ονόματα και οντότητες πραγματικού κόσμου, κάτι που είναι ιδιαίτερος χρήσιμο για κείμενα ιστορίας, γεωγραφίας ή πολιτισμού.

Η τεχνική λογική της αξιοποίησης της Wikipedia είναι η εξής: όταν η ορθή απάντηση ανήκει σε κατηγορία όπως LOC, GPE ή FAC, το σύστημα μπορεί να αναζητήσει το αντίστοιχο λήμμα και να εξαγάγει από το περιβάλλον του άρθρου άλλες οντότητες του ίδιου τύπου. Οι υποψήφιοι αυτοί distractors δεν επιλέγονται αυθαίρετα, αλλά προκύπτουν από ένα συμφραζόμενο το οποίο είναι ήδη εννοιολογικά συγγενές με τη σωστή απάντηση. Αν για παράδειγμα η απάντηση αφορά μία αρχαία πόλη ή μια χώρα, το άρθρο της Wikipedia είναι πιθανό να περιέχει γειτονικές τοποθεσίες, άλλες πολιτικές οντότητες ή συναφή γεωγραφικά στοιχεία, τα οποία μπορούν να λειτουργήσουν ως πειστικές εναλλακτικές επιλογές.

Παρά τα πλεονεκτήματα, υπάρχουν τεχνικές δυσκολίες. Η Wikipedia είναι ανοιχτός και δυναμικός πόρος, οπότε η ποιότητα των αποτελεσμάτων επηρεάζεται από redirects, αποσαφηνίσεις, πολυσημία ονομάτων και γλωσσικές ιδιαιτερότητες. Για παράδειγμα, ένα πρόσωπο μπορεί να αντιστοιχεί σε πολλά λήμματα ή μια αναζήτηση να επιστρέφει μη αναμενόμενα αποτελέσματα. Επιπλέον, το άρθρο μπορεί να περιέχει οντότητες με χαλαρή μόνο συνάφεια με τη σωστή απάντηση

2.5.3 Cross-sentence entity pooling

Η τρίτη και συχνά ιδιαίτερα αποτελεσματική πηγή distractors είναι η δεξαμενή οντοτήτων που προκύπτει από το ίδιο το κείμενο εισόδου. Αντί το σύστημα να περιορίζεται στη μεμονωμένη πρόταση από την οποία παράγεται η ερώτηση, μπορεί να αξιοποιεί όλες τις οντότητες που εντοπίστηκαν σε ολόκληρο το έγγραφο. Η στρατηγική αυτή ονομάζεται εδώ cross-sentence entity pooling και στηρίζεται στην ιδέα ότι το ίδιο το διδακτικό κείμενο περιέχει συνήθως ήδη ένα θεματικά συνεκτικό σύνολο προσώπων, τόπων, ημερομηνιών ή γεγονότων.

Η τεχνική αξία αυτής της προσέγγισης είναι σημαντική. Σε ιστορικό κείμενο παραδείγματος, για παράδειγμα, αν εμφανίζονται πολλαπλά πρόσωπα ή τοπωνύμια, αυτά αποτελούν ισχυρούς υποψήφιους distractors για κάθε σχετική ερώτηση. Η χρήση τέτοιων οντοτήτων αυξάνει τη θεματική συνοχή του quiz και περιορίζει την ανάγκη εξωτερικής

αναζήτησης. Επιπλέον, επειδή οι οντότητες προέρχονται από το ίδιο πηγείο υλικό, μειώνεται ο κίνδυνος εισαγωγής εξωτερικής γνώσης που θα αποσυνδέει την ερώτηση από το κείμενο αναφοράς.

Το cross-sentence pooling είναι επίσης χρήσιμο ως fallback μηχανισμός. Υπάρχουν περιπτώσεις όπου ούτε το WordNet ούτε η Wikipedia επιστρέφουν επαρκείς ή κατάλληλους distractors. Ιδίως σε μικρά ή εξειδικευμένα κείμενα, η πιο αξιόπιστη πηγή μπορεί να είναι οι ίδιες οι οντότητες του κειμένου. Η στρατηγική αυτή είναι συνεπής με την υβριδική αρχιτεκτονική της εργασίας, καθώς συνδυάζει information extraction από τη Stanza με downstream logic επιλογής distractors.

2.6 Νευρωνική Μηχανική Μετάφραση (NMT)

Η Νευρωνική Μηχανική Μετάφραση (Neural Machine Translation-NMT) βασίζεται σε encoder-decoder μοντέλα (Sutskever et al., 2014· Vaswani et al., 2017) που μαθαίνουν να αντιστοιχούν ακολουθίες από μία γλώσσα σε ακολουθίες μιας άλλης. Σε αντίθεση με τις παλαιότερες στατιστικές ή phrase-based προσεγγίσεις, τα NMT μοντέλα μαθαίνουν συνεχή, συμφραζόμενη αναπαράσταση ολόκληρης της πρότασης και παράγουν το μεταφρασμένο κείμενο token προς token. Η μετάβαση αυτή βελτίωσε σημαντικά τη φυσικότητα και τη συνοχή της αυτόματης μετάφρασης, ιδίως μετά την υιοθέτηση των Transformers.

Στο πλαίσιο της παρούσας εργασίας, η NMT δεν αποτελεί αυτοσκοπό αλλά υποστηρικτικό στρώμα της AQG αρχιτεκτονικής. Η μετάφραση χρησιμοποιείται ως τεχνολογικός καταλύτης που επιτρέπει σε ένα αγγλοκεντρικό pipeline να επεκταθεί σε ελληνικά κείμενα χωρίς να επανασχεδιαστούν όλα τα υποσυστήματα. Αυτή η λογική είναι σημαντική γιατί διαχωρίζει τη μετάφραση από το ίδιο το έργο της ερώτησης: η NMT δεν δημιουργεί ερώτηση, αλλά επιτρέπει στα άλλα εργαλεία να λειτουργήσουν σε γλώσσα για την οποία είναι πιο ώριμα.

2.6.1 MarianMT

Το Marian NMT αποτελεί ένα ολοκληρωμένο πλαίσιο εργασίας (framework) (Junczys-Dowmunt et al., 2018) για την υλοποίηση Νευρωνικής Μηχανικής Μετάφρασης (Neural Machine Translation - NMT), αναπτυγμένο κυρίως από την ομάδα επεξεργασίας φυσικής

γλώσσας της Microsoft και του Πανεπιστημίου του Εδιμβούργου. Η αρχιτεκτονική του εδράζεται στο μοντέλο Transformer, το οποίο βασίζεται στον μηχανισμό της προσοχής (attention mechanism), επιτρέποντας την αποτελεσματική διαχείριση εξαρτήσεων μεγάλου εύρους στο κείμενο. Η υλοποίησή του σε C++ προσφέρει σημαντικά πλεονεκτήματα ως προς την υπολογιστική απόδοση και την ταχύτητα εκπαίδευσης, καθιστώντας το μία από τις πιο αποδοτικές λύσεις για τη μετάφραση σε πραγματικό χρόνο.

2.6.2 Μοντέλα Helsinki-NLP

Τα μοντέλα Helsinki-NLP/opus-mt αποτελούν fine-tuned παραλλαγές για εκατοντάδες ζεύγη γλωσσών, διαθέσιμες ελεύθερα μέσω του Hugging Face Hub. Βασίζονται σε Transformer encoder-decoder αρχιτεκτονική και εκπαιδεύτηκαν στο OPUS corpus (Tiedemann & Thottingal, 2020), ένα από τα μεγαλύτερα διαθέσιμα σώματα παράλληλων κειμένων.

Για το ζεύγος ελληνικά-αγγλικά, το Helsinki-NLP/opus-mt-tc-big-el-en επιτυγχάνει BLEU score 68.8, το οποίο αντιστοιχεί σε υψηλή ποιότητα μετάφρασης για γενικά κείμενα. Για το αντίστροφο ζεύγος (αγγλικά-ελληνικά), το Helsinki-NLP/opus-mt-en-el επιτυγχάνει BLEU score 56.4, αποδεκτό επίπεδο για τον σκοπό της εργασίας, παρότι χαμηλότερο από την αντίστροφη κατεύθυνση (Helsinki-NLP, n.d.). Και τα δύο μοντέλα εκτελούνται τοπικά χωρίς σύνδεση στο διαδίκτυο, σε συμφωνία με την αρχική offline-first αρχιτεκτονική.

2.7 Βελτίωση Παραγόμενου Κειμένου

Ένα από τα ζητήματα που προέκυψαν κατά την σύνθεση της offline κυρίως υβριδικής αρχιτεκτονικής με σκοπό την παραγωγή ερωτήσεων καθώς και τη σύνθεση των distractors στις single choice ερωτήσεις, ήταν πως η ποιότητα τόσο των αγγλικών και των ελληνικών προϊόντων υστερούσαν λόγω των εγγενών ορίων των μοντέλων που χρησιμοποιήθηκαν. Ειδικότερα, όταν ένα offline μοντέλο είναι διαθέσιμο μόνο σε προεκπαιδευμένη ή περιορισμένα fine-tuned μορφή, το εύρος, το ύψος και η ακρίβεια των παραγόμενων προϊόντων παραμένουν αναγκαστικά εντός των δυνατοτήτων για τις οποίες έχει ήδη προσαρμοστεί.

Για τον λόγο αυτό αποφασίστηκε η ενσωμάτωση ενός δεύτερου στρώματος ανάλυσης και βελτίωσης του pipeline με την προσθήκη ενός Μεγάλου Γλωσσικού Μοντέλου (Large Language Model — LLM).

2.7.1 LLM-based post-processing

Το post-processing με τη συνδρομή LLM αποσκοπεί στην αντιμετώπιση τεσσάρων κατηγοριών προβλημάτων: ερωτήσεων που αποκαλύπτουν την απάντησή τους μέσα στο κείμενό τους (tautology), ερωτήσεων με γραμματικά σφάλματα, ανεπαρκών distractors και ερωτήσεων με hallucinations.

Συγκεκριμένα το φαινόμενο των hallucinations στα γλωσσικά μοντέλα (Ji et al., 2023) αναφέρεται στην παραγωγή πληροφοριών που δεν βρίσκονται στο κείμενο εισόδου αλλά αντλούνται από τα δεδομένα προεκπαίδευσης. Στο πλαίσιο του AQG, αυτό εκδηλώνεται ως ερωτήσεις που αναφέρονται σε πρόσωπα, γεγονότα ή έργα που δεν εμφανίζονται στο κείμενο — ένα σοβαρό πρόβλημα για εκπαιδευτικές εφαρμογές όπου η ακρίβεια ως προς το υλικό αναφοράς είναι κρίσιμη.

2.7.2 Επιλογή LLM

Στο πλαίσιο της παρούσας εργασίας, το Groq επιλέχθηκε ως η πρακτικότερη λύση για την ενσωμάτωση LLM-based post-processing (Groq, n.d.), καθώς παρείχε άμεση πρόσβαση μέσω API key, δωρεάν επίπεδο χρήσης και χαμηλό εμπόδιο εισόδου για τον τελικό χρήστη. Η επιλογή αυτή κρίθηκε ιδιαίτερα κατάλληλη για ένα πειραματικό και εκπαιδευτικό σύστημα, όπου ήταν σημαντικό ο χρήστης να μπορεί να εισαγάγει εύκολα το δικό του API key χωρίς σύνθετες διαδικασίες ενεργοποίησης ή αρχική οικονομική επιβάρυνση.

Από τεχνολογική άποψη, η σημασία του δεν περιορίζεται μόνο στη φιλοξενία μοντέλων, αλλά και στην ικανότητά του να προσφέρει γρήγορη εκτέλεση σε εργασίες γλωσσικής επεξεργασίας, γεγονός που το καθιστά ιδιαίτερα κατάλληλο για χρήση ως μεταγενέστερο στάδιο σε pipelines που απαιτούν επιπλέον ποιοτικό έλεγχο.

2.7.3 Επιλογή μοντέλου Groq

Για το στάδιο του post-processing αξιοποιείται μοντέλο της οικογένειας Llama, συγκεκριμένα το llama-3.1-8b-instant (Grattafiori et al., 2024), το οποίο χρησιμοποιείται ως μηχανισμός αναδιατύπωσης, ελέγχου και επιλογής ποιοτικότερης εξόδου. Η επιλογή ενός τέτοιου μοντέλου βασίζεται στο γεγονός ότι διαθέτει μεγαλύτερη γλωσσική ευελιξία από τα στενότερα fine-tuned question generation μοντέλα και μπορεί να εξετάσει μια ήδη παραγμένη ερώτηση όχι μόνο ως ακολουθία λέξεων, αλλά και ως γλωσσικό προϊόν που πρέπει να είναι σαφές, φυσικό και συνεπές ως προς το συμφραζόμενο.

Η χρησιμότητα του μοντέλου σε αυτή την εργασία συνδέεται κυρίως με τη δυνατότητά του να επαναξιολογεί την έξοδο του βασικού pipeline και να παρεμβαίνει σε περιπτώσεις όπου απαιτείται βελτίωση με κατάλληλο prompting. Έτσι, δεν χρησιμοποιείται ως κύριος generator του συστήματος, αλλά ως δευτερογενές φίλτρο ποιότητας.

2.8 Learning Management Systems (LMS)

Τα Συστήματα Διαχείρισης Μάθησης, γνωστά ως Learning Management Systems (LMS), αποτελούν βασική τεχνολογική υποδομή της ηλεκτρονικής και μικτής μάθησης. Ένα LMS επιτρέπει την οργάνωση μαθημάτων, τη διάθεση εκπαιδευτικού υλικού, τη διαχείριση χρηστών, την παρακολούθηση της μαθησιακής πορείας και την υποστήριξη δραστηριοτήτων αξιολόγησης. Στο πλαίσιο της σύγχρονης εκπαίδευσης, τα LMS δεν λειτουργούν απλώς ως αποθετήρια αρχείων, αλλά ως ολοκληρωμένα περιβάλλοντα στα οποία συνδυάζονται εκπαιδευτικό περιεχόμενο, επικοινωνία, αλληλεπίδραση και αξιολόγηση. Σύμφωνα με τη βιβλιογραφία (Gamage et al., 2022), τα LMS γεφυρώνουν τις παραδοσιακές διδακτικές πρακτικές με τους ψηφιακούς μαθησιακούς πόρους, προσφέροντας δυνατότητες εξατομικευμένης ηλεκτρονικής μάθησης και υποστήριξης σύγχρονων ή ασύγχρονων εκπαιδευτικών διαδικασιών.

2.8.1 Moodle

Στην κατηγορία αυτή των LMS, το Moodle αποτελεί μία από τις σημαντικότερες και πιο διαδεδομένες πλατφόρμες ανοικτού κώδικα. Η αρχική φιλοσοφία του Moodle συνδέθηκε με θεωρητικές προσεγγίσεις κοινωνικού εποικοδομισμού (Dougiamas & Taylor, 2003), σύμφωνα με τις οποίες η μάθηση δεν αντιμετωπίζεται ως απλή μετάδοση πληροφορίας,

αλλά ως διαδικασία συνεργατικής οικοδόμησης γνώσης μέσα από κοινότητες μάθησης. Σύμφωνα με τη βιβλιογραφία, το Moodle ως ένα ανοικτού κώδικα σύστημα διαχείρισης μαθημάτων, σχεδιασμένο με βάση παιδαγωγικές αρχές και με έμφαση στη συμμετοχή της κοινότητας χρηστών και προγραμματιστών. Η σύνδεση της τεχνολογικής αρχιτεκτονικής με παιδαγωγικές αρχές καθιστά το Moodle ιδιαίτερα κατάλληλο για εκπαιδευτικά περιβάλλοντα όπου ζητείται όχι μόνο διανομή περιεχομένου, αλλά και ενεργή συμμετοχή των εκπαιδευομένων.

Η σύγχρονη ερευνητική βιβλιογραφία επιβεβαιώνει τη σημασία του Moodle στην ανώτατη εκπαίδευση και ειδικότερα σε περιβάλλοντα STEM, μικτής μάθησης και διαδικτυακής αξιολόγησης. Η συστηματική ανασκόπηση σύμφωνα με τη βιβλιογραφία (Gamage et al., 2022) καταγράφει ότι το Moodle χρησιμοποιείται ευρέως στη διδασκαλία και μάθηση, ενώ σχετίζεται με βελτιώσεις στην επίδοση, στην ικανοποίηση και στην εμπλοκή των φοιτητών. Η ίδια μελέτη επισημαίνει ότι το Moodle εξελίσσεται ως πλατφόρμα υποστήριξης συνεργατικής μάθησης, προσαρμοστικών δραστηριοτήτων και διαδικτυακών αξιολογήσεων.

Για την παρούσα εργασία, το Moodle παρουσιάζει ιδιαίτερο ενδιαφέρον επειδή λειτουργεί ως πραγματικό περιβάλλον υποδοχής των παραγόμενων διαδραστικών αντικειμένων. Η εφαρμογή που αναπτύσσεται δεν έχει ως στόχο να αντικαταστήσει ένα LMS, αλλά να παράγει εκπαιδευτικά αντικείμενα σε μορφή H5P, τα οποία μπορούν να ενσωματωθούν σε υπάρχουσες πλατφόρμες μάθησης. Με αυτόν τον τρόπο, το σύστημα αυτόματης δημιουργίας ερωτήσεων λειτουργεί ως εργαλείο παραγωγής περιεχομένου, ενώ το Moodle αναλαμβάνει τον ρόλο της διάθεσης, οργάνωσης, χρήσης και πιθανής παρακολούθησης των δραστηριοτήτων από τους εκπαιδευόμενους.

2.8.2 H5P

Η ενσωμάτωση του H5P στο Moodle είναι σημαντική, καθώς το H5P επιτρέπει τη δημιουργία διαδραστικού περιεχομένου, όπως κουίζ, παρουσιάσεις και διαδραστικά βίντεο. Σύμφωνα με την επίσημη τεκμηρίωση του Moodle (Moodle, n.d.), το H5P είναι συντομογραφία του HTML5 Package και μπορεί να δημιουργηθεί είτε μέσα από το Content Bank του Moodle είτε σε εξωτερικά εργαλεία, όπως το H5P.com ή το Lumi Desktop, και

στη συνέχεια να προστεθεί σε μάθημα ως H5P activity ή να ενσωματωθεί σε άλλες δραστηριότητες και πόρους.

Η λειτουργία Content Bank του Moodle αποτελεί βασικό σημείο διαχείρισης H5P αντικειμένων. Ο εκπαιδευτής μπορεί να δημιουργήσει ή να επεξεργαστεί H5P περιεχόμενο μέσα από το Content Bank, να επιλέξει τύπο δραστηριότητας και στη συνέχεια να το προσθέσει σε μάθημα. Παράλληλα, υπάρχει η δυνατότητα εισαγωγής αρχείων .h5p που έχουν δημιουργηθεί σε εξωτερικά εργαλεία. Αυτό είναι κρίσιμο για την παρούσα εργασία, διότι το σύστημα AQG εξάγει αρχεία H5P εκτός Moodle, τα οποία πρέπει να μπορούν να εισαχθούν και να εκτελεστούν σε πραγματικό LMS περιβάλλον. Η τεκμηρίωση του Moodle επισημαίνει επίσης ότι, για να λειτουργούν σωστά τα ανεβασμένα .h5p αρχεία, ο διακομιστής Moodle πρέπει να διαθέτει εγκατεστημένες τις αντίστοιχες H5P βιβλιοθήκες.

Από τεχνική άποψη, το H5P λειτουργεί ως πακέτο περιεχομένου. Ένα αρχείο .h5p βασίζεται σε μεταδεδομένα, σε ορισμούς βιβλιοθηκών και σε αρχεία περιεχομένου. Η επίσημη τεκμηρίωση του H5P (H5P Group, n.d.) αναφέρει ότι το αρχείο h5p.json περιλαμβάνει βασικά μεταδεδομένα και εξαρτήσεις βιβλιοθηκών, όπως τον τίτλο, την κύρια βιβλιοθήκη, τη γλώσσα, τις προφορτωμένες εξαρτήσεις και τους τύπους ενσωμάτωσης. Επιπλέον, η δομή περιεχομένου μπορεί να περιλαμβάνει το αρχείο content.json, στο οποίο αποθηκεύονται οι παράμετροι του διαδραστικού αντικειμένου, καθώς και οι βιβλιοθήκες που είναι απαραίτητες για την εκτέλεσή του.

Για την παραγωγή των διαδραστικών αντικειμένων H5P επιλέχθηκαν δύο τύποι ερωτήσεων, οι ερωτήσεις πολλαπλής επιλογής (Single Choice) και οι ερωτήσεις συμπλήρωσης κενού (Fill in the Blanks), καθώς συνδυάζουν αυτόματη παραγωγή, αξιόπιστη αυτόματη βαθμολόγηση και συμπληρωματική γνωστική στόχευση. Οι δύο αυτοί τύποι δοκιμάζουν διακριτά επίπεδα επεξεργασίας: οι ερωτήσεις πολλαπλής επιλογής αξιολογούν την αναγνώριση (recognition) της σωστής απάντησης μεταξύ εναλλακτικών, ενώ οι ερωτήσεις συμπλήρωσης κενού απαιτούν ανάκληση (recall) και ελεύθερη παραγωγή της απάντησης, που αποτελεί γνωστικά αυστηρότερη διαδικασία. Καθοριστικό κριτήριο επιλογής αποτέλεσε η αντικειμενική βαθμολόγηση: αμφότεροι οι τύποι έχουν σαφώς ορισμένη σωστή απάντηση, επιτρέποντας στο H5P να αξιολογεί αυτόματα την απόκριση του μαθητή χωρίς ανθρώπινη παρέμβαση. Αντιθέτως, τύποι όπως οι ανοιχτές ερωτήσεις ανάπτυξης ή οι ερωτήσεις σύντομης απάντησης απαιτούν σημασιολογική κρίση για τη βαθμολόγηση, κάτι που δεν μπορεί να εξασφαλιστεί αξιόπιστα με αυτόματο τρόπο. Άλλοι

διαδραστικοί τύποι, όπως οι ερωτήσεις αντιστοίχισης (matching) ή σειροθέτησης (ordering), προϋποθέτουν την εξαγωγή πολλαπλών αλληλοσχετιζόμενων ζευγών ή σχέσεων διάταξης από το κείμενο, διαδικασία σημαντικά πιο σύνθετη και επιρρεπή σε σφάλματα κατά την αυτόματη παραγωγή.

2.8.3 H5P.Single Choice Set

Οι τύποι δραστηριοτήτων που αξιοποιούνται στην παρούσα εργασία, όπως το H5P.Single Choice Set και το H5P.Fill In The Blanks, συνδέονται άμεσα με τη λογική της αυτόματης δημιουργίας ερωτήσεων. Το Single Choice Set επιτρέπει τη δημιουργία συνόλων ερωτήσεων με μία σωστή απάντηση ανά ερώτηση και προσφέρει άμεση ανατροφοδότηση στον τελικό χρήστη. Η τεκμηρίωση του H5P (H5P Group, n.d.) διευκρινίζει ότι η πρώτη εναλλακτική απάντηση που εισάγεται στον editor θεωρείται η σωστή, ενώ η σειρά των απαντήσεων ανακατεύεται στην προβολή του εκπαιδευόμενου. Αυτό έχει άμεση τεχνική σημασία για το σύστημα εξαγωγής, καθώς η σωστή απάντηση πρέπει να τοποθετείται στη θέση που απαιτεί η δομή του H5P, ανεξάρτητα από τον τρόπο που θα εμφανιστεί τελικά στον χρήστη

2.8.4 H5P.Fill In The Blanks

Αντίστοιχα, το Fill in the Blanks χρησιμοποιείται για δραστηριότητες συμπλήρωσης κενού, στις οποίες ο εκπαιδευόμενος καλείται να αναπαραγάγει μία λέξη ή φράση που λείπει από το κείμενο. Η τεκμηρίωση του H5P (H5P Group, n.d.) περιγράφει τον συγκεκριμένο τύπο ως HTML5 question type για cloze tests, ο οποίος μπορεί να χρησιμοποιηθεί σε συστήματα όπως Moodle, Canvas, Brightspace, Blackboard και WordPress. Για τους συγγραφείς περιεχομένου, οι λέξεις που πρέπει να μετατραπούν σε κενά δηλώνονται με ειδική σύνταξη, ενώ για τον εκπαιδευόμενο παρουσιάζονται ως πεδία συμπλήρωσης. Αυτό καθιστά τον τύπο αυτό κατάλληλο για την αυτόματη παραγωγή ερωτήσεων από εκπαιδευτικό κείμενο, καθώς η επιλογή μιας σημαντικής λέξης ή έννοιας μπορεί να μετατραπεί απευθείας σε δραστηριότητα συμπλήρωσης κενού

2.8.5 Lumi Desktop

Το Lumi Desktop χρησιμοποιείται στην παρούσα εργασία ως εργαλείο τοπικής δημιουργίας, επεξεργασίας και δοκιμής H5P αρχείων. Πρόκειται για εφαρμογή επιφάνειας εργασίας που επιτρέπει στον χρήστη να δημιουργεί, να επεξεργάζεται και να προεπισκοπεί H5P περιεχόμενο χωρίς να απαιτείται σύνδεση στο διαδίκτυο. Η επίσημη ιστοσελίδα του Lumi (Lumi Education, n.d.) το περιγράφει ως δωρεάν και ανοικτού κώδικα εργαλείο, διαθέσιμο για Windows, macOS και Linux. Η δυνατότητα τοπικής προεπισκόπησης είναι ιδιαίτερα χρήσιμη για την ανάπτυξη της παρούσας εφαρμογής, επειδή επιτρέπει τον έλεγχο των παραγόμενων .h5p αρχείων πριν αυτά εισαχθούν σε ένα LMS όπως το Moodle.

Η χρήση του Lumi Desktop προσφέρει πρακτικά πλεονεκτήματα κατά τη φάση ανάπτυξης. Πρώτον, επιτρέπει τον γρήγορο έλεγχο της ορθότητας του παραγόμενου H5P πακέτου, χωρίς να απαιτείται κάθε φορά εισαγωγή σε Moodle. Δεύτερον, βοηθά στον εντοπισμό προβλημάτων στη δομή του content.json, στη συμβατότητα βιβλιοθηκών ή στη σύνταξη των ερωτήσεων. Τρίτον, λειτουργεί ως ενδιάμεσο περιβάλλον επιβεβαίωσης ότι το αποτέλεσμα της εφαρμογής είναι αξιοποιήσιμο από εκπαιδευτικό χρήστη. Για τον λόγο αυτό, το Lumi Desktop χρησιμοποιείται όχι ως τελικό LMS, αλλά ως εργαλείο ελέγχου και προεπισκόπησης πριν την εισαγωγή του περιεχομένου σε Moodle.

2.9 Μετρικές Αξιολόγησης Παραγόμενου Κειμένου

Η αξιολόγηση παραγόμενου κειμένου αποτελεί ένα από τα πιο σύνθετα ζητήματα στην Επεξεργασία Φυσικής Γλώσσας, καθώς η ποιότητα ενός παραγόμενου αποτελέσματος δεν εξαρτάται μόνο από την ορθότητα της σύνταξης, αλλά και από τη σημασιολογική ακρίβεια, τη φυσικότητα της διατύπωσης και τη σχέση του παραγόμενου κειμένου με το αρχικό κείμενο πηγής. Στην περίπτωση της αυτόματης δημιουργίας ερωτήσεων, το πρόβλημα γίνεται ακόμη πιο απαιτητικό, διότι μια ερώτηση μπορεί να είναι γραμματικά σωστή, αλλά να μην είναι παιδαγωγικά χρήσιμη, να μην αντιστοιχεί σε σημαντική πληροφορία του κειμένου ή να μην οδηγεί με σαφήνεια στη σωστή απάντηση. Για τον λόγο αυτό, η αξιολόγηση των συστημάτων AQG συνήθως συνδυάζει αυτόματες μετρικές σύγκρισης κειμένου με ποιοτική ανάλυση των παραγόμενων ερωτήσεων.

Οι αυτόματες μετρικές, όπως το BLEU και το ROUGE, χρησιμοποιούνται ευρέως σε εργασίες παραγωγής φυσικής γλώσσας, επειδή επιτρέπουν τη γρήγορη και επαναλήψιμη

σύγκριση μεταξύ παραγόμενου κειμένου και ανθρώπινων ερωτήσεων αναφοράς. Η βασική ιδέα είναι ότι το σύστημα παράγει μια ερώτηση και αυτή συγκρίνεται με μία ή περισσότερες ερωτήσεις που έχουν δημιουργηθεί από άνθρωπο. Όσο μεγαλύτερη είναι η λεκτική ή φραστική επικάλυψη μεταξύ της παραγόμενης ερώτησης και της ερώτησης αναφοράς, τόσο υψηλότερη είναι συνήθως η τιμή της μετρικής. Αυτή η προσέγγιση είναι πρακτική, ιδίως όταν υπάρχουν gold questions, δηλαδή ερωτήσεις αναφοράς, αλλά δεν ταυτίζεται πλήρως με την ανθρώπινη αξιολόγηση της ποιότητας. Στη βιβλιογραφία της AQG (Guo et al., 2024· Mulla & Gharpure, 2023) αναφέρεται ότι οι κλασικές μετρικές, όπως BLEU, ROUGE, METEOR και CIDEr [(Banerjee & Lavie, 2005) / (Vedantam et al., 2015)], μπορούν να χρησιμοποιηθούν για την αυτόματη αξιολόγηση ερωτήσεων, αλλά δεν αρκούν πάντα για να εκτιμήσουν στοιχεία όπως φυσικότητα, γραμματική, συντακτική ποιότητα και παιδαγωγική καταλληλότητα.

2.9.1 Bilingual Evaluation Understudy (BLEU)

Η αρχική του χρήση του BLEU (Bilingual Evaluation Understudy) (Papineni et al., 2002) ήταν η σύγκριση μιας παραγόμενης μετάφρασης με μία ή περισσότερες ανθρώπινες μεταφράσεις αναφοράς. Η βασική λογική του είναι ότι ένα παραγόμενο κείμενο θεωρείται ποιοτικότερο όταν μοιράζεται περισσότερες κοινές ακολουθίες λέξεων (n-grams) με το κείμενο αναφοράς. Τα n-grams μπορεί να είναι μεμονωμένες λέξεις, ζεύγη λέξεων, τριάδες λέξεων ή μεγαλύτερες ακολουθίες. Για παράδειγμα, στο επίπεδο 1-gram εξετάζεται η επικάλυψη μεμονωμένων λέξεων, ενώ στο επίπεδο 2-gram ή 3-gram εξετάζεται κατά πόσο το παραγόμενο κείμενο αναπαράγει μικρές φράσεις που υπάρχουν και στο κείμενο αναφοράς. Με αυτόν τον τρόπο, το BLEU δεν εξετάζει μόνο αν χρησιμοποιούνται ίδιες λέξεις, αλλά και αν αυτές εμφανίζονται σε παρόμοιους συνδυασμούς.

Από τη φύση της, η BLEU είναι μετρική προσανατολισμένη στην ακρίβεια (precision): ρωτά πόσο από αυτό που παρήγαγε το σύστημα υπάρχει και στην αναφορά. Επειδή ένα τέτοιο κριτήριο θα μπορούσε να «εξαπατηθεί» από πολύ σύντομες εξόδους με λίγες αλλά κοινές λέξεις, η μετρική ενσωματώνει ποινή συντομίας (brevity penalty) που μειώνει τη βαθμολογία όταν η έξοδος είναι δυσανάλογα μικρή σε σχέση με την αναφορά. Στο πλαίσιο της παραγωγής ερωτήσεων, αυτό σημαίνει ότι μια υπερβολικά λακωνική ερώτηση δεν μπορεί να κερδίσει τεχνητά υψηλό σκορ.

2.9.2 Recall-Oriented Understudy for Gisting Evaluation (ROUGE)

Το ROUGE (Recall-Oriented Understudy for Gisting Evaluation), προτάθηκε αρχικά ως πακέτο μετρικών για την αυτόματη αξιολόγηση περιλήψεων (Lin, 2004). Η βασική ιδέα του ROUGE είναι η σύγκριση ενός παραγόμενου κειμένου με ένα ή περισσότερα κείμενα αναφοράς, με βάση την επικάλυψη λέξεων, ακολουθιών λέξεων και άλλων μονάδων κειμένου.

Σε αντίθεση με το BLEU, το οποίο δίνει έμφαση στο precision, το ROUGE συνδέεται περισσότερο με τη λογική του recall. Δηλαδή, εξετάζει σε ποιο βαθμό οι πληροφορίες που υπάρχουν στο κείμενο αναφοράς καλύπτονται από το παραγόμενο κείμενο. Αν και το ROUGE σχεδιάστηκε αρχικά για αξιολόγηση περιλήψεων, χρησιμοποιείται ευρέως και σε άλλες εργασίες παραγωγής κειμένου, όπως η παραγωγή ερωτήσεων, επειδή επιτρέπει την ποσοτική σύγκριση μεταξύ παραγόμενων και ανθρώπινων διατυπώσεων.

Οι βασικές παραλλαγές που χρησιμοποιούνται στην παρούσα εργασία είναι οι ROUGE-1, ROUGE-2 και ROUGE-L. Το ROUGE-1 υπολογίζει την επικάλυψη μεμονωμένων λέξεων μεταξύ παραγόμενης ερώτησης και ερώτησης αναφοράς. Το ROUGE-2 υπολογίζει την επικάλυψη ζευγών διαδοχικών λέξεων, δηλαδή bigrams, και είναι πιο αυστηρό, επειδή απαιτεί μεγαλύτερη ομοιότητα στη φραστική διατύπωση. Το ROUGE-L βασίζεται στη μεγαλύτερη κοινή υποακολουθία, δηλαδή Longest Common Subsequence, και αξιολογεί κατά πόσο δύο κείμενα έχουν παρόμοια σειρά λέξεων χωρίς να απαιτείται απαραίτητα συνεχόμενη αντιστοίχιση.

Στο πλαίσιο της παρούσας εργασίας, το ROUGE είναι ιδιαίτερα χρήσιμο επειδή επιτρέπει τη σύγκριση παραγόμενων ερωτήσεων με gold questions σε επίπεδο λέξεων, φράσεων και δομής. Τα αποτελέσματα ROUGE δεν ερμηνεύονται ως απόλυτη μέτρηση παιδαγωγικής ποιότητας, αλλά ως ένδειξη εγγύτητας προς τις ανθρώπινες ερωτήσεις αναφοράς.

3. Σχεδίαση

3.1 Αρχιτεκτονική

Το προτεινόμενο σύστημα αυτόματης παραγωγής ερωτήσεων (Automatic Question Generation – AQG) σχεδιάστηκε ως ένα υβριδικό pipeline που συνδυάζει γλωσσολογικά

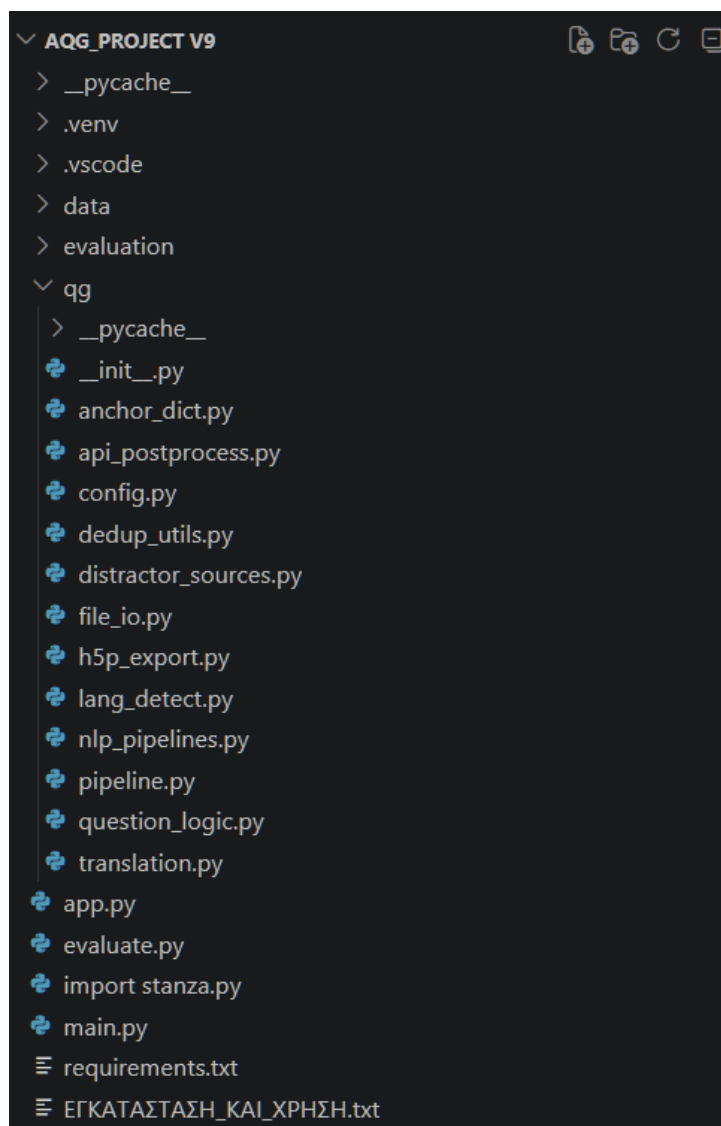
εργαλεία, νευρωνικά μοντέλα ακολουθίας-σε-ακολουθία (seq-2-seq) και ένα τελικό στάδιο μεταγενέστερης βελτίωσης με Large Language Model (LLM). Στόχος της αρχιτεκτονικής είναι η αξιοποίηση των ισχυρών σημείων του καθενός μηχανισμού σε διακριτά στάδια της παραγωγής, ώστε το τελικό αποτέλεσμα να είναι μια πλήρως χρησιμοποιήσιμη ερώτηση πολλαπλής επιλογής ή συμπλήρωσης κενού σε μορφή H5P, έτοιμη για ενσωμάτωση στο Moodle.

3.1.1 Φιλοσοφία σχεδιασμού

Η βασική σχεδιαστική απόφαση που διέπει όλο το σύστημα είναι η διαχωρισμένη ευθύνη ανά υποσύστημα που έχει έναν σαφή, περιορισμένο ρόλο:

- Η γλωσσολογική ανάλυση (Stanza) απομονώνει υποψήφιες απαντήσεις και παρέχει συντακτικά και μορφολογικά χαρακτηριστικά. Δεν παράγει ερωτήσεις, δεν επιλέγει distractors – απλώς εφοδιάζει τα επόμενα στάδια με δομημένη πληροφορία.
- Το νευρωνικό μοντέλο T5 είναι υπεύθυνο για τη μετατροπή μιας πρότασης και μιας επιλεγμένης απάντησης, που θα διοχετευθούν από τα άλλα υποσυστήματα, σε μια συντακτικά ορθή ερώτηση. Δεν αποφασίζει τι θα ρωτηθεί, ούτε τι λάθος επιλογές θα προταθούν.
- Το σύστημα παραγωγής distractors συνδυάζει WordNet (σημασιολογική συγγένεια) και Wikipedia API (θεματική συγγένεια), με σαφή ιεραρχία προτεραιοτήτων.
- Το LLM post-processing (Groq API με Llama-3.1) λειτουργεί ως δεύτερο επίπεδο ποιοτικού ελέγχου, αναλαμβάνοντας ό,τι τα προηγούμενα εργαλεία δεν μπορούν να εντοπίσουν αξιόπιστα (πχ ταυτολογίες, γραμματικά λάθη και νοηματικές ασυνέπειες).

Αυτός ο διαχωρισμός επιτρέπει την ανεξάρτητη βελτίωση κάθε υποσυστήματος και διευκολύνει τον εντοπισμό σφαλμάτων: όταν προκύπτει πρόβλημα ποιότητας, μπορεί να εντοπιστεί συγκεκριμένα σε ποιο στάδιο ή module γεννήθηκε.

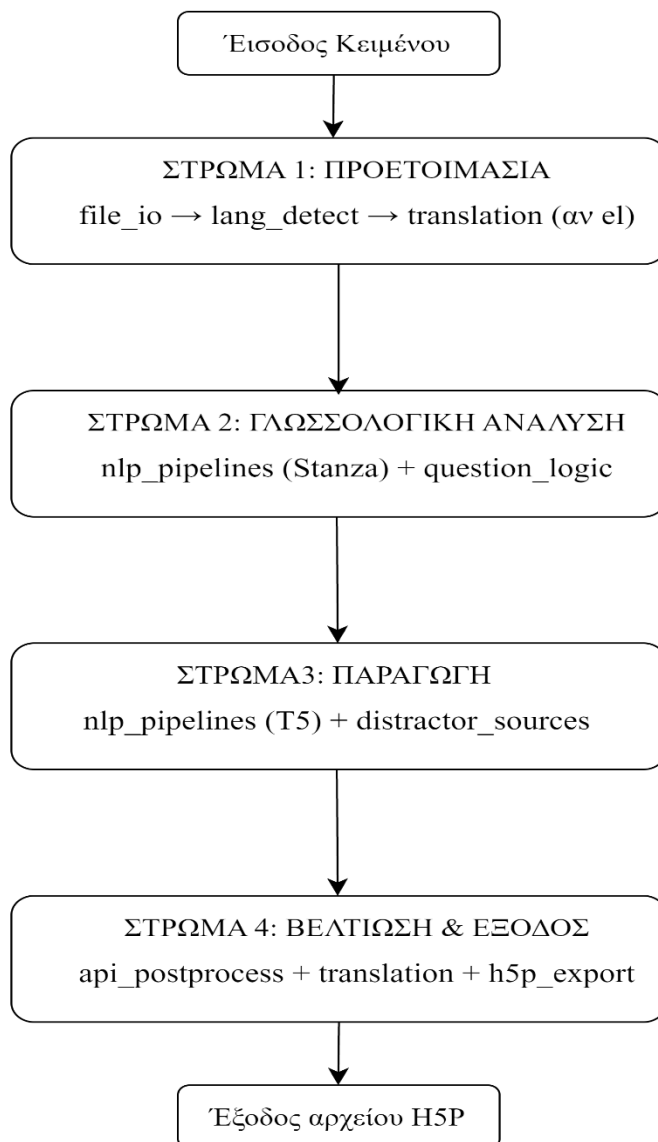


Εικόνα 1 Διάρθρωση του πυρήνα του συστήματος.

Η διάρθρωση αυτή φαίνεται και στην οργάνωση του κώδικα όπου ο πυρήνας του συστήματος βρίσκεται στον φάκελο qg/, με έντεκα διακριτά modules που το καθένα έχει συγκεκριμένη αρμοδιότητα.

3.1.2 Τα 4 Στρώματα Οργάνωσης

Η αρχιτεκτονική οργανώνεται σε τέσσερα λογικά στρώματα, καθένα με συγκεκριμένη ευθύνη. Τα στρώματα εκτελούνται σειριακά για κάθε κείμενο εισόδου, με την έξοδο του ενός να γίνεται είσοδος του επόμενου.



Διάγραμμα 1 Τα τέσσερα στρώματα του αγωγού

Το **Στρώμα Προετοιμασίας** χειρίζεται την είσοδο: φόρτωση αρχείου (.txt ή .pdf), ανίχνευση γλώσσας, και αν χρειάζεται μετάφραση στα αγγλικά. Αν η είσοδος είναι αγγλική, το στρώμα περιορίζεται σε φόρτωση ενώ η ανίχνευση και η μετάφραση παρακάμπτεται.

Το **Στρώμα Γλωσσολογικής Ανάλυσης** εκτελεί tokenization, POS tagging, lemmatization και NER μέσω της βιβλιοθήκης Stanza, και επιλέγει υποψήφιες απαντήσεις μέσω στρατηγικής top-K. Από αυτό το στάδιο προκύπτει η κρίσιμη απόφαση του τι θα ρωτηθεί όχι πώς θα διατυπωθεί η ερώτηση.

Το **Στρώμα Παραγωγής** μετατρέπει τις επιλεγμένες απαντήσεις σε διατυπωμένες ερωτήσεις μέσω του μοντέλου T5, και συγκεντρώνει εναλλακτικές λανθασμένες επιλογές

(distractors) από τέσσερις συμπληρωματικές πηγές. Στο τέλος αυτού του στρώματος, οι ερωτήσεις είναι δομικά πλήρεις αλλά συχνά γλωσσικά ή σημασιολογικά ατελείς.

Το **Στρώμα Βελτίωσης** και Εξόδου εφαρμόζει ποιοτικό έλεγχο μέσω LLM (Groq API), μεταφράζει τις ερωτήσεις πίσω στα ελληνικά αν χρειάζεται, και παράγει τα τελικά αρχεία H5P για εισαγωγή σε πλατφόρμες μάθησης όπως το Moodle.

3.1.3 Σειριακή Ροή Εκτέλεσης

Πέρα από τη γενική εικόνα των τεσσάρων στρωμάτων, ο αγωγός αναλύεται σε δεκαεννέα διακριτά στάδια εκτέλεσης, που εκτελούνται με συγκεκριμένη χρονική σειρά για κάθε κείμενο εισόδου. Ο πίνακας που ακολουθεί λειτουργεί ως οδηγός ανάγνωσης και ανάλυσης των σταδίων στις επόμενες ενότητες.

a/a	Στάδιο	Module	Κύριες Συναρτήσεις	Περιγραφή λειτουργίας
1	Εκκίνηση	app.py main.py	• main() — CLI argparse • launch_app() — Gradio UI → καλούν run_pipeline()	Σημεία εισόδου του συστήματος. Παρέχουν δύο διεπαφές (Web UI, γραμμή εντολών) που καλούν την ίδια run_pipeline().
2	Ενορχήστρωση	qg/pipeline.py	• run_pipeline(text, max_questions, use_api, use_translation, ...)	Συντονίζει όλη τη ροή από είσοδο έως έξοδο, καλώντας τα modules στη σωστή σειρά.
3	Φόρτωση κειμένου	qg/file_io.py	• load_input_file() • read_txt_file() • read_pdf_file() • _clean_page_text() • _detect_repeated_lines()	Διαβάζει αρχεία .txt και .pdf, καθαρίζοντας επαναλαμβανόμενες κεφαλίδες/υποσέλιδα.
4	Ανίχνευση γλώσσας	qg/lang_detect.py	• detect_language(text, prob_threshold=0.7, min_chars=20)	Επιστρέφει «el» ή «en» χρησιμοποιώντας langdetect με ντετερμινιστικό seed.
5	Συγκρότηση Anchor Dictionary	build_anchor_dict +build_context_anchor_dict	• build_anchor_dict() • build_context_anchor_dict()	Χτίσιμο λεξικού αγκύρωσης από το πρωτότυπο ελληνικό κείμενο (μόνο για ελληνική είσοδο), ώστε οι

α/α	Στάδιο	Module	Κύριες Συναρτήσεις	Περιγραφή λειτουργίας
				πρωτότυποι όροι να επιστραφούν αυτούσιοι στο τέλος.
6	Μετάφραση εισόδου (el→en) μόνο για ελληνική είσοδο	qg/translation.py	• translate_to_english() • _get_el_en() • split_chunks(text, max_chars=450) • _clean()	Ενεργοποιείται μόνο όταν η γλώσσα εισόδου είναι ελληνική.
7	Διαχωρισμός προτάσεων	qg/nlp_pipelines.py	• split_text_to_sentences(text, lang) • get_stanza_pipeline(lang)	Χρησιμοποιεί τον Stanza tokenizer για τεμαχισμό των προτάσεων του κειμένου.
8	Γλωσσολογική ανάλυση	qg/nlp_pipelines.py	• analyze_sentence(sentence, lang)	Για κάθε έγκυρη πρόταση εκτελεί POS tagging, lemmatization και NER μέσω Stanza.
9	Επιλογή υποψήφιων απαντήσεων	qg/question_logic.py	• select_answer_tokens(tokens_info, k=2) • _is_valid_answer_token() • _group_ner_spans()	Υβριδική πολιτική 1-ανά-πρόταση + fallback: 1ο πέρασμα κρατά την top-1 απάντηση ανά πρόταση· 2ο πέρασμα προσθέτει 2η απάντηση μόνο αν δεν έχει γεμίσει το buffer (MAX_QUESTIONS×3). Εσωτερικά η select_answer_tokens() ταξινομεί υποψήφιες με 5 επίπεδα προτεραιότητας (PERSON>LOC>ORG>DATE>...).
10	Σύνθεση fill-in items	qg/question_logic.py	• build_fill_in_items(sentences_info, max_items)	Για κάθε επιλεγμένη απάντηση αντικαθιστά τη λέξη με placeholder [[answer]] μέσα στην πρόταση.
11	Sliding window για Single Choice	qg/question_logic.py	• prepare_sc_inputs(sentences_info)	Για κάθε υποψηφιότητα πολλαπλής επιλογής χτίζει συμφραστικό παράθυρο (προηγούμενη + κεντρική + επόμενη πρόταση).
12	Παραγωγή ερώτησης (T5)	qg/nlp_pipelines.py	• generate_questions_batch(pairs)	Μαρκάρει τη σωστή απάντηση με <hl>...<hl>, προθέτει «generate

α/α	Στάδιο	Module	Κύριες Συναρτήσεις	Περιγραφή λειτουργίας
				question:» και τροφοδοτεί όλες τις υποψηφιότητες σε μία batched κλήση του valhalla/t5-base-qg-hl.
13	Παραγωγή Distractors	qg/distractor_sources.py	• get_wordnet_distractors() — co-hyponyms • get_wikipedia_distractors() — auto_suggest=False	Συμπληρωματικές πηγές ανά τύπο: (α) ίδια πρόταση, (β) WordNet co-hyponyms για κοινά ουσιαστικά, (γ) Wikipedia MONO για LOCATION/ORG (OXI PERSON — επιστρέφει άσχετα πρόσωπα), (δ) cross-sentence pool για ημερομηνίες/ποσότητες, για PERSON και ως καθολικό fallback.
14	Σύνθεση Single Choice item	qg/question_logic.py	• finalize_sc_item(prepared, raw_question) • postprocess_question()	Καθαρίζει T5 artifacts, κανονικοποιεί κενά και στίξη, εξασφαλίζει ερωτηματικό· δομεί τα 4 options (1 σωστή + 3 distractors) και διατηρεί distractor_pool ως εφεδρεία.
15	LLM post-processing (EN)	qg/api_postprocess.py	• postprocess_batch_sc(items, use_api, key) • postprocess_batch_fill_in(items, use_api, key) • _has_content_overlap()	Στέλνει τα items σε batched κλήσεις στο Groq (Llama-3.1) για: (1) coherence check, (2) vagueness check, (3) template/circular detection, (4) διόρθωση γραμματικής και αντικατάσταση κακών distractors.
16	Μετάφραση εξόδου (en→el) μόνο για ελληνική είσοδο	qg/translation.py	• translate_questions_to_greek(items) • apply_anchor_dict() • _fix_contextual_mistranslation()	Μεταφράζει ερωτήσεις, σωστές απαντήσεις, distractors και ολόκληρο το distractor_pool στα ελληνικά· εφαρμόζει το anchor_dict για επαναφορά πρωτότυπων όρων.
17	LLM post-processing (EL) μόνο για ελληνική είσοδο	qg/api_postprocess.py	• postprocess_batch_greek_sc(items) • postprocess_batch_greek_fill_in(items) • _has_content_overlap()	Δεύτερος γύρος Groq, εξειδικευμένος για διόρθωση μη φυσικών διατυπώσεων και word-

a/a	Στάδιο	Module	Κύριες Συναρτήσεις	Περιγραφή λειτουργίας
				sense σφαλμάτων που προέκυψαν από τη μηχανική μετάφραση.
18	Επικύρωση και κανονικοποίηση	qg/h5p_export.py	• <code>_fix_brackets(text)</code> • <code>answer_already_seen()</code> • <code>is_circular_question()</code>	Πριν τη σειριοποίηση εφαρμόζονται τελικοί κανονιστικοί έλεγχοι (διόρθωση brackets, stemmed dedup, circular filter).
19	Εξαγωγή H5P + JSON	qg/h5p_export.py	• <code>export_single_choice_h5p(items, path)</code> • <code>export_fill_in_h5p(items, path)</code> • <code>make_single_choice_content_json()</code> • <code>make_fill_in_content_json()</code> • <code>save_json()</code>	Δημιουργεί τα τελικά .h5p αρχεία ως ZIP με συγκεκριμένη δομή. Παράλληλη παραγωγή ευανάγνωστων JSON αρχείων για επιθεώρηση και αξιολόγηση.
20	Αξιολόγηση ROUGE	evaluation/metrics.py evaluation.py	• <code>compute_rouge_scores(pred, gold)</code> • <code>compute_bleu_score()</code> • <code>_tokenize()</code> • <code>_lcs_length()</code>	Υπολογίζει ROUGE-1, ROUGE-2, ROUGE-L με best-match στρατηγική: κάθε παραγόμενη ερώτηση συγκρίνεται με ΚΑΘΕ gold ερώτηση και κρατείται το καλύτερο score.

Πίνακας 1 Σειριακή Ροή Εκτέλεσης

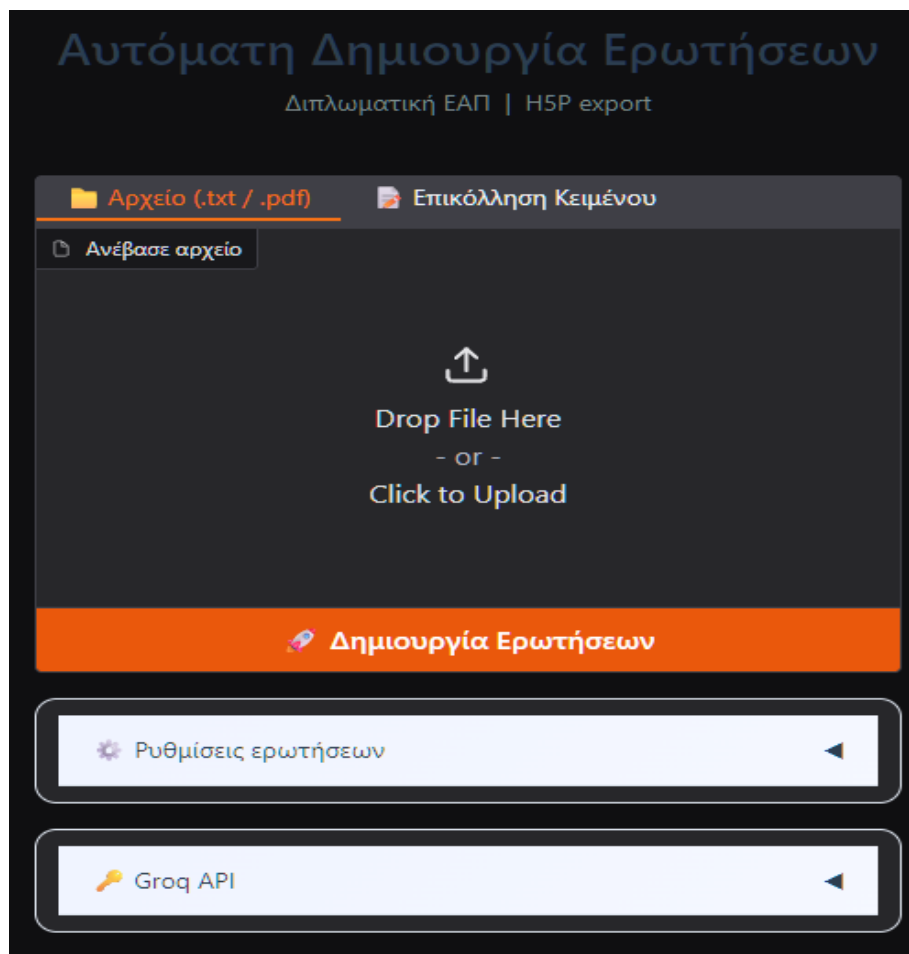
Για αγγλική είσοδο εκτελούνται **δεκαεφτά στάδια** (παραλείπονται τα 6, 16, 17). Για ελληνική είσοδο εκτελούνται και τα **είκοσι**. Το στάδιο 20 (αξιολόγηση ROUGE) δεν είναι μέρος της κανονικής ροής παραγωγής, παρά εργαλείο για την εξαγωγή μετρήσεων για τον developer.

3.2 Είσοδος και Προετοιμασία

3.2.1 Σημεία Εισόδου

Όπως προαναφέρθηκε το σύστημα προσφέρει δύο σημεία εισόδου για τον χρήστη, τη διαδικτυακή διεπαφή (app.py) και τη διεπαφή γραμμής εντολών (main.py). Παρά τις φαινομενικές τους διαφορές, και τα δύο έχουν ταυτόσημη λειτουργικότητα ως προς την παραγωγή ερωτήσεων.

Το app.py υλοποιείται με τη βιβλιοθήκη Gradio που επιτρέπει γρήγορη ανάπτυξη διεπαφών χωρίς ξεχωριστό HTML/CSS/JavaScript. Παρέχει tabs εισόδου (κείμενο ή αρχείο), κουμπιά download για τα παραγόμενα H5P αρχεία, και toggle για ενεργοποίηση/απενεργοποίηση του Groq post-processing. Ενώ το main.py όταν τρέχει δέχεται διαδρομή αρχείου ως όρισμα μέσω argparse, μαζί με προαιρετικά flags όπως --max_questions και --run_name για παραμετροποίηση.



Εικόνα 2 Η Gradio διεπαφή σε λειτουργία

```
(.venv) C:\Users\...desktop\aqg_project v8>python main.py "data\input\txt\olympics.txt"
[INFO] Φόρτωση αρχείου: data\input\txt\olympics.txt
[INFO] Ανιχνεύθηκε γλώσσα: en
2026-06-21 18:23:00 WARNING: Language en package default expects mwt, which has been added
2026-06-21 18:23:00 INFO: Loading these models for language: en (English):
```

Εικόνα 3 Το Command line σε λειτουργία

Και τα δύο σημεία εισόδου λειτουργούν ως λεπτά περιβλήματα (thin wrappers) γύρω από την κεντρική συνάρτηση ενορχήστρωσης. Δεν περιέχουν αλγοριθμική λογική παραγωγής καθώς η μοναδική τους δουλειά είναι: (α) να διαβάσουν την είσοδο του χρήστη, (β) να την περάσουν στη `run_pipeline()`, (γ) να παρουσιάσουν την έξοδο.

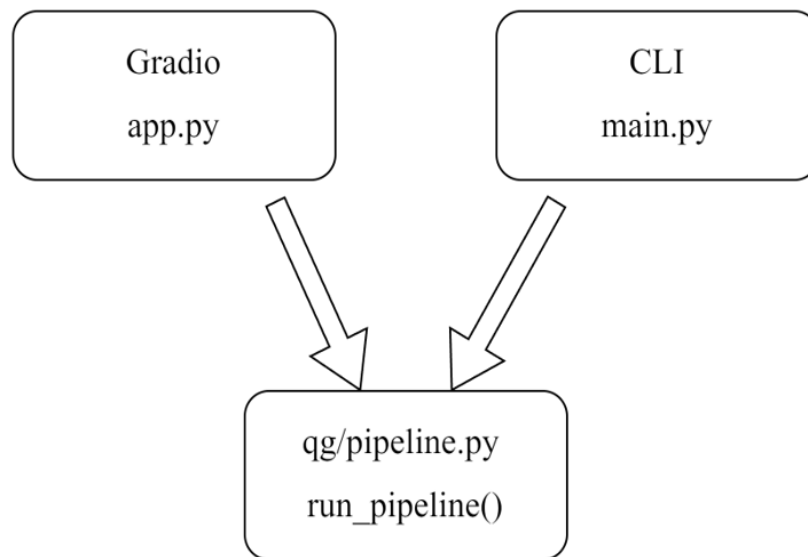
3.2.2 Ενορχήστρωση μέσω «run_pipeline»

Από τα παραπάνω modules, το πιο κεντρικό είναι το `qg/pipeline.py` που λειτουργεί ως ορχηστρωτής. Δεν περιέχει το ίδιο αλγοριθμική λογική, αλλά καλεί τα υπόλοιπα modules με τη σωστή σειρά και περνά δεδομένα μεταξύ τους. Όλες οι παράμετροι περνούν ως arguments. Αυτό επιτρέπει σε διαφορετικούς clients (Web UI, CLI) να καλούν την ίδια συνάρτηση με διαφορετικές ρυθμίσεις χωρίς να αλλάζει ο κεντρικός κώδικας.

```
def run_pipeline(  
    text: str,  
    max_questions: Optional[int] = None,  
    max_sentences: Optional[int] = None,  
    use_api: Optional[bool] = None,  
    groq_api_key: Optional[str] = None,  
    groq_model: Optional[str] = None,  
    use_translation: Optional[bool] = None,  
    verbose: bool = False,  
    log_fn: Optional[Callable[[str], None]] = None,  
) -> Tuple[str, int, List[Dict], List[Dict], Dict[str, float]]:  
  
    _cap = max_questions if max_questions else _CFG_MAX_QUESTIONS  
    _max_sent = max_sentences if max_sentences else _CFG_MAX_SENTENCES  
    _use_api = _CFG_USE_API if use_api is None else use_api  
    _groq_key = _CFG_GROQ_API_KEY if groq_api_key is None else groq_api_key  
    _groq_model = _CFG_GROQ_MODEL if groq_model is None else groq_model  
    _use_translation = _CFG_USE_TRANSLATION if use_translation is None else use_translation
```

Εικόνα 4 Κεντρική συνάρτησης ενορχήστρωσης

Η σχεδιαστική αυτή απόφαση εξυπηρετεί συνέπεια αποτελεσμάτων όπου το ίδιο κείμενο εισόδου παράγει ακριβώς τις ίδιες ερωτήσεις ανεξάρτητα από το αν εκτελέστηκε μέσω Web UI (Gradio) ή CLI και συντηρησιμότητα μιας και η βελτίωση σε ένα σημείο εμφανίζεται σε όλους τους clients.



Διάγραμμα 2 Δύο σημεία εισόδου καλούν την ίδια συνάρτηση ενορχήστρωσης

Οι διαφορετικοί clients, η διαδικτυακή διεπαφή (app.py) και η διεπαφή γραμμής εντολών (main.py) καλούν αυτή τη συνάρτηση χωρίς καμία αλγοριθμική λογική να επαναλαμβάνεται σε αυτά τα entry points.

```
def run_pipeline(text: str, max_questions: int = None):  
    from qg import config as _cfg  
    from qg.pipeline import run_pipeline as _core_run_pipeline  
    return _core_run_pipeline(  
        text=text,  
        max_questions=max_questions,  
        use_api=_cfg.USE_API,  
        groq_api_key=_cfg.GROQ_API_KEY,  
        groq_model=_cfg.GROQ_MODEL,  
        use_translation=_cfg.USE_TRANSLATION,  
        verbose=False,  
    )
```

Εικόνα 5 Entry points της app.py

```
args = parser.parse_args()

if args.max_questions is not None and args.max_questions > 0:
    print(f"[INFO] MAX_QUESTIONS override από CLI: {args.max_questions}")

print(f"[INFO] Φόρτωση αρχείου: {args.input_path}")
text = load_input_file(args.input_path)

input_lang, n_sentences, fill_in_items, single_choice_items, _timings = run_pipeline(
    text=text,
    max_questions=args.max_questions,
    verbose=True,
)
```

Εικόνα 6 Entry points της main.py

3.2.3 Φόρτωση και καθαρισμός κειμένου

Το πρώτο πραγματικό στάδιο είναι η ανάγνωση του αρχείου εισόδου. Το module `qg/file_io.py` παρέχει ενιαία είσοδο μέσω της `load_input_file()` που εξετάζει την κατάληξη του αρχείου και δρομολογεί στον κατάλληλο reader. Για αρχεία `.txt`, η φόρτωση είναι απλή ανάγνωση με UTF-8 encoding. Η δεύτερη επιλογή που αφορά την εισαγωγή αρχείων `.pdf` ενέχει κάποια στάδια επεξεργασίας.

```
def read_txt_file(path: Path) -> str:
    with path.open("r", encoding="utf-8") as f:
        text = f.read()
    return text
```

Εικόνα 7 Φόρτωση `.txt` αρχείου

Η εξαγωγή κειμένου από PDF γίνεται μέσω της βιβλιοθήκης `pdfplumber` επειδή χειρίζεται καλύτερα πολύπλοκες διατάξεις σελίδας με στήλες, πίνακες και headers. Ωστόσο, η ωμή έξοδος της `pdfplumber` περιέχει αρκετό θόρυβο όπως αριθμούς σελίδων, headers/footers, μεταδεδομένα που επηρεάζει την ποιότητα της επόμενης ανάλυσης. Οπότε ο καθαρισμός γίνεται σε δύο επίπεδα. Η συνάρτηση `_clean_page_text()` εφαρμόζεται per page και αφαιρεί με regular expressions: μεμονωμένους αριθμούς σελίδας («12», «- 5 -»), φράσεις τύπου «Page 3 of 20», γραμμές με URLs, και citations με «Πηγή:». Η συνάρτηση `_detect_repeated_lines()` εφαρμόζεται cross-page και εντοπίζει γραμμές που εμφανίζονται σε τουλάχιστον τρεις σελίδες — αυτές είναι πιθανότατα headers ή footers του εγγράφου και αφαιρούνται.

```
def _clean_page_text(text: str) -> str:
    if re.match(r"(πηγή|source)\s*:", stripped, re.IGNORECASE):
        continue

    if i in (0, len(lines) - 1) and len(stripped.split()) < 4:
        continue

    cleaned.append(line)

    return "\n".join(cleaned)

def _detect_repeated_lines(pages_text: list[str], min_pages: int = 3) -> set[str]:
    from collections import Counter
    line_counts: Counter = Counter()

    for page_text in pages_text:
        unique_lines = {ln.strip() for ln in page_text.splitlines() if ln.strip()}
        for ln in unique_lines:
            line_counts[ln] += 1

    return {ln for ln, count in line_counts.items() if count >= min_pages}
```

Εικόνα 8 Καθαρισμός σελίδων πριν την ανάγνωση αρχείου

Για όλα τα PDF επιβάλλεται **όριο 15 σελίδων** ανά αρχείο. Η επιλογή είναι εμπειρική καθώς αντιστοιχούν σε περίπου 50-80 προτάσεις, αρκετές για ένα αυτοτελές quiz σε εκπαιδευτική ενότητα. Πιο εκτεταμένα κείμενα θα χρειάζονταν χρόνο επεξεργασίας που δεν δικαιολογείται από το παιδαγωγικό όφελος.

3.2.4 Ανίχνευση Γλώσσας

Μετά τη φόρτωση, το κείμενο περνά από τη `detect_language()` του `qg/lang_detect.py`. Σκοπός είναι να αποφασιστεί αν το pipeline θα τρέξει στο αγγλικό μονοπάτι (απευθείας επεξεργασία) ή στο ελληνικό μονοπάτι (με ενδιάμεση μετάφραση).

Η ανίχνευση χρησιμοποιεί τη βιβλιοθήκη `langdetect`, εκπαιδευμένο σε n-grams από τη Wikipedia. Επιλέχθηκε επειδή είναι ελαφριά, ντετερμινιστική μέσω `fixed seed`, και έχει υψηλή ακρίβεια σε εκπαιδευτικά κείμενα >100 χαρακτήρων.

```
import re
from langdetect import detect_langs, DetectorFactory

from .config import SUPPORTED_LANGS

def detect_language(
    text: str,
    prob_threshold: float = 0.7,
    min_chars: int = 20,
) -> str:

    DetectorFactory.seed = 0

    if not text or not isinstance(text, str):
        return "en"

    cleaned = re.sub(
        r"^[^A-Za-z0-9-ωεξκ-η-ιέ-κθ-9\.,\;\:\!\?\\-\\s]",
        "",
        text,
    )
    cleaned = re.sub(r"\\s+", " ", cleaned).strip()

    if len(re.sub(r"\\s+", "", cleaned)) < min_chars:
        return "en"

    try:
        langs = detect_langs(cleaned)
        if not langs:
            return "en"

        top = langs[0]
        if top.prob >= prob_threshold and top.lang in SUPPORTED_LANGS:
            return top.lang

        for cand in langs:
            if cand.lang in SUPPORTED_LANGS and cand.prob >= prob_threshold:
                return cand.lang
    except Exception:
        return "en"

    return "en"
```

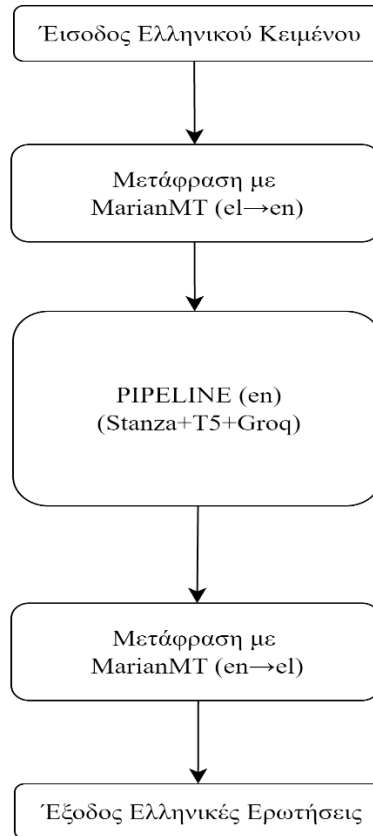
Εικόνα 9 qg/lang_detect.py

Ένα σημείο που αξίζει προσοχής είναι ο ντετερμινισμός(ίδια είσοδος → ίδια έξοδος). Το default API της langdetect επιστρέφει διαφορετική γλώσσα σε διαφορετικές εκτελέσεις για το ίδιο κείμενο, ειδικά σε σύντομα ή πολυγλωσσικά. Η ρύθμιση `DetectorFactory.seed = 0` εξαλείφει αυτή τη μη-ντετερμινιστικότητα. Τέλος η στρατηγική fallback είναι επιστροφή "en", δηλαδή αν η ανίχνευση αποτύχει, το αγγλικό μονοπάτι είναι το «βασικό» που λειτουργεί χωρίς μετάφραση. Σε χειρότερη περίπτωση, ένα ελληνικό κείμενο που λανθασμένα αναγνωρίστηκε ως αγγλικό θα παράξει χαμηλής ποιότητας ή κενά αποτελέσματα, αλλά δεν θα προκαλέσει crash.

3.2.5 Μετάφραση Εισόδου (el→en)

Αν η ανίχνευση εντοπίσει ελληνικά, ενεργοποιείται το translation layer. Το `qg/translation.py` υλοποιεί την προσέγγιση «sandwich» όπου το ελληνικό κείμενο

μεταφράζεται στα αγγλικά πριν την επεξεργασία, τρέχει όλο το pipeline σε αγγλικά, και στο τέλος οι ερωτήσεις μεταφράζονται πίσω στα ελληνικά.



Διάγραμμα 3 Προσέγγιση «sandwich»

Κάθε μεταφραστικό μοντέλο απαιτεί περίπου 500 MB μνήμης. Αν φορτώνονταν στην έναρξη του προγράμματος ανεξάρτητα από αν θα χρησιμοποιηθούν, ο χρόνος εκκίνησης θα ήταν αδικαιολόγητα μεγάλος για αγγλικά κείμενα που δεν χρειάζονται μετάφραση. Η λύση είναι **lazy loading** μέσω του Python decorator `@lru_cache(maxsize=1)` όπου το μοντέλο φορτώνεται μόνο την πρώτη φορά που χρειάζεται και διατηρείται στη μνήμη για όλη τη συνεδρία.

```
from __future__ import annotations
from functools import lru_cache
import re
from .config import DEVICE, USE_API, GROQ_API_KEY, GROQ_MODEL

_EL_EN_MODEL = "Helsinki-NLP/opus-mt-tc-big-el-en"
_EN_EL_MODEL = "Helsinki-NLP/opus-mt-en-el"
_CHUNK_SIZE = 450

@lru_cache(maxsize=1)
def _get_el_en():
    from transformers import MarianMTModel, MarianTokenizer
    tok = MarianTokenizer.from_pretrained(_EL_EN_MODEL)
    model = MarianMTModel.from_pretrained(_EL_EN_MODEL).to(DEVICE)
    return tok, model

@lru_cache(maxsize=1)
def _get_en_el():
    from transformers import MarianMTModel, MarianTokenizer
    tok = MarianTokenizer.from_pretrained(_EN_EL_MODEL)
    model = MarianMTModel.from_pretrained(_EN_EL_MODEL).to(DEVICE)
    return tok, model
```

Εικόνα 10 Lazy loading MarianMT

Τέλος, το MarianMT παράγει γνωστά artifacts που πρέπει να αφαιρεθούν. Το πιο χαρακτηριστικό εμφανίζεται κολλημένο σε κύρια ονόματα τόπων: π.χ. το «Κωνσταντίνος» επιστρέφεται ως «ΚωνσταντίνοςCity name (optional, probably does not need a translation)». Η συνάρτηση `_clean()` αφαιρεί αυτές τις περιπτώσεις με regex.

```
def _clean(text: str) -> str:
    text = re.sub(r'City name \s(optional,?\s*probably does not need a translation)', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text
```

Εικόνα 11 Καθαρισμός με regex

3.2.6 Anchor Dictionary

Η προσέγγιση «sandwich» (μετάφραση el→en, επεξεργασία, μετάφραση en→el) εισάγει μια αδυναμία που αφορά στην διπλή διέλευση ενός όρου από τη μηχανική μετάφραση με μεγάλη πιθανότητα αλλοίωσης. Λόγω ύπαρξης όρων με ταυτόσημες σημασίες ή απώλειας συμφραζομένων, ο όρος που επιστρέφει στα ελληνικά δεν ταυτίζεται πάντα με τον πρωτότυπο. Για την αντιμετώπιση του φαινομένου, χρησιμοποιείται το `qg/anchor_dict.py`, έναν μηχανισμό που λειτουργεί σε δύο φάσεις. Το βασικό πλεονέκτημά του είναι ότι, σε αντίθεση με τον έλεγχο μέσω LLM, δεν εξαρτάται από εξωτερικό API και είναι 100%

αξιόπιστος για όρους που υπήρχαν στο αρχικό κείμενο. Η λειτουργία και χρήση του αναλύεται σε 2 φάσεις:

Φάση 1: Χτίσιμο (build). Πριν ξεκινήσει η μετάφραση εισόδου, η συνάρτηση `build_anchor_dict()` σαρώνει το ελληνικό κείμενο και εξάγει τους όρους που δεν επιτρέπεται η αλλαγή τους, όπως κύρια ονόματα, κεφαλαιοποιημένες φράσεις και ουσιαστικά περιεχομένου. Για κάθε όρο χτίζεται μια αντιστοίχιση που συνδέει την αναμενόμενη αλλοιωμένη του μορφή με τον πρωτότυπο. Έτσι, σαν παράδειγμα ακόμη κι αν αργότερα η μετάφραση παράγει «κεφάλαιο», το σύστημα γνωρίζει ότι ο σωστός όρος ήταν «πρωτεύουσα».

```
def build_anchor_dict(greek_text: str, max_anchors: int = 200) -> Dict[str, str]:
    if not greek_text or not isinstance(greek_text, str):
        return {}

    anchors = _extract_greek_anchors(greek_text)
    if not anchors:
        return {}

    if len(anchors) > max_anchors:
        anchors = anchors[:max_anchors]

    try:
        from .translation import translate_to_english, translate_to_greek
    except Exception:
        return {}

    anchor_dict: Dict[str, str] = {}

    original_greek_lowercase = {a.lower() for a in anchors}

    for greek_term in anchors:
        try:
            english = translate_to_english(greek_term).strip()
        except Exception:
            continue
        if not english:
            continue
```

Εικόνα 12 Χτίσιμο Anchor Dict

Φάση 2: Εφαρμογή (apply). Στο τέλος της ροής, κατά τη μετάφραση εξόδου en→el, η συνάρτηση `apply_anchor_dict()` σαρώνει το μεταφρασμένο ελληνικό κείμενο και επαναφέρει τους πρωτότυπους όρους όπου εντοπίζει τις αλλοιωμένες μορφές. Η αντικατάσταση γίνεται με Unicode-safe word boundaries, ώστε να μη «σπάει» μέσα σε

ελληνικές λέξεις— και με προτεραιότητα στις μεγαλύτερες φράσεις, ώστε π.χ. το «Κωνσταντίνος Α'» να διορθώνεται πριν το σκέτο «Κωνσταντίνος».

```
def apply_anchor_dict(text: str, anchor_dict: Optional[Dict[str, str]]) -> str:
    if not text or not anchor_dict:
        return text

    result = text

    sorted_keys = sorted(anchor_dict.keys(), key=len, reverse=True)

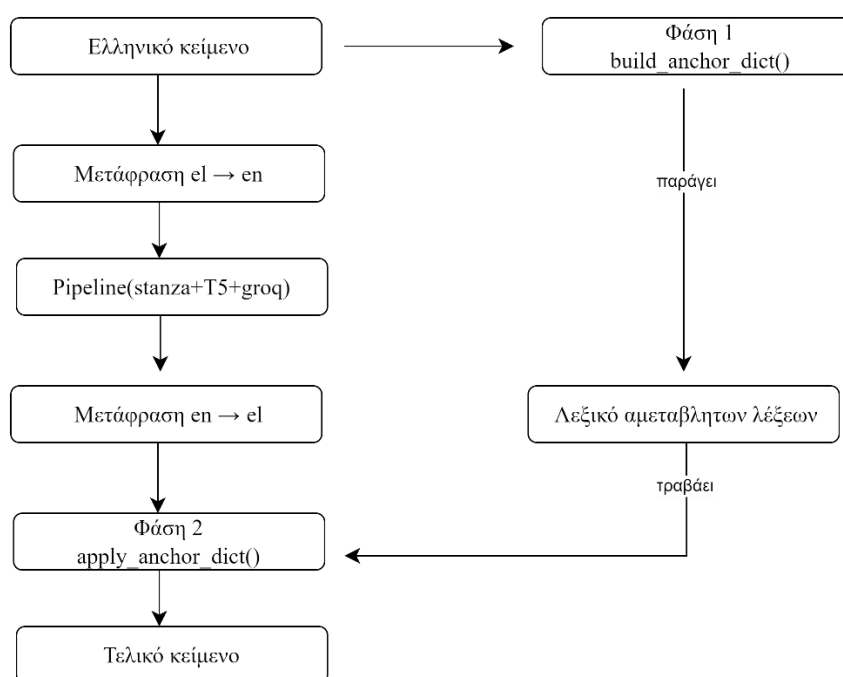
    for english_lower in sorted_keys:
        greek_original = anchor_dict[english_lower]
        if not english_lower or not greek_original:
            continue

        pattern = r"(?<!\w)" + re.escape(english_lower) + r"(?!\\w)"
        try:
            result = re.sub(pattern, greek_original, result, flags=re.IGNORECASE)
        except re.error:
            continue

    return result
```

Εικόνα 13 Εφαρμογή anchor dictionary

Ο μηχανισμός είναι fail-safe, δηλαδή αν το λεξικό προκύψει κενό ή αποτύχει το χτίσιμο, το pipeline συνεχίζει κανονικά χωρίς αυτό. Έτσι, στη χειρότερη περίπτωση η συμπεριφορά υποβαθμίζεται στην προηγούμενη έκδοση, χωρίς ποτέ να σπάει.



Διάγραμμα 4 Ροή λειτουργίας anchor dictionary

3.3 Γλωσσολογική Ανάλυση και Επιλογή

3.3.1 Stanza Pipeline και Φιλτράρισμα Προτάσεων

Με το κείμενο πλέον σε αγγλικά (είτε αρχικά είτε μετά τη μετάφραση), ξεκινά η γλωσσολογική φάση. Το `qg/nlp_pipelines.py` παρέχει wrappers γύρω από τη βιβλιοθήκη Stanza του Stanford NLP Group, που εκτελεί τα τέσσερα βασικά στάδια γλωσσολογικής ανάλυσης. Όπως και τα μεταφραστικά μοντέλα, τα Stanza pipelines φορτώνονται “lazily” και διατηρούνται σε module-level dictionary. Η κρίσιμη επιλογή `REUSE_RESOURCES` σημαίνει ότι μετά την πρώτη φόρτωση δεν χρειάζεται σύνδεση στο internet, δηλαδή τα μοντέλα παραμένουν τοπικά.

```
def get_stanza_pipeline(lang: str) -> stanza.Pipeline:
    global _STANZA_PIPELINES

    if lang not in SUPPORTED_LANGS:
        lang = "en"

    if lang not in _STANZA_PIPELINES:

        from stanza.pipeline.core import DownloadMethod

        processors = "tokenize,pos,lemma,ner"
        try:
            import json
            from pathlib import Path
            res_path = Path.home() / "stanza_resources" / "resources.json"
            if res_path.exists():
                with res_path.open("r", encoding="utf-8") as f:
                    resources = json.load(f)
                    if "ner" not in resources.get(lang, {}):
                        processors = "tokenize,pos,lemma"
        except Exception:
            pass

        _STANZA_PIPELINES[lang] = stanza.Pipeline(
            lang=lang,
            processors=processors,
            use_gpu=(DEVICE == "cuda"),
            download_method=DownloadMethod.REUSE_RESOURCES,
        )

    return _STANZA_PIPELINES[lang]
```

Εικόνα 14 Lazy load Stanza

Πριν η Stanza εφαρμοστεί ανά πρόταση, χρειάζεται διαχωρισμός σε προτάσεις. Η συνάρτηση `split_text_to_sentences()` χρησιμοποιεί τον tokenizer του Stanza, που χειρίζεται σωστά συντομογραφίες («Dr. Smith went to Washington» = μία πρόταση, όχι δύο λόγω τελείας στο Dr)

Ωστόσο κάθε πρόταση που επιστρέφει ο tokenizer δεν έχει εκπαιδευτική αξία. Πολλές είναι τίτλοι κεφαλαίων ή ονομαστικές φράσεις που δεν προσφέρονται για παραγωγή ερώτησης. Εφαρμόζονται δύο διαδοχικά φίλτρα: ελάχιστο 7 λέξεις (αποκλείει σύντομους τίτλους), και ύπαρξη ρήματος (UPOS VERB ή AUX που αποκλείει ονομαστικές κατασκευές χωρίς ρηματική ενέργεια).

```
def split_text_to_sentences(text: str, lang: str) -> List[str]:  
  
    pipeline = get_stanza_pipeline(lang)  
  
    doc = pipeline(text)  
  
    sentences: List[str] = []  
  
    for sent in doc.sentences:  
        sentence_text = " ".join(token.text for token in sent.tokens).strip()  
        if not sentence_text:  
            continue  
  
        words = sentence_text.split()  
  
        if len(words) < 7:  
            continue  
  
        has_verb = any(  
            w.upos in {"VERB", "AUX"}  
            for token in sent.tokens  
            for w in token.words  
        )  
        if not has_verb:  
            continue  
  
        sentences.append(sentence_text)  
  
    return sentences
```

Εικόνα 15 Διαχωρισμός και φιλτράρισμα προτάσεων

Για κάθε πρόταση που πέρασε τα φίλτρα, η `analyze_sentence()` εκτελεί την πλήρη ανάλυση και επιστρέφει λίστα από dicts (ένα ανά token) με το κείμενο, το lemma, το POS tag (upos

και χρος), το NER tag, και τις θέσεις χαρακτήρων (start_char, end_char). Οι τελευταίες είναι κρίσιμες για το επόμενο στάδιο, όπου επιλέγονται οι υποψήφιες απαντήσεις.

Είσοδος:		"Aristotle taught Alexander in 343 BC."
Στάδιο 1	Tokenize:	[Aristotle, taught, Alexander, in, 343, BC, .]
Στάδιο 2	POS:	PROPN VERB PROPN ADP NUM PROPN PUNCT PUNCT
Στάδιο 3	Lemma:	Aristotle teach Alexander in 343 BC .
Στάδιο 4	NER:	PERSON - PERSON -DATE - DATE

Πίνακας 2 Ανάλυση Stanza

3.3.2 Επιλογή Απαντήσεων

Έχοντας πλήρη γλωσσολογική ανάλυση κάθε πρότασης, το pipeline εισέρχεται στο πιο κρίσιμο σχεδιαστικά στάδιο της επιλογής του τι θα ρωτηθεί. Το ερώτημα δεν είναι «πώς θα διατυπώσουμε μια ερώτηση», αλλά «ποια λέξη ή φράση της πρότασης πρέπει να γίνει η σωστή απάντηση». Η επιλογή καθορίζει σε σημαντικό βαθμό την παιδαγωγική αξία πριν καν τρέξει η νευρωνική παραγωγή.

Πρόταση: "Aristotle taught Alexander in 343 BC."		
Απάντηση	Παραγόμενη Ερώτηση	Ποιότητα
"Αριστοτέλης"	"Ποιος δίδαξε τον Αλέξανδρο;"	Καλή
"Μακεδονία"	"Πού δίδαξε ο Αριστοτέλης;"	Ικανοποιητική
"δίδαξε":	"Τι έκανε ο Αριστοτέλης;"	Τετριμμένη (fallback)
"τον"	(γραμματικά κατεστραμμένη)	Αποτυχία

Πίνακας 3 Επιλογή απάντησης

Η επιλογή υλοποιείται μέσω στρατηγικής top-K — επιλογή K απαντήσεων ανά πρόταση. Με top-K, μια πλούσια πρόταση όπως αυτή του παραδείγματος μπορεί να αποδώσει 2-3 ερωτήσεις (Αριστοτέλης, Αλέξανδρος, Μακεδονία).

Η επιλογή των K υποψηφίων δεν γίνεται τυχαία ούτε με βαθμολόγηση μοντέλου. Εφαρμόζεται ντετερμινιστικό σύστημα πέντε επιπέδων προτεραιότητας (tiers):

```
def select_answer_tokens(tokens_info: List[Dict], k: int = 3) -> List[Dict]:
    named_entities = _find_named_entities(tokens_info)

    def _ner_base(t: Dict) -> str:
        n = (t.get("ner") or "").upper()
        for pfx in ("S-", "B-", "I-", "E-"):
            if n.startswith(pfx):
                return n[2:]
        return n

    _HIGH_VALUE = {
        "PERSON", "PER",
        "LOC", "GPE", "FAC", "LOCATION",
        "ORG", "ORGANIZATION",
        "EVENT",
        "NORP",
        "WORK_OF_ART",
    }

    tiers: List[List[Dict]] = [[], [], [], [], []]

    for t in named_entities:
        base = _ner_base(t)
        if not _is_valid_answer_text(t.get("text"), token_info=t):
            continue
        if base in _HIGH_VALUE:
            tiers[0].append(t)
        elif base in {"DATE", "TIME"}:
            tiers[1].append(t)
        elif base in {"QUANTITY", "PERCENT", "MONEY"}:
            tiers[2].append(t)

    for t in _find_nouns(tokens_info):
        if _is_valid_answer_text(t.get("text"), token_info=t):
            tiers[3].append(t)

    for t in tokens_info:
        upos = (t.get("upos") or "").upper()
        xpos = (t.get("xpos") or "").upper()
        if upos in {"NOUN", "PROPN"} or xpos.startswith("NN"):
            if _is_valid_answer_text(t.get("text"), token_info=t):
                tiers[4].append(t)
```

Εικόνα 16 Προτεραιότητα επιλογής υποψήφιας απάντησης

Το **Tier 1** περιλαμβάνει κατηγορίες υψηλής αξίας όπως πρόσωπα, τόποι, οργανισμοί, γεγονότα, εθνότητες, έργα τέχνης. Αυτά είναι εξ ορισμού τα πιο πλούσια ερωτηματικά στοιχεία ενός εκπαιδευτικού κειμένου. Το **Tier 2** περιλαμβάνει χρονικές αναφορές

(ημερομηνίες, εποχές). Το **Tier 3** ποσοτικές αναφορές. Το **Tier 4** κοινά ουσιαστικά. Το **Tier 5** είναι fallback με χαλαρότερα κριτήρια.

Το σύστημα εξαντλεί ένα tier πριν προχωρήσει στο επόμενο. Αν τα κύρια ονόματα μιας πρότασης δίνουν αρκετές απαντήσεις, δεν χρησιμοποιούνται καθόλου κοινά ουσιαστικά. Έτσι δίνεται προτεραιότητα στην ποιότητα. Πριν προστεθεί υποψήφιος σε tier, περνά από τη συνάρτηση `_is_valid_answer_text()` που εφαρμόζει τριπλό φίλτρο.

Οι **stopwords** αποκλείουν άρθρα, αντωνυμίες, προθέσεις. Οι **spelled-out αριθμοί** οδηγούν σε τετριμμένες ερωτήσεις τύπου «How many two...» και αποκλείονται. Τα **epithets** όπως «Great», «Elder» εμφανίζονται ως μέρος ονομάτων («Alexander the Great»), αλλά δεν στέκουν ως αυτόνομες απαντήσεις. Το υπόψη φίλτρο είναι δυναμικό και δύναται να ενισχυθεί με την παραγωγή περισσότερων ειδών κειμένου και τις αντίστοιχες ορολογίες που δεν επιθυμούνται ως επιλογή απάντησης.

Ένα τεχνικό σημείο είναι ο χειρισμός πολυλεκτικών οντοτήτων. Η Stanza επιστρέφει το NER σε επίπεδο token με BIO/BIOES tagging π.χ. το «William Shakespeare» αποτελείται από δύο tokens, ένα με B-PERSON και ένα με E-PERSON. Η συνάρτηση `_group_ner_spans()` συγχωνεύει αυτά τα διαδοχικά tokens σε ενιαία οντότητα, έτσι ώστε η επιλογή απάντησης να επιστρέφει «William Shakespeare» αντί για δύο ξεχωριστές απαντήσεις «William» και «Shakespeare».

```
def _group_ner_spans(tokens_info: List[Dict]) -> List[Dict]:
    spans: List[Dict] = []
    current_tokens: List[Dict] = []
    current_type: Optional[str] = None

    for token in tokens_info:
        ner = (token.get("ner") or "0").upper()

        if ner == "0":
            if current_tokens:
                spans.append(_make_span_dict(current_tokens, current_type))
                current_tokens = []
                current_type = None
            continue

        if len(ner) > 2 and ner[1] == "-":
            prefix, base_type = ner[0], ner[2:]
        else:
            prefix, base_type = "S", ner

        if prefix == "S":
            if current_tokens:
                spans.append(_make_span_dict(current_tokens, current_type))
                current_tokens = []
                current_type = None
            spans.append(_make_span_dict([token], base_type))

        elif prefix == "B":
            if current_tokens:
                spans.append(_make_span_dict(current_tokens, current_type))
            current_tokens = [token]
            current_type = base_type
```

Εικόνα 17 Χειρισμός πολυλεκτικών οντοτήτων

3.3.3 Σύνθεση Fill in the blanks items

Έχοντας επιλεγμένες τις υποψήφιες απαντήσεις, το pipeline σχηματίζει δύο τύπους ερωτήσεων: **fill in the blanks** και **single choice**. Παρόλο που μοιράζονται την ίδια διαδικασία επιλογής απάντησης, διαφέρουν θεμελιωδώς στο τι ζητούν από τον μαθητή και συνεπώς στους σχεδιαστικούς περιορισμούς που εφαρμόζονται στη σύνθεσή τους

Στα fill-in, η ερώτηση είναι η ίδια η πρόταση με ένα κενό στη θέση της απάντησης. Π.χ. από την πρόταση «Ο Αριστοτέλης δίδαξε τον Αλέξανδρο», με επιλεγμένη απάντηση «Αριστοτέλης», παράγεται: Ο [[Αριστοτέλης]] δίδαξε τον Αλέξανδρο.

Τα διπλά αγκύλια [[...]] σηματοδοτούν το blank. Στο τελικό H5P αυτά μετατρέπονται σε *answer* που είναι η σύνταξη της library H5P.Blanks.

```
def _make_fill_in_item(sentence: str, answer_token: Dict) -> Optional[Dict]:
    answer_text = _normalize_text(answer_token.get("text"))
    if not answer_text:
        return None

    answer_text = _clean_fill_in_answer(answer_text)
    if not answer_text or len(answer_text) < 4:
        return None

    _ner = (answer_token.get("ner") or "0").upper()
    _is_entity = _ner != "0"
    if not _is_entity and answer_text.lower() in _WEAK_FILL_IN_NOUNS:
        return None

    if len(answer_text.split()) > FILL_IN_MAX_ANSWER_WORDS:
        return None

    count = sentence.count(answer_text)
    marker = f"[[{answer_text}]]"
    text_with_blank = sentence.replace(answer_text, marker, 1)
    if count > 1:

        head, sep, tail = text_with_blank.partition(marker)
        tail = tail.replace(answer_text, "...")
        text_with_blank = head + sep + tail

    return {
        "type": "fill_in",
        "original_sentence": sentence,
        "answer": answer_text,
        "text_with_blank": text_with_blank,
    }
```

Εικόνα 18 Σύνθεση fill in the blanks

Τρεις κανόνες εφαρμόζονται στη σύνθεση κάθε fill in item που προέκυψαν κατά τον πειραματισμό σε εκπαιδευτικά κείμενα.

1. **Μασκάρισμα επαναλαμβανόμενων εμφανίσεων.** Αν η απάντηση εμφανίζεται περισσότερες από μία φορές στην ίδια πρόταση, μόνο η πρώτη εμφάνιση γίνεται blank. Οι υπόλοιπες εμφανίσεις αντικαθιστούνται με «...» ώστε να μην αποκαλύπτουν τη σωστή απάντηση στον μαθητή:

Πρόταση:	«Ο Αριστοτέλης γεννήθηκε το 384 π.Χ. και ο Αριστοτέλης πέθανε το 322 π.Χ.»
Παραγόμενο fill-in:	«Ο [[Αριστοτέλης]] γεννήθηκε το 384 π.Χ. και ο πέθανε το 322 π.Χ.»

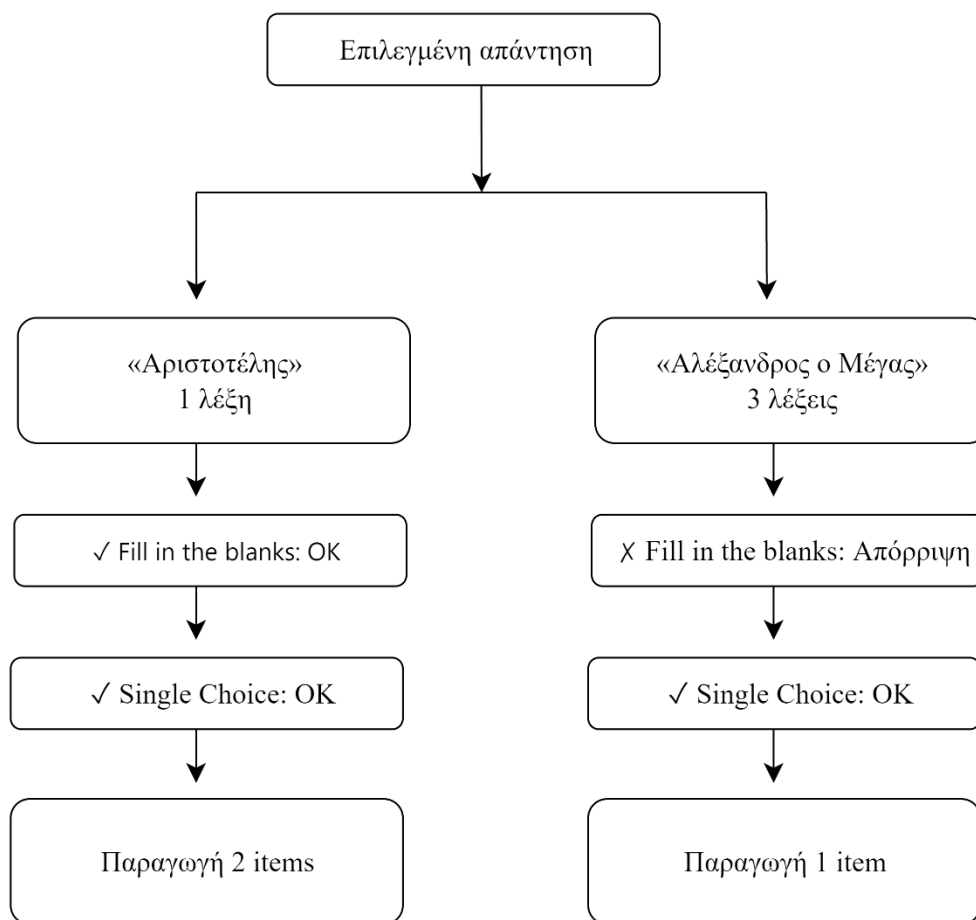
Πίνακας 4 Μασκάρισμα επαναλαμβανόμενων εμφανίσεων

Χωρίς αυτόν τον μηχανισμό, η ίδια η ερώτηση θα έδινε την απάντησή της — ο μαθητής θα διάβαζε το δεύτερο «Αριστοτέλης» και θα συμπλήρωνε το κενό χωρίς να σκεφτεί.

2. **Καθαρισμός εισαγωγικών άρθρων.** Πριν την τοποθέτηση του blank, τα άρθρα στην αρχή της απάντησης αφαιρούνται μέσω της `_clean_fill_in_answer`. Π.χ. αν η Stanza εντοπίσει το «the Roman Empire» ως ονομαστική οντότητα, η απάντηση καθαρίζεται σε «Roman Empire» πριν γίνει blank. Έτσι ο μαθητής δεν χρειάζεται να μαντέψει αν η σωστή απάντηση περιλαμβάνει ή όχι το άρθρο μια σημαντικότητα μορφής που θα οδηγούσε σε ψευδώς αρνητικές αξιολογήσεις από το H5P (το «Roman Empire» θα θεωρούνταν λάθος επειδή λείπει το «the»).

3. **Απορρίψη αδύναμων κοινών ουσιαστικών** ως κενά. Ορισμένα κοινά ουσιαστικά που δηλώνουν ρόλο ή ιδιότητα (king, queen, leader, ruler κ.ο.κ.) μαντεύονται εύκολα από το γλωσσικό περιβάλλον και δίνουν φτωχά fill in κενά. Όταν μια τέτοια λέξη δεν αποτελεί ονομαστική οντότητα (named entity), απορρίπτεται ως fill in μέσω της λίστας `_weak_fill_in_nouns`. Δεν χάνεται όμως και παραμένει διαθέσιμη ως single-choice option, όπου αποκτά εκπαιδευτική αξία σε συνδυασμό με σημασιολογικά συναφείς distractors (π.χ. king → emperor, czar, sultan).

Επιπλέον ένα σημαντικό σημείο είναι ότι μια απάντηση που απορρίπτεται για fill in the blank λόγω μήκους δεν χάνεται από το σύστημα. Παραμένει διαθέσιμη ως υποψηφιότητα για single choice, όπου ο περιορισμός μήκους δεν ισχύει καθώς εκεί ο μαθητής επιλέγει απάντηση και δεν πληκτρολογεί.



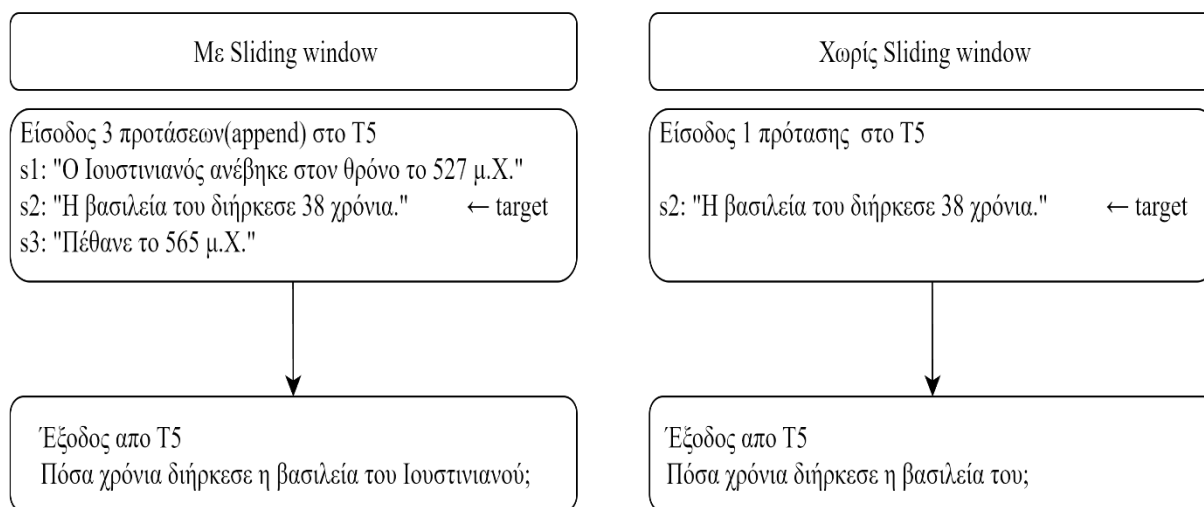
Διάγραμμα 5 Εκμετάλλευση «answer»

Έτσι το σύστημα αξιοποιεί την ίδια γλωσσολογική ανάλυση δύο φορές, με διαφορετικά φίλτρα ανά τύπο ερώτησης. Από μια πλούσια πρόταση μπορεί να προκύψουν:

- Ένα fill in item για απάντηση 1-2 λέξεων
- Ένα single choice item για την ίδια απάντηση
- Ένα επιπλέον SC item για μεγαλύτερη απάντηση που απορρίφθηκε από fill-in

3.3.4 Sliding Window – Προετοιμασία για Single Choice items

Σε αντίθεση με τα fill-in items, τα single-choice items απαιτούν νέα γλωσσική διατύπωση που παράγονται νευρωνικά. Πριν όμως κληθεί το T5, χρειάζεται κρίσιμη προεργασία, την κατασκευή κατάλληλου γλωσσικού περιβάλλοντος γύρω από την επιλεγμένη απάντηση. Αν δώσουμε στο T5 μόνο την κεντρική πρόταση, ενδέχεται να παράξει ερώτηση με κενές αναφορές όπως αντωνυμίες χωρίς αντίστοιχο πρόσωπο.



Διάγραμμα 6 Χρήση sliding window

Έτσι όταν παράγεται ερώτηση για μια συγκεκριμένη πρόταση, το T5 λαμβάνει ως είσοδο τη συνένωση τριών διαδοχικών προτάσεων: της προηγούμενης, της κεντρικής, και της επόμενης

```

sc_prepared: List[Dict] = []
_sc_buffer_cap = _cap * 3
for idx, sent in enumerate(sentences):
    if len(sc_prepared) >= _sc_buffer_cap:
        break
    parts: List[str] = []
    if idx > 0:
        parts.append(sentences[idx - 1])
    parts.append(sent)
    if idx + 1 < len(sentences):
        parts.append(sentences[idx + 1])
    context = " ".join(parts)
    for prep in prepare_sc_inputs(sent, lang=lang, global_token_pool=global_token_pool, context=context, k=2):
        sc_prepared.append(prep)
        if len(sc_prepared) >= _sc_buffer_cap:
            break
    _log(f"[INFO] SC candidates: {len(sc_prepared)} (top-K).")

```

Εικόνα 19 Σύνθεση sliding window

Η απάντηση παραμένει επιλεγμένη μόνο από την κεντρική πρόταση, όχι από τις γειτονικές. Το context χρησιμεύει αποκλειστικά ως γλωσσικό περιβάλλον και όχι ως πηγή απαντήσεων. Αν επιτρεπόταν η επιλογή απάντησης από γειτονικές προτάσεις, η ίδια απάντηση θα μπορούσε να επιλεγεί τρεις φορές καθώς διατρέχουμε το κείμενο.

3.4 Παραγωγή Ερωτήσεων

3.4.1 T5 και Highlight Format

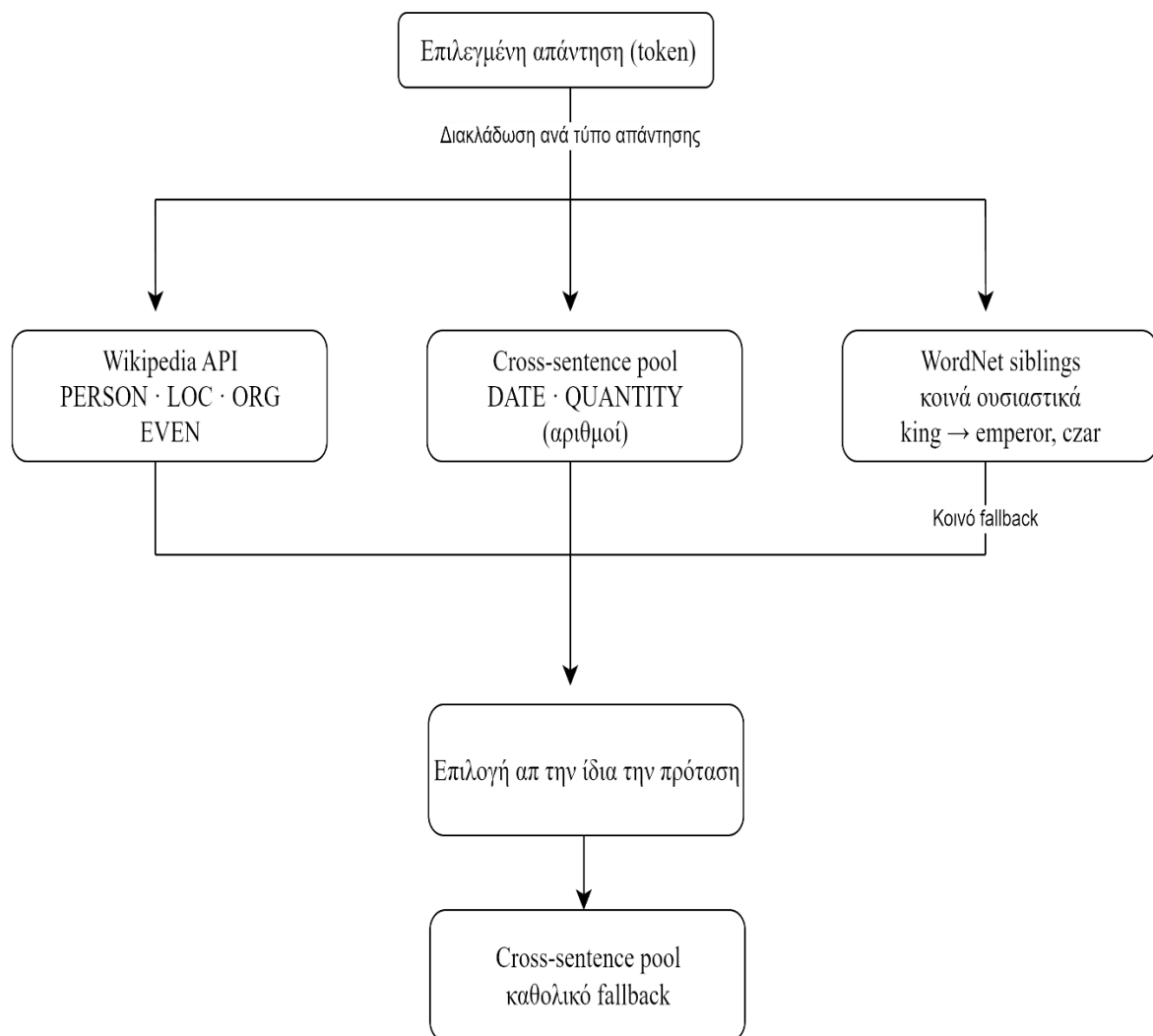
Με τις υποψήφιες απαντήσεις επιλεγμένες και το context window έτοιμο, το pipeline εισέρχεται στον νευρωνικό πυρήνα για τη γλωσσική παραγωγή της ερώτησης από το T5. Το T5 εκπαιδεύτηκε με συγκεκριμένο πρωτόκολλο εισόδου όπου η σωστή απάντηση μαρκάρεται μέσα στην πρόταση με `<hl> ... <hl>`, και το prefix `generate question` σηματοδοτεί το είδος της εργασίας. Η ενσωμάτωση των markers γίνεται μόνο μία φορά (`count=1`) ακόμη και αν η απάντηση εμφανίζεται πολλές φορές στο context — έτσι το μοντέλο δεν μπερδεύεται με πολλαπλά highlights.

Όταν για κάποιον candidate το T5 επιστρέφει κενή ερώτηση (πιθανό αν η απάντηση δεν βρέθηκε στο context λόγω μετάφρασης), εφαρμόζεται fallback batch, δηλαδή ξανά κλήση χωρίς highlights, αφήνοντας το T5 να επιλέξει μόνο του τι θα ρωτήσει. Η ερώτηση που προκύπτει δεν εγγυάται ότι έχει τη σχεδιαζόμενη απάντηση, αλλά είναι προτιμότερη από καμία· οι έλεγχοι του επόμενου σταδίου θα την απορρίψουν αν δεν ταιριάζει.

3.4.2 Παραγωγή Distractors

Με την ερώτηση παραγμένη και τη σωστή απάντηση επιλεγμένη το pipeline χρειάζεται τρεις επιπλέον λανθασμένες επιλογές (distractors) για να σχηματίσει μια πλήρη ερώτηση πολλαπλής επιλογής. Το σύστημα συνδυάζει τέσσερις συμπληρωματικές πηγές για να εξασφαλίσει επάρκεια και ποικιλομορφία:

Η ίδια πρόταση (στάδιο `_collect_from_pool inline`) είναι η πρώτη πηγή για όλους τους τύπους. Άλλα tokens της ίδιας πρότασης με συμβατό τύπο NER ή POS είναι από τη φύση τους τοπολογικά συνδεδεμένα με την απάντηση. Π.χ. σε πρόταση «Ο Αριστοτέλης δίδαξε τον Αλέξανδρο τον Μέγα στη Μακεδονία», αν η απάντηση είναι «Αριστοτέλης» (PERSON), το «Αλέξανδρος ο Μέγας» (PERSON) είναι αυτόματα κατάλληλο distractor.



Διάγραμμα 7 Επιλογή κατάλληλου distractor

Το **WordNet** είναι η κύρια πηγή για κοινά ουσιαστικά. Πρόκειται για λεξιλογική βάση που οργανώνει την αγγλική γλώσσα σε σύνολα συνωνύμων (synsets) με σχέσεις υπερωνυμίας. Για τη λέξη «king», βρίσκουμε το hypernym «male_monarch» (γενικότερη έννοια), και συλλέγουμε τα hyponyms του («αδέρφια»): emperor, czar, sultan, pharaoh.

Το **Wikipedia API** χρησιμοποιείται για κύρια ονόματα τύπου τοποθεσίας (LOC/GPE/FAC) και οργανισμού (ORG). Για ένα entity, το σύστημα ψάχνει στη Wikipedia, επιλέγει το άρθρο που ταιριάζει στο όνομα (έλεγχος συσχέτισης τίτλου), κατεβάζει το summary των πρώτων 12 προτάσεων, τρέχει NER πάνω σε αυτό και κρατά μόνο entities ίδιου σημασιολογικού τύπου. Οι υποψήφιοι δεν επιλέγονται με τη σειρά εμφάνισης αλλά κατατάσσονται κατά συνάφεια: όσοι αναφέρονται συχνότερα (και νωρίτερα) στο άρθρο προτιμώνται. Για «Greece» (LOC), τυπικά επιστρέφει «Italy», «Egypt», «Persia». Για

PERSON entities η Wikipedia δίνει ενεργά άσχετα πρόσωπα (συγγενείς, σύγχρονους πολιτικούς), γι' αυτό δεν χρησιμοποιείται και οι PERSON distractors προέρχονται από το cross-sentence pool.

Το **cross-sentence pool** χτίζεται στο qg/pipeline.py συνενώνοντας τα tokens όλων των προτάσεων σε ενιαία λίστα, όπου ανάμεσα σε κάθε πρόταση εισάγεται ένας ειδικός «barrier» token (ner="O"). Επειδή τα char offsets της Stanza επανεκκινούν από το 0 σε κάθε πρόταση, χωρίς αυτόν τον διαχωριστή ο μηχανισμός ομαδοποίησης οντοτήτων θα συγχώνευε λανθασμένα γειτονικές οντότητες διαφορετικών προτάσεων (π.χ. «Aristotle Alexander», «the Persian Empire Egypt»). Ο barrier κλείνει κάθε ανοιχτό span στο όριο της πρότασης και αποτρέπει αυτό το cross-sentence merge:

```
global_token_pool: List[Dict] = []
for s in sentences:
    global_token_pool.extend(analyze_sentence(s, lang=lang))
    global_token_pool.append(dict(_BARRIER))
timings["stanza"] = round(time.time() - _t, 1)
_t = time.time()
```

Εικόνα 20 Global token pool

Χρησιμοποιεί με δύο τρόπους. Για ημερομηνίες και ποσότητες, είναι η κύρια πηγή για άλλες ημερομηνίες/ποσότητες του ίδιου κειμένου που είναι πιο προσεγγιστικές από αυθαίρετους αριθμούς. Π.χ. για απάντηση «776 π.Χ.» σε κείμενο για αρχαία Ολυμπιακούς αγώνες, αναζητούνται άλλες χρονολογίες του κειμένου: «490 π.Χ.», «323 π.Χ.». Για άλλους τύπους, χρησιμεύει ως fallback όταν οι κύριες πηγές δεν έδωσαν αρκετούς υποψήφιους distractors.

Καθολική αρχή σε όλες τις πηγές είναι το semantic type matching όπου ένας distractor πρέπει να ανήκει στον ίδιο σημασιολογικό τύπο με τη σωστή απάντηση, δηλαδή ο PERSON συσχετίζεται μόνο με PERSON distractors, αντίστοιχα για LOC distractors και ούτω καθεξής. Δεν αναμειγνύονται τύποι, ώστε να αποφεύγονται προφανείς αποτυχίες όπως «Ποιος έγραψε την Ιλιάδα;» με μία επιλογή «Αθήνα» που εννοιολογικά δεν ταιριάζει.

3.4.3 Τελική Σύνθεση Single Choice items

Έχοντας ερώτηση από το T5 και distractors από τις τέσσερις πηγές ξεκινούν κανονιστικοί έλεγχοι ποιότητας, που βασίζονται σε regex, λεξιλόγια και γραμματικούς ελέγχους που πιάνουν τα πιο τυπικά σφάλματα του T5.

Πρώτη ενέργεια είναι η αφαίρεση artifacts που εμφανίζονται περιστασιακά στην έξοδο του T5 ως υπολείμματα όπως «answer», «context». Στη συνέχεια, προχωρά στην κανονικοποίηση κενών, διόρθωση πολλαπλών ερωτηματικών, και εξασφάλιση τελικού ερωτηματικού (?/;). Μετά τον καθαρισμό, εφαρμόζεται έλεγχος επάρκειας, αν η ερώτηση περιέχει τουλάχιστον τρεις λέξεις και τουλάχιστον μία ερωτηματική λέξη (who, what, where, when, why, how, which). Η δεύτερη απαίτηση πιάνει περιπτώσεις όπου το T5 παρήγαγε δηλωτική πρόταση με ερωτηματικό στο τέλος, π.χ. «He is a king?» που είναι γραμματικά ελλιπής ως ερώτηση.

```
def postprocess_question(  
    raw_question: str,  
    answer_text: str,  
    answer_token: Dict,  
) -> str:  
    q = _normalize_text(raw_question)  
    a = _normalize_text(answer_text)  
  
    if not q:  
        q = ""  
  
    q = q.replace("answer>", " ")  
    q = q.replace("context>", " ")  
    q = q.replace("<answer>", " ")  
    q = q.replace("<context>", " ")  
  
    q = re.sub(r"\s+", " ", q).strip()  
  
    q = re.sub(r"^\?+\s*\?+$", "?", q)  
  
    if q and not q.endswith("?"):  
        q = q.rstrip(".")  
        q = q.strip()  
        if q:  
            q = q + "?"  
  
    wh_words = {"who", "what", "where", "when", "why", "how", "which"}  
    tokens = q.split() if q else []  
    has_wh = any(t.lower() in wh_words for t in tokens)  
  
    if len(tokens) >= 3 and has_wh:  
        return q
```

Εικόνα 21 Καθαρισμός artifacts

Παράλληλα εφαρμόζονται δύο επιπλέον έλεγχοι σε επίπεδο Single Choice item. Ο circular question filter εντοπίζει ερωτήσεις όπου η ίδια η απάντηση εμφανίζεται μέσα στην ερώτηση π.χ. «Ποια πόλη ονομάστηκε Ρώμη;» με απάντηση «Ρώμη». Αυτές μαρκάρονται ως κυκλικές και τοποθετούνται σε μειονεκτική θέση κατά την τελική επιλογή. Τέλος το options deduplication ελέγχει τις τέσσερις τελικές επιλογές για διπλότυπα (case-insensitive). Αν δύο

options καταλήγουν να είναι ίδια λόγω αλλαγών στίξης ή casing, το ένα εξ' αυτών αντικαθίσταται από εφεδρικό candidate του distractor pool.

```
def is_circular_question(question: str, answer: str) -> bool:
    if not question or not answer:
        return False

    ans = re.escape(answer.strip().lower())
    return bool(
        re.search(r"(?!\\w)" + ans + r"(?!\\w)", question.lower(), re.UNICODE)
    )
```

Εικόνα 22 Έλεγχος circular question

3.5 Ποιοτικός Έλεγχος με LLM

3.5.1 Επιλογή Groq(Llama-3.1)

Κατόπιν των προαναφερθέντων ελέγχων, η ερώτηση έχει σωστή μορφή αλλά δεν είναι ακόμη βέβαιο ότι έχει νόημα καθώς τα προβλήματα του T5, περιλαμβάνουν γραμματικά λάθη, ταυτολογίες που ξεφεύγουν από τον circular filter και νοηματικές ασυνέπειες, που δεν διορθώνονται με κανονιστικούς ελέγχους.

Συναφώς, εφαρμόστηκε LLM Post-processing, δηλαδή ένα δεύτερο επίπεδο νευρωνικού ελέγχου, για αξιολόγηση και διόρθωση των παραγόμενων αποτελεσμάτων. Η LLM επεξεργασία γίνεται μέσω του Groq API που τρέχει το open-source Llama-3.1.

Η επιλογή του Groq έγινε με κριτήρια την free tier χρήση για την έκδοση του κωδικού API που θα χρησιμοποιηθεί κάθε φορά. Είναι αρκετά ταχύ καθώς πετυχαίνει >500 tokens/sec σημαντικά γρηγορότερα από συμβατικά cloud GPUs. Χρησιμοποιεί open-source μοντέλα, το Llama-3.1 στην δική μας περίπτωση μπορεί μελλοντικά να τρέξει τοπικά μέσω Ollama, αφαιρώντας την εξάρτηση από cloud.

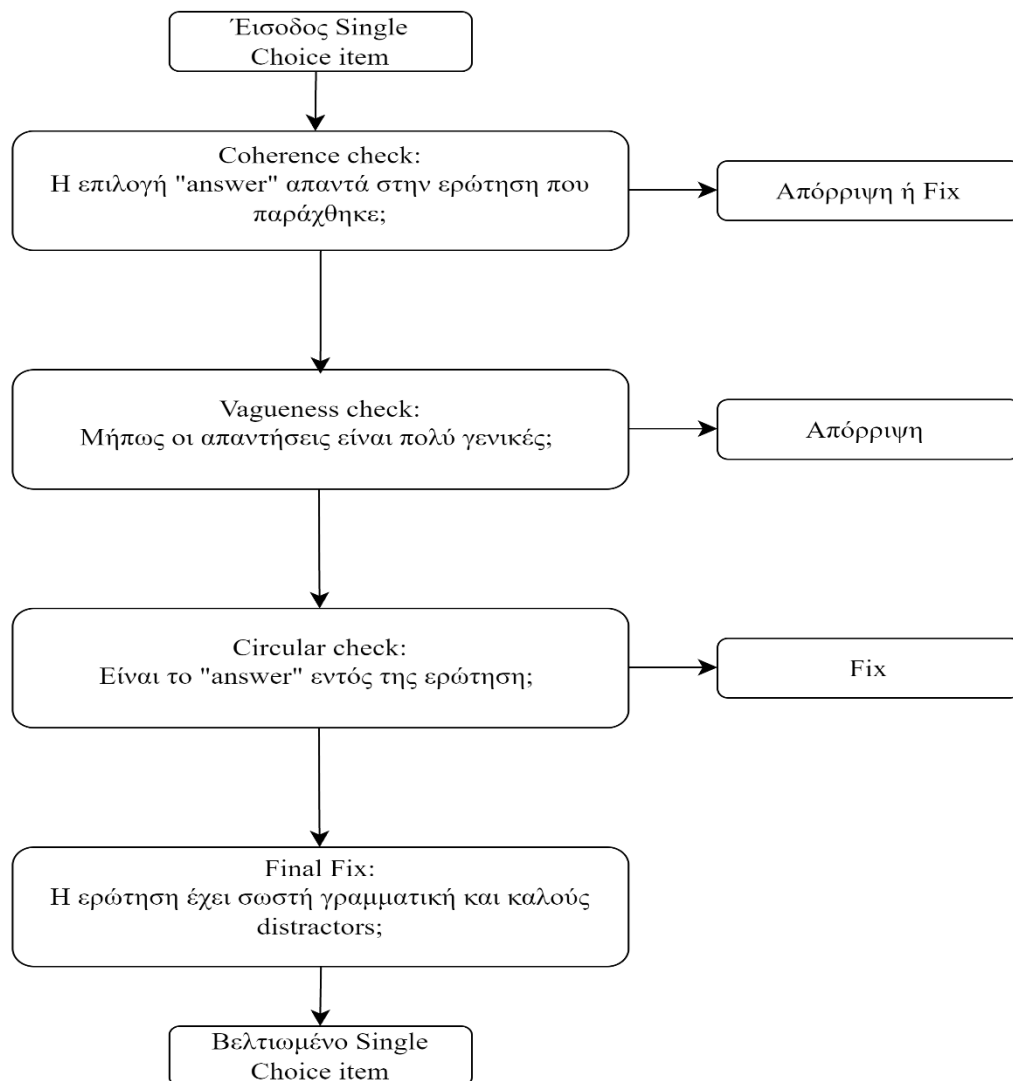
Πρέπει να τονιστεί ότι το LLM post-processing είναι προαιρετικό. Αν λείπει το API key ή η σύνδεση, το pipeline συνεχίζει χωρίς αυτό στάδιο, χρησιμοποιώντας τις ερωτήσεις όπως παρήχθησαν από το T5 και το κανονιστικό post-processing.

3.5.2 Εφαρμογή Τεσσάρων Βημάτων Ελέγχου

Επειδή το pipeline χειρίζεται τέσσερα διαφορετικά σενάρια με τις αντίστοιχες παραγωγές :

- Fill in the blanks (en)
- Single Choice items (en)
- Fill in the blanks (el)
- Single Choice items (el)

συντάχθηκαν τέσσερις διαφορετικές `batched functions` στο `qg/api_postprocess.py` που έкаστο περιλαμβάνει προσαρμοσμένο `prompt`. Εξ' αυτών το πιο πλούσιο `prompt` είναι αυτό που αφορά τον έλεγχο των Single Choice items (en) καθώς σε αυτό απαντώνται οι κύριες εννοιολογικές δυσχέρειες λόγω δομής, σε σύγκριση με τις fill in the blanks ερωτήσεις.



Διάγραμμα 8 Έλεγχος Single Choice items(en)

Coherence Check: Στο πρώτο βήμα το prompt έχει την δομή: «Read the context sentence, the question, and the answer. Does this answer correctly answer this question, based on the context? ». Το prompt περιλαμβάνει συγκεκριμένα παραδείγματα incoherent ζευγών (π.χ. context «Japan has thousands of years of history», question «What is the capital of Japan? », με απάντηση «thousands», η οποία απορρίπτεται). Αυτά τα παραδείγματα διδάσκουν στο LLM να αναγνωρίζει το pattern.

Vagueness Check: Απορρίπτει υπερβολικά γενικές απαντήσεις όπως «information», «data», «structure», «process», «system», «thing». Αυτές οδηγούν σε «invalid» μεν αλλά παιδαγωγικά τετριμμένες ερωτήσεις.

Circular Check: Εντοπίζει ερωτήσεις τύπου «What is X?» όπου X είναι η ίδια η απάντηση. Το LLM ζητείται να τις ξαναγράψει χρησιμοποιώντας διακριτικά στοιχεία από το context. Η μετατροπή «Who is Aristotle?» σε «Who was the Greek philosopher who taught Alexander the Great?» είναι παράδειγμα αυτής της λογικής.

Final Fix: Στο τελικό βήμα γίνονται διορθώσεις γραμματικής και αντικατάσταση κακών distractors με νέα ίδιας κατηγορίας.

Πέρα από τους τέσσερις ελέγχους που εκτελεί το ίδιο το LLM, η έξοδος του δεν γίνεται δεκτή άμεσα. Ειδικά για τα fill-in items, η απάντηση του Groq επικυρώνεται με ελέγχους. Το ξαναγραμμένο κείμενο πρέπει να περιέχει ακριβώς ένα κενό (διπλό «___» σπάει το H5P και απορρίπτεται), πρέπει να μοιράζεται τουλάχιστον μία ουσιαστική λέξη με την αρχική πρόταση ώστε να αποκλειστεί η περίπτωση το μοντέλο να αντέγραψε παράδειγμα ή να κατάλαβε άσχετο περιεχόμενο, και η ίδια η απάντηση δεν επιτρέπεται να εμφανίζεται ήδη μέσα στην πρόταση εκτός του κενού (διαφορετικά το κουίζ θα αποκάλυπτε τη λύση). Όποια έξοδος αποτυγχάνει σε αυτούς τους ελέγχους απορρίπτεται και διατηρείται το αρχικό item.

3.5.3 Έλεγχος Κατόπιν Μετάφρασης

Μετά τη μετάφραση των items πίσω στα ελληνικά (στάδιο 15), εκτελείται επανέλεγχος με Groq για τις ελληνικές εκδοχές. Το ελληνικό prompt εστιάζει σε γραμματική και ορθογραφία ελληνικών, σε μη φυσικές διατυπώσεις που προκύπτουν από τη μηχανική μετάφραση μέσω του MarianMT.

```
def postprocess_batch_greek_sc(
    ], ensure_ascii=False)

    user_prompt = f"""You are editing {n} Greek multiple-choice questions that were machine-translated from English. Fix every problem you find.

    Each item includes an English "context" sentence (the original source). Use it ONLY as a reference to verify facts and to find a distinguishing detail when rewriting

    Items (id, context, question, answer, distractors):
    {items_json}

    For each item, fix ALL of the following problems if present:

    LANGUAGE FIXES:
    - English words left untranslated → transliterate or translate to Greek.
      Apply to ANY Latin-script word (proper noun → transliteration; term → Greek term),
      not only the illustration. (e.g. "World War" → "Παγκόσμιος Πόλεμος")
    - Wrong grammatical gender/case agreement
    - Awkward or unnatural phrasing in Greek
    - Λάθος σημασία λέξης από μηχανική μετάφραση (WORD-SENSE MISTRANSLATION):
      X "capital" → "κεφάλαιο" (οικονομικό) ✓ "πρωτεύουσα" (πόλη)
      X "capital" → "κεφάλαια" ✓ "πρωτεύουσα"
      X "Roman" → "Ρόμαν" ✓ "Ρωμαίος" / "Ρωμαϊκός"
      X "Romans" → "Ρωμάνοι" ✓ "Ρωμαίοι"
      X "court" → "δικαστήριο" ✓ "αυλή" (βασιλική αυλή)
    → Χρησιμοποιήστε το context της ερώτησης για να αποφασίσετε ποια σημασία
    ταιριάζει, και διόρθωσε και distractors που έχουν λάθος σημασία.
```

Εικόνα 23 Groq prompt βελτίωσης Single Choice

Παράδειγμα κατά τη ροή el→en→el ο όρος «σάκχαρο» μπορεί να επιστρέψει ως «ζάχαρη» σε κείμενο βιολογίας. Στην παρούσα υλοποίηση η κύρια αντιμετώπιση γίνεται από τον μηχανισμό anchor_dict (qg/anchor_dict.py) όπου πριν τη μετάφραση εξόδου εξάγονται οι όροι του πρωτότυπου ελληνικού κειμένου και, όταν η διπλή μετάφραση τους αλλοιώσει, επαναφέρονται αυτούσιοι («ζάχαρη» → «σάκχαρο»). Ο επανέλεγχος με Groq λειτουργεί συμπληρωματικά, πιάνοντας δυναμικά περιπτώσεις που δεν υπήρχαν ως όροι στο πρωτότυπο.

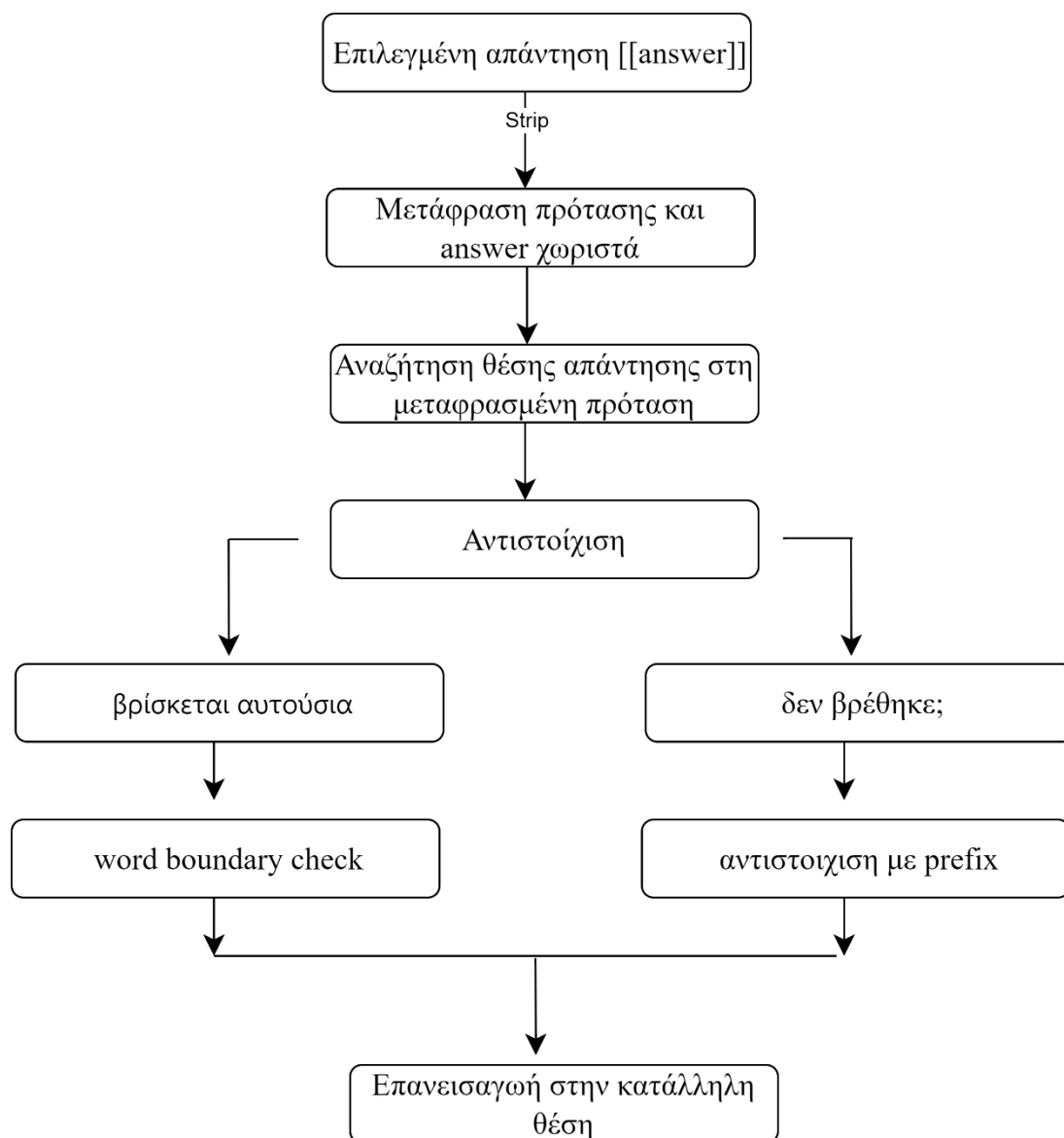
3.6 Έξοδος και παράδοση

3.6.1 Μετάφραση Εξόδου (en→el)

Όταν η αρχική γλώσσα του κειμένου ήταν ελληνική, σε αυτό το στάδιο τα items που έχουν παραχθεί είναι στην αγγλική. Πριν την εξαγωγή σε αρχεία H5P, πρέπει να μεταφραστούν πίσω στα ελληνικά. Η συνάρτηση translate_questions_to_greek του qg/translation.py υλοποιεί τη μετάφραση με διαφορετική στρατηγική για κάθε τύπο item.

Για τα Fill in the blanks items, ο πυρήνας είναι μια πρόταση με το placeholder [[answer]] στη μέση. Αν μεταφραζόταν αυτούσια, το MarianMT θα μπερδευόταν από τα διπλά αγκύλια. Η λύση είναι αποσύνθεση κατά τύπου μετάφραση και επανασύνθεση. Η αναζήτηση και αντιστοίχιση μεταφρασμένων items χρήζει προσοχής λόγω ελληνικής κλίσης. Η μεταφρασμένη απάντηση «Αλέξανδρος» μπορεί να εμφανίζεται στη

μεταφρασμένη πρόταση σε διαφορετική πτώση («Αλέξανδρο»). Αν αποτύχει το exact match, εφαρμόζεται fuzzy match με αντιστοίχιση των πρώτων 4 χαρακτήρων ως κοινό prefix.



Διάγραμμα 9 Μετάφραση Fill in the blanks (en->el)

Για τα Single Choice items, η μετάφραση γίνεται πεδίο προς πεδίο, με την μετάφραση της ερώτησης πρώτα και ύστερα της απάντησης και των distractors ξεχωριστά. Η μεταφρασμένη ερώτηση χρησιμεύει ως γλωσσικό περιβάλλον για τη μετάφραση των επιλογών («answer»/«distractor»). Συγκεκριμένα, μετά τη μετάφραση κάθε επιλογής καλείται η συνάρτηση `_fix_contextual_mistranslation()` που εφαρμόζει ορισμένες

διορθώσεις για γνωστές αποτυχίες του MarianMT σε απομονωμένα options. Π.χ. η αγγλική λέξη «capital» σε ένα option μεταφράζεται ως «κεφάλαιο» (η συχνότερη χρήση). Αν όμως η μεταφρασμένη ερώτηση περιέχει τη λέξη «πρωτεύουσα», η διορθωτική συνάρτηση εντοπίζει ότι το context αφορά πόλη και διορθώνει την επιλογή σε «πρωτεύουσα».

```
elif q_type == "single_choice":

    gr_question = translate_to_greek(_pre_anchor_en(item["question"]))

    gr_question = _maybe_anchor(gr_question)
    new_options = []
    for opt in item["options"]:
        new_opt = dict(opt)
        raw_translated = translate_to_greek(_pre_anchor_en(opt["text"]))

        raw_translated = _maybe_anchor(raw_translated)

        new_opt["text"] = _fix_contextual_mistranslation(raw_translated, gr_question)

        if not new_opt.get("correct") and _is_untranslated_latin(new_opt["text"]):
            continue
        new_options.append(new_opt)

    translated_pool: list = []
    seen_pool: set = set()
    for d in item.get("distractor_pool", []):
        td_raw = translate_to_greek(_pre_anchor_en(d))

        td_raw = _maybe_anchor(td_raw)
        td = _fix_contextual_mistranslation(td_raw, gr_question)
        if not td:
            continue

        if _is_untranslated_latin(td):
            continue
        tdl = td.strip().lower()
        if tdl and tdl not in seen_pool:
            seen_pool.add(tdl)
            translated_pool.append(td)

    new_item["question"] = gr_question
    new_item["options"] = new_options
    new_item["distractor_pool"] = translated_pool

    translated.append(new_item)

return translated
```

Εικόνα 24 Μετάφραση Single Choice (en->el)

Πέρα από τις rule-based διορθώσεις, η μετάφραση κάθε πεδίου περνά από δύο επιπλέον στρώματα που διαφαίνεται παραπάνω. Πρώτον, εφαρμόζεται το anchor dictionary: τα κύρια ονόματα του πρωτότυπου κειμένου προστατεύονται ήδη μέσα στο αγγλικό κείμενο πριν τη μετάφραση (_pre_anchor_en) και, μετά τη μετάφραση, οι πρωτότυποι όροι επαναφέρονται

(_maybe_anchor), ώστε ένα κύριο όνομα ή εξειδικευμένος όρος να μην αλλοιωθεί ούτε από τη MarianMT ούτε από την contextual διόρθωση. Δεύτερον, εφαρμόζεται φίλτρο αμετάφραστων λατινικών (_is_untranslated_latin): distractors που προέρχονται από WordNet/Wikipedia και έμειναν στα αγγλικά (π.χ. «firedamp», «Yoshiki Hayashi») απορρίπτονται, ενώ η σωστή απάντηση διατηρείται πάντα (μπορεί νόμιμα να είναι π.χ. «DNA»). Επειδή αυτό ενδέχεται να μειώσει τον αριθμό των επιλογών, ο μεταφρασμένος distractor_pool κρατά εφεδρικούς υποψήφιους ώστε το item να καταλήγει πάντα με τέσσερις επιλογές.

3.6.2 Έλεγχοι πριν την Εξαγωγή

Πριν την τελική εξαγωγή, εφαρμόζονται τελικοί κανονιστικοί έλεγχοι για να διασφαλιστεί η εγκυρότητα των παραγόμενων αρχείων. Η συνάρτηση _fix_brackets() στο qg/h5p_export.py αντιμετωπίζει συγκεκριμένα ζητήματα που προκύπτουν από τη μετάφραση εξόδου σε fill-in the blanks items.

Συγκεκριμένα, τρία regex patterns αντιμετωπίζουν τρεις διαφορετικές μορφές κακής μορφής που προκύπτουν όταν η μετάφραση αφαιρεί ή επανατοποθετεί τα αγκύλια λανθασμένα, π.χ. μονό κλείσιμο ([[word]]), μονό άνοιγμα ([word]), και κενά αγκύλια ([[]]).

```
def _fix_brackets(text: str) -> str:
    """Διορθώνει malformed [...] brackets στο fill-in κείμενο."""
    import re

    text = re.sub(r'\N([^\N]+)\N(?:\N)?', r'[[\N]]', text)

    text = re.sub(r'(?!\N)\N([^\N]+)\N', r'[[\N]]', text)

    text = re.sub(r'\N[\s*\N]', '', text)
    return text
```

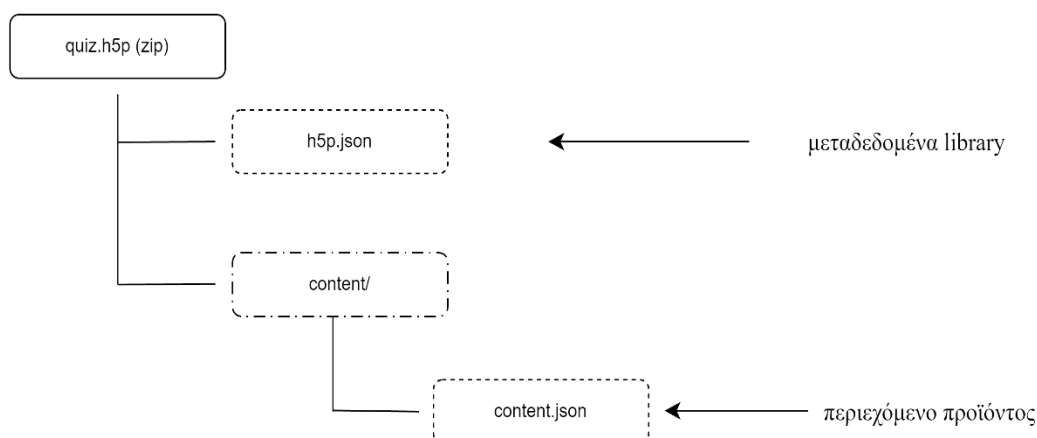
Εικόνα 25 Διόρθωση brackets

Στα Single Choice items, εφαρμόζεται επιπλέον dedup έλεγχος στις τέσσερις επιλογές. Δύο επιλογές που διαφέρουν μόνο σε casing, σε άρθρα ή σε κενά συγχωνεύονται σε μία. Αν αυτό μειώσει τον αριθμό κάτω από τις απαιτούμενες τέσσερις, το σύστημα αναζητά αντικαταστάσεις από το distractor_pool που φυλάχθηκε από το προηγούμενο στάδιο. Ο distractor_pool από τον οποίο αντλούνται οι αντικαταστάσεις δεν είναι ακατέργαστος, έχει ήδη μεταφραστεί, έχει περάσει από αφαίρεση διπλοτύπων (case-insensitive) και από το

φίλτρο αμετάφραστων λατινικών, ώστε κάθε εφεδρικός υποψήφιος που προστίθεται να είναι έγκυρη ελληνική επιλογή.

3.6.3 Εξαγωγή H5P

Το τελευταίο στάδιο του αγωγού μετατρέπει τα έτοιμα items σε πραγματικά αρχεία .h5p που μπορούν να εισαχθούν στο Moodle ή στο Lumi Desktop. Τεχνικά, ένα .h5p αρχείο είναι ZIP archive με συγκεκριμένη εσωτερική δομή:



Διάγραμμα 10 Δομή αρχείου .h5p

fill_in.zip - ZIP archive, unpacked size 1.716 bytes					
Name	Size	Packed	Type	Modified	CRC32
..			File folder		
content	1.441	584	File folder		
h5p.json	275	164	JSON Source File	22/6/2026 7:17...	338894AE

Εικόνα 26 Αρχείο h5p ως zip

Το παρόν σύστημα παράγει δύο διακριτούς τύπους H5P, το H5P.SingleChoiceSet 1.11 για ερωτήσεις πολλαπλής επιλογής, και το H5P.Blanks 1.14 για fill in the blanks. Οι εκδόσεις των libraries επιλέχθηκαν εμπειρικά για συμβατότητα με Moodle 4.x και Lumi Desktop 0.9+.

Στη δομή content.json για SingleChoiceSet, η σωστή απάντηση τοποθετείται πάντα στη θέση 0. Η H5P library αναλαμβάνει την τυχαιοποίηση της σειράς κατά την εμφάνιση στον χρήστη έτσι σε διαφορετικά viewing sessions η σωστή θα εμφανιστεί άλλοτε ως πρώτη, άλλοτε ως δεύτερη κ.ο.κ.

```
{
  "choices": [
    {
      "subContentId": "79b0d521-08b4-4ea2-bc3d-b15bb594b8d7",
      "question": "<p>Who was the king of Macedon in the 4th century BCE?</p>",
      "answers": [
        "<p>Alexander the Great</p>",
        "<p>Pyrrhus of Epirus</p>",
        "<p>Philip II of Macedon</p>",
        "<p>Aristotle</p>"
      ]
    },
    {
      "subContentId": "f962ae96-8ee2-48d6-b29b-eef5a8acfc61",
      "question": "<p>Where was Alexander the Great a king?</p>",
      "answers": [
        "<p>Macedon</p>",
        "<p>Egypt</p>",
        "<p>the Persian Empire</p>",
        "<p>Greece</p>"
      ]
    }
  ],
}
```

Εικόνα 27 Content Single Choice

Για Fill-in items, η σύνταξη του H5P χρησιμοποιεί ένα ενιαίο string κειμένου όπου τα κενά σημειώνονται με τη μορφή *answer*. Το σύστημα μετατρέπει εσωτερικά τη δική του σύνταξη [[answer]] σε αυτή την H5P σύνταξη πριν την εξαγωγή.

```
{
  "media": {
    "disableImageZooming": false
  },
  "text": "<p>Fill in the Blanks</p>",
  "questions": [
    "<p>Alexander the Great was a king of *Macedon* who lived in the 4th century BCE .</p>",
    "<p>He became king when he was very young and quickly proved to be a talented *leader* .</p>",
    "<p>He was taught by the philosopher *Aristotle* , which helped shape his ideas about culture , learning , and leadership .</p>"
  ]
}
```

Εικόνα 28 Content Fill in the blanks

Παράλληλα με τα H5P αρχεία, παράγονται και απλά JSON dumps των items στο ίδιο directory. Δεν είναι μέρος του H5P format, αλλά εργαλεία επιθεώρησης που επιτρέπουν στον developer να ελέγξει το περιεχόμενο χωρίς να ανοίξει Lumi. Χρησιμοποιούνται επίσης από τον κώδικα αξιολόγησης του επόμενου σταδίου.

Σε κάθε εκτέλεση, οι έξοδοι αποθηκεύονται σε ξεχωριστά subfolders ανάλογα με τον client:

- Από app.py (Gradio): data/output/gradio_ui/
- Από main.py (CLI): data/output/cli/

Αυτή η οργάνωση εξασφαλίζει ότι δεν αναμειγνύονται outputs από διαφορετικούς τρόπους εκτέλεσης.

3.7 Αξιολόγηση

Σε αντίθεση με όλα τα προηγούμενα στάδια, η αξιολόγηση δεν αποτελεί μέρος της κανονικής ροής παραγωγής. Εκτελείται μέσω του ξεχωριστού script evaluate.py, που συγκρίνει τις παραγόμενες ερωτήσεις του συστήματος (system_questions.txt) με ένα σύνολο ερωτήσεων αναφοράς (gold standard) γραμμένων χειροκίνητα από εκπαιδευτικό. Σκοπός είναι μια ποσοτική ένδειξη ποιότητας που επιτρέπει τη σύγκριση διαφορετικών διαμορφώσεων του pipeline μεταξύ τους και όχι μια απόλυτη βαθμολογία.

3.7.1 Αρχιτεκτονική ROUGE

Η κεντρική μετρική είναι η ROUGE (Recall-Oriented Understudy for Gisting Evaluation), στις τρεις παραλλαγές της: ROUGE-1, ROUGE-2, ROUGE-L. Υλοποιείται από μηδενική βάση στο evaluation/metrics.py καθώς η Google rouge_score που χρησιμοποιείται συνήθως δεν υποστηρίζει επαρκώς Unicode χαρακτήρες. Σε δοκιμή με ελληνικά κείμενα, έδινε σχεδόν μηδενικά scores ακόμη και για πανομοιότυπες ερωτήσεις. Με re.UNICODE, το \w+ αναγνωρίζει σωστά τα ελληνικά γράμματα ως word characters.

```
def tokenize(text: str) -> List[str]:  
    return re.findall(r'\w+', text.lower(), re.UNICODE)
```

Εικόνα 29 Unicode aware tokenizer

Επιπλέον, η compute_rouge_scores υποστηρίζει best-match στρατηγική σύγκρισης αντί για αυστηρό positional matching (η ισοτή παραγόμενη ερώτηση με την ισοτή gold), κάθε παραγόμενη ερώτηση συγκρίνεται με όλες τις gold και διατηρείται το καλύτερο score. Αυτή

η προσέγγιση είναι πιο δίκαιη για AQG, καθώς το pipeline δεν εγγυάται ίδια σειρά ή ίδιο πλήθος ερωτήσεων με το gold standard.

```
def compute_rouge_scores(
    references: List[str],
    hypotheses: List[str],
    best_match: bool = False,
) -> Dict[str, float]:

    if not references or not hypotheses:
        return {"rouge1": 0.0, "rouge2": 0.0, "rougeL": 0.0}

    totals: Dict[str, float] = {"rouge1": 0.0, "rouge2": 0.0, "rougeL": 0.0}
```

Η εκτέλεση γίνεται από τη γραμμή εντολών:

```
python evaluate.py --gold data/evaluation/internet_en.txt \
    --system data/output/cli/run_main/system_questions.txt \
    --best_match
```

3.7.2 Αποτελέσματα και παρατηρήσεις

Για την αξιολόγηση εκτελέστηκε συστηματικό πείραμα σε σώμα οκτώ εκπαιδευτικών κειμένων που καλύπτουν διαφορετικούς τομείς γνώσης και τις δύο γλώσσες υποστήριξης του συστήματος.

Κείμενο	Γλώσσα	Θεματική	Gold Ερωτήσεις
alexander the great	EN	Ιστορία	8
ancient rome en	EN	Ιστορία	10
heart	EN	Βιολογία	10
internet en	EN	Τεχνολογία	10
olympics	EN	Ιστορία	10
romeo juliet	EN	Λογοτεχνία	10
dna gr	EL	Βιολογία	10
japan gr	EL	Γεωγραφία	10

Πίνακας 5 Κείμενα εφαρμογής ROUGE

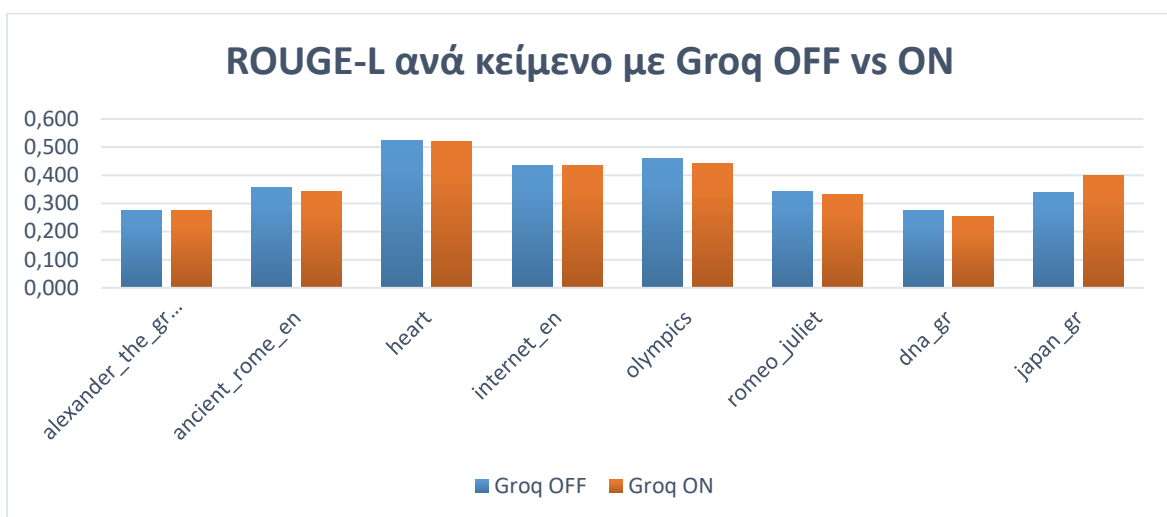
Για κάθε κείμενο το pipeline εκτελέστηκε δύο φορές, μία με ενεργοποιημένο και μία με απενεργοποιημένο το LLM post-processing μέσω Groq. Αυτό επιτρέπει να απομονωθεί η συνεισφορά του LLM σταδίου από την υπόλοιπη αρχιτεκτονική, οδηγώντας σε 16 σημεία μέτρησης (8 κείμενα $\times$ 2 διαμορφώσεις).

Μια σημείωση είναι ότι η διαμόρφωση χωρίς Groq ήταν πλήρως ντετερμινιστική, δηλαδή επαναλαμβανόμενες εκτελέσεις έδιναν πανομοιότυπα αποτελέσματα, καθώς τα Stanza, T5 και οι κανονιστικοί έλεγχοι παράγουν σταθερή έξοδο. Αντίθετα, η διαμόρφωση με Groq ενεργοποιημένο εμπεριέχει μια μικρή διακύμανση καθώς το Groq, ως υπηρεσία LLM, μπορεί να επιστρέψει ελαφρώς διαφορετικές διατυπώσεις μεταξύ εκτελέσεων. Σε επαληθευτικές επαναλήψεις παρατηρήθηκε διακύμανση της τάξης του ± 0.01 έως ± 0.06 στο ROUGE-L των ελληνικών κειμένων.

Τα αποτελέσματα ROUGE-L (η πιο αντιπροσωπευτική παραλλαγή για ερωτήσεις) των 16 εκτελέσεων συνοψίζονται παρακάτω.

Κείμενο	Groq OFF	Groq ON	Μεταβολή
alexander the great	0.275	0.310	+12.8%
ancient_rome_en	0.357	0.330	-7.6%
heart	0.524	0.519	-1.0%
internet_en	0.436	0.436	=
olympics	0.459	0.443	-3.5%
romeo_juliet	0.343	0.332	-3.3%
dna_gr	0.274	0.266	-2.9%
japan_gr	0.340	0.342	+0.6%

Πίνακας 6 Αποτελέσματα ROUGE L



Διάγραμμα 11 Γράφημα ROUGE L

Συγκεντρωτικά σε μέσους όρους ανά γλώσσα:

Κείμενο	Groq OFF	Groq ON	Μεταβολή
Αγγλικά (n=6)	0.399	0.395	-1.0%
Ελληνικά (n=2)	0.307	0.304	-1.0%

Συνολικά (n=8)	0.376	0.372	-1.0%
----------------	-------	-------	-------

Πίνακας 7 Μέσος όρος ανά γλώσσα

Από τα δεδομένα προκύπτουν δύο παρατηρήσεις.

Πρώτον: Στο σύνολο, το ROUGE παραμένει ουσιαστικά σταθερό (-1.0%) με την ενεργοποίηση του Groq, σταθερά και στις δύο γλώσσες. Αυτό δεν σημαίνει ότι το LLM στάδιο είναι περιττό, σημαίνει ότι η ROUGE μετρά κάτι διαφορετικό από αυτό που βελτιώνει κυρίως το Groq. Οι ερωτήσεις του T5 είναι συχνά ήδη γραμματικά αποδεκτές, οπότε το Groq επεμβαίνει κυρίως αναδιατυπώνοντας και αντικαθιστώντας αδύναμα distractors. Η ROUGE, ως μετρική n-gram επικάλυψης, «τιμωρεί» την αλλαγή λεξιλογίου της ερώτησης ακόμη κι όταν αυτή τη βελτιώνει, ενώ αγνοεί εντελώς τη βελτίωση στην ποιότητα των εναλλακτικών επιλογών

Δεύτερον: Η επίδραση του Groq είναι ετερογενής ανά κείμενο. Σε ορισμένες περιπτώσεις βελτιώνει αισθητά (το alexander_the_great ανεβαίνει +12.8%, καθώς το Groq διορθώνει αδύναμα distractors και αναδιατυπώνει ασαφείς ερωτήσεις), σε άλλες διατηρεί την ποιότητα (internet_en, japan_gr), και σε ορισμένες τη μειώνει ελαφρώς ως προς τη ROUGE (ancient_rome_en, olympics). Η μείωση δεν αντιστοιχεί σε υποβάθμιση των ερωτήσεων: όπου το T5 είχε ήδη παράξει διατυπώσεις λεξιλογικά κοντινές στο gold, η αναδιατύπωση του Groq, ακόμη κι αν είναι σημασιολογικά ισοδύναμη ή καλύτερη, απομακρύνεται από το ακριβές λεξιλόγιο και «τιμωρείται» από τη μετρική. Σημαντικότερο είναι ότι η ποιότητα διατηρείται και στα ελληνικά κείμενα (μέση μεταβολή -1.0%, ίδια με τα αγγλικά) παρά τη διπλή μετάφραση el→en→el που μεσολαβεί.

4. Υλοποίηση και Λειτουργία

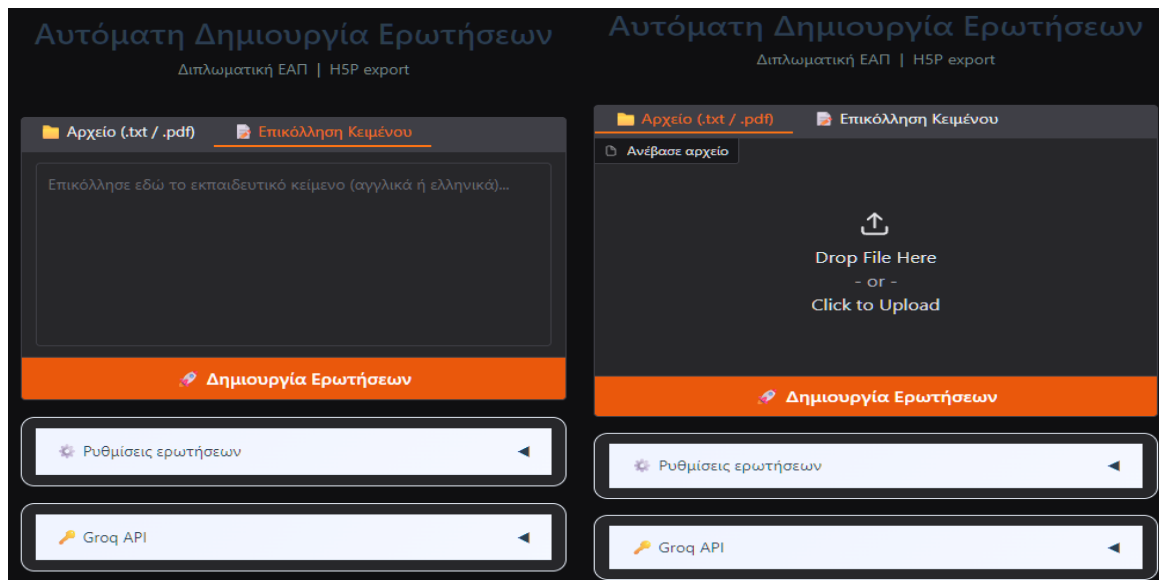
4.1 Λειτουργία μέσω Web UI

Η διαδικτυακή διεπαφή είναι ο πρωτεύων τρόπος αλληλεπίδρασης για εκπαιδευτικούς. Δεν απαιτεί τεχνική γνώση πέρα από βασική χρήση browser, καθώς αυτοσκοπός είναι η δυνατότητα παραγωγής των προϊόντων άμεσα και σε ικανοποιητικό χρόνο. Η εφαρμογή ξεκινά σε τοπικό web server στη διεύθυνση <http://127.0.0.1:7860> και η κύρια οθόνη παρέχει δύο εναλλακτικούς τρόπους εισόδου:

- Tab «Κείμενο»: ο χρήστης επικολλά εκπαιδευτικό κείμενο σε πεδίο κειμένου

- Tab «Αρχείο»: ο χρήστης ανεβάζει αρχείο .txt ή .pdf (μέγιστο 15 σελίδες για PDF)

Η ανίχνευση γλώσσας γίνεται αυτόματα μετά την υποβολή και ο χρήστης δεν χρειάζεται να δηλώσει αν το κείμενο είναι ελληνικό ή αγγλικό.



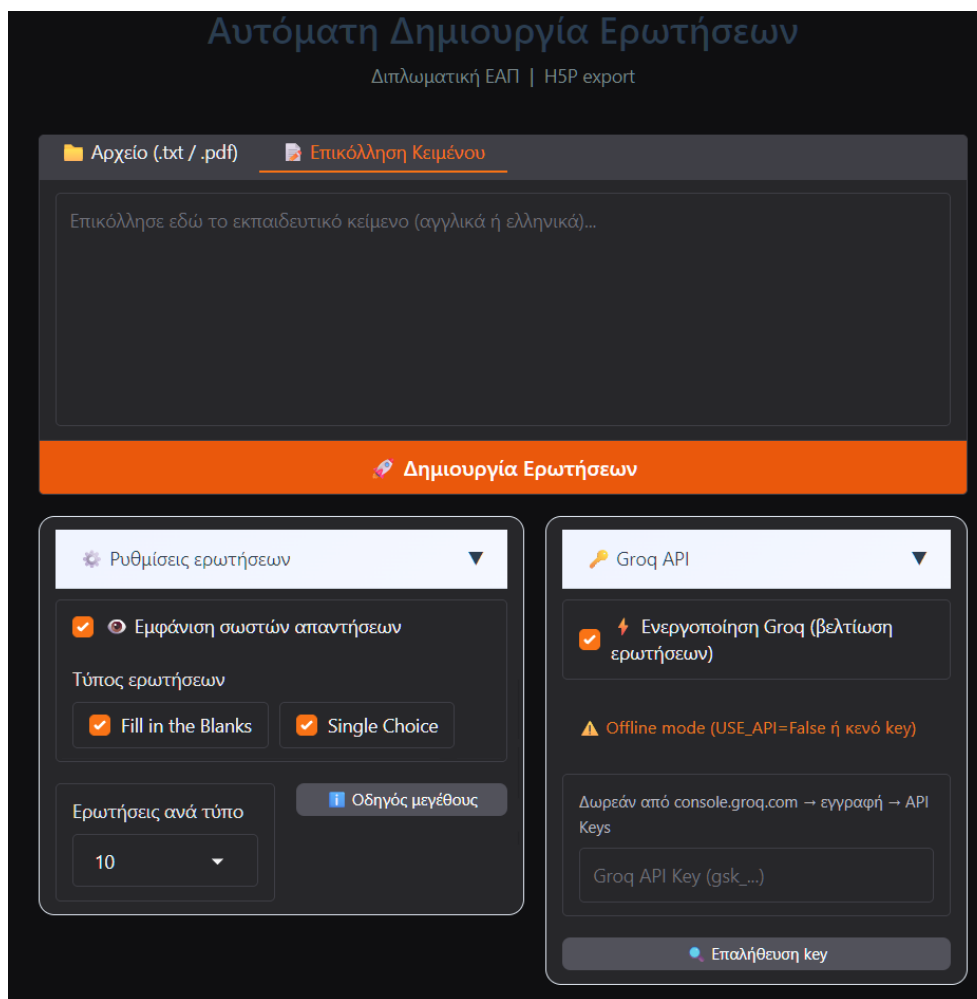
Εικόνα 30 Επιλογές εισόδου αρχείου/κειμένου

4.2 Παραμετροποίηση

Πριν την παραγωγή, ο χρήστης μπορεί να ρυθμίσει δύο βασικές παραμέτρους:

Το **Groq toggle** που ενεργοποιεί ή απενεργοποιεί το LLM post-processing με την καταχώριση ενός API key, το οποίο επαληθεύεται αν είναι έγκυρο πριν το «τρέξιμο» της εφαρμογής. Με την εφαρμογή του διαπιστώνεται βελτίωση της ποιότητας των ερωτήσεων, ωστόσο ο χρόνος αυξάνεται αναλόγως του αριθμού των προτάσεων που έχει να διαχειριστεί και της γλώσσας του αρχικού κειμένου.

Οι **Ρυθμίσεις ερωτήσεων** δίνουν τη δυνατότητα στον χρήστη να επιλέξει αν κατά την παραγωγή θα εμφανιστούν οι απαντήσεις, αν θα παραχθούν και οι δυο τύποι ταυτόχρονα ή έκαστος ξεχωριστά όπως και τον αριθμό των ερωτήσεων που επιθυμεί.



Εικόνα 31 Παραμετροποίηση

Υπάρχει η δυνατότητα παραγωγής από 5,8,10,12 και 15 ερωτήσεων τα οποία εμφανίζονται σε ένα dropdown menu. Πατώντας στην προεπισκόπηση του οδηγού μεγέθους, δίνονται οδηγίες για τον προσεγγιστικό και συνιστώμενο όριο ερωτήσεων που θα παραχθούν βάσει του μεγέθους του αρχικού κειμένου με το οποίο τροφοδοτήθηκε η είσοδος της εφαρμογής.

Ρυθμίσεις ερωτήσεων

Groq API

☒ Εμφάνιση σωστών απαντήσεων

Τύπος ερωτήσεων

☒ Fill in the Blanks
☒ Single Choice

✓ 10

10

Οδηγός μεγέθους

Οδηγός — Πόσες ερωτήσεις να περιμένω;

Ο αριθμός που επιλέγεις είναι το **ανώτατο όριο ανά τύπο** (fill-in και single-choice ξεχωριστά). Αν το κείμενο δεν έχει αρκετές κατάλληλες προτάσεις, το σύστημα παράγει αυτόματα λιγότερες.

Μέγεθος κειμένου	Αναμενόμενες ερωτήσεις / τύπο	Συνιστώμενο όριο
Μικρό (~200 λέξεις)	3 – 5	5
Μεσαίο (~400 λέξεις)	6 – 8	8
Κανονικό (~600 λέξεις)	8 – 10	10 ★
Μεγάλο (~800+ λέξεις)	10 – 12	12 – 15

★ Προεπιλογή. Πάνω από 12 ερωτήσεις η ποιότητα τείνει να μειώνεται.

Εικόνα 32 Ρυθμίσεις ερωτήσεων

4.3 Παραγωγή και download H5P αρχείων

Κατά την παραγωγή τα επιλεγμένα προϊόντα παρατίθενται με τη μορφή καρτών για τον πρώτο έλεγχο από την πλευρά του χρήστη. Δίνονται και στοιχεία που αφορούν τον χρόνο που χρειάστηκε για να παραχθεί σε κάθε στάδιο όπως και αν η εφαρμογή έτρεξε στη CPU ή GPU του υπολογιστή του χρήστη

Αυτόματη Δημιουργία Ερωτήσεων

Διπλωματική ΕΑΠ | H5P export

📁 Αρχείο (.txt / .pdf) 📄 Επικόλληση Κειμένου

Alexander the Great was a king of Macedon who lived in the 4th century BCE. He became king when he was very young and quickly proved to be a talented leader. He was taught by the philosopher Aristotle, which helped shape his ideas about culture, learning, and leadership.

Alexander is famous for building one of the largest empires in the ancient world. He led his army from Greece into Asia, defeating the Persian Empire in several major battles. His conquests reached as far as Egypt and parts of India, and he founded many cities along the way, including Alexandria in Egypt.

Even after his death, Alexander's influence remained strong. The spread of Greek language and culture through his empire helped create the Hellenistic world, where different cultures mixed and exchanged ideas. Today, he is remembered as a brilliant military commander and one of history's most important figures.

🔧 Δημιουργία Ερωτήσεων

⚙️ Ρυθμίσεις ερωτήσεων

☑️ Εμφάνιση σωστών απαντήσεων

Τύπος ερωτήσεων

☑️ Fill in the Blanks
☑️ Single Choice

Ερωτήσεις ανά τύπο

5

ℹ️ Οδηγός μεγέθους

🔑 Groq API

⚡ ☐ Ενεργοποίηση Groq (βελτίωση ερωτήσεων)

☐ Groq ανενεργό (offline mode)

Δωρεάν από console.groq.com → εγγραφή → API Keys

.....

🔑 Επαλήθευση key

🌐 Γλώσσα: gb Αγγλικά
📄 Προτάσεις: 9
✏️ Fill-in: 5
● Single Choice: 5

🕒 **Σύνολο: 8.1s**

🕒 **ΑΝΑΛΥΣΗ ΧΡΟΝΟΥ ΑΝΑ ΣΤΑΔΙΟ**

stanza (NER/POS)		1.0s (13%)
T5 (παραγωγή)		6.6s (87%)

Συνολικός χρόνος: **7.6s** | Συσκευή: GPU (CUDA)

✏️ Fill in the Blanks
5

ΕΡΩΤΗΣΗ #1

Alexander the Great was a king of Macedon who lived in the 4th century BCE .

● Single Choice
5

ΕΡΩΤΗΣΗ #1

Who was king of Macedon in the 4th century BCE?

✕ Alexander 's

Εικόνα 33 Εμφάνιση παραγωγής

Στο κάτω μέρος της παραγωγής εμφανίζονται τα αρχεία που ο χρήστης επιθυμεί να χρησιμοποιήσει και τα κατεβάζει ως μορφή .h5p.

The screenshot displays the H5P interface with two columns of quiz questions. The left column, titled 'Fill in the Blanks', contains five questions where the user must fill in missing words. The right column, titled 'Single Choice', contains five multiple-choice questions. The correct answers are highlighted in green in the 'Single Choice' column.

Fill in the Blanks Questions:

- ΕΡΩΤΗΣΗ #1: Alexander the Great was a king of **Macedon** who lived in the 4th century BCE .
- ΕΡΩΤΗΣΗ #2: Alexander the Great was a **king** of Macedon who lived in the 4th century BCE .
- ΕΡΩΤΗΣΗ #3: He became king when he was very young and quickly proved to be a talented **leader** .
- ΕΡΩΤΗΣΗ #4: He was taught by the philosopher **Aristotle** , which helped shape his ideas about culture , learning , and leadership .
- ΕΡΩΤΗΣΗ #5: He was taught by the **philosopher** Aristotle , which helped shape his ideas about culture , learning , and leadership .

Single Choice Questions:

- ΕΡΩΤΗΣΗ #1: Who was the king of Macedon in the 4th century BCE?
☐ X Greece
☒ Alexander the Great
☐ X Philip II of Macedon
☐ X Aristotle
- ΕΡΩΤΗΣΗ #2: Where was Alexander the Great a king?
☐ X Egypt
☐ X Greece
☐ X the Persian Empire
☒ Macedon
- ΕΡΩΤΗΣΗ #3: What was Alexander the Great's title?
☐ X tsar
☐ X czar
☒ king
☐ X leader
- ΕΡΩΤΗΣΗ #4: Who taught Alexander the Great?
☐ X Macedon
☐ X Alexander the Great
☒ Aristotle
- ΕΡΩΤΗΣΗ #5: What was Aristotle's profession?
☒ philosopher
☐ X leadership
☐ X learning
☐ X culture

Κατέβασε τα .h5p αρχεία (άνοιξε στο Lumi):

Single Choice (.h5p) 1.1 KB ↓

Fill in the Blanks (.h5p) 978.0 B ↓

Εικόνα 34 Εμφάνιση παραγωγής

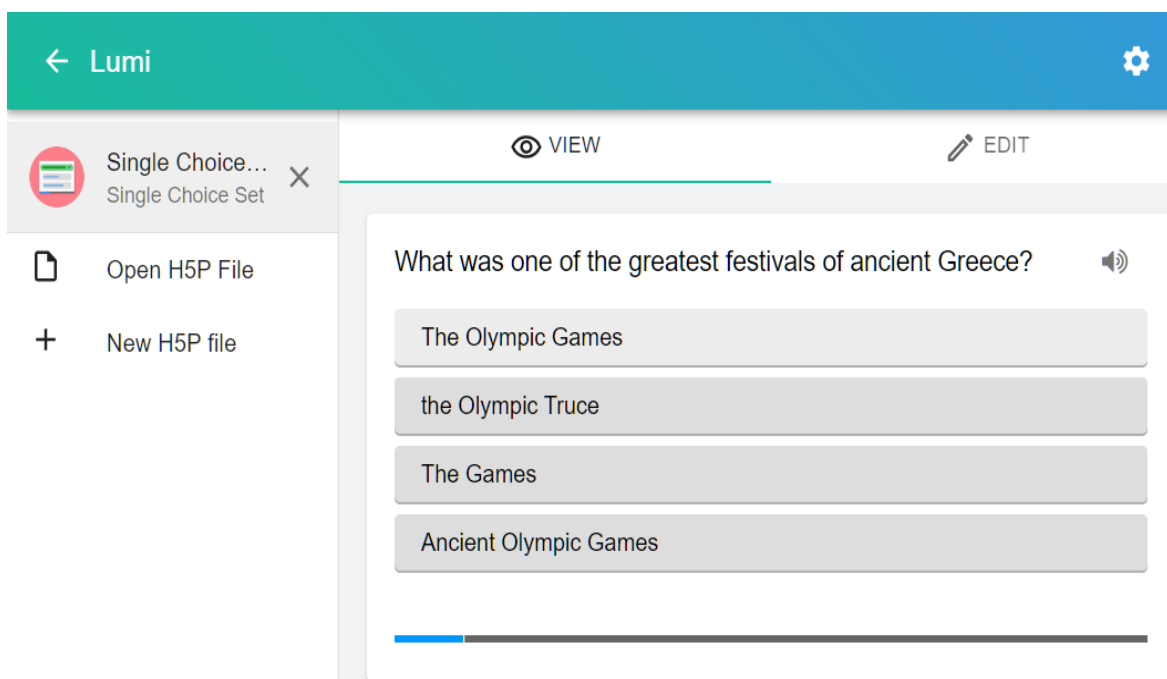
Τα παραγόμενα αρχεία αποθηκεύονται στον φάκελο `data/output/cli/<run_name>/` και περιλαμβάνουν:

- `single_choice.h5p` και `fill_in.h5p`
- `system_questions.txt`
- 2 αρχεία.json

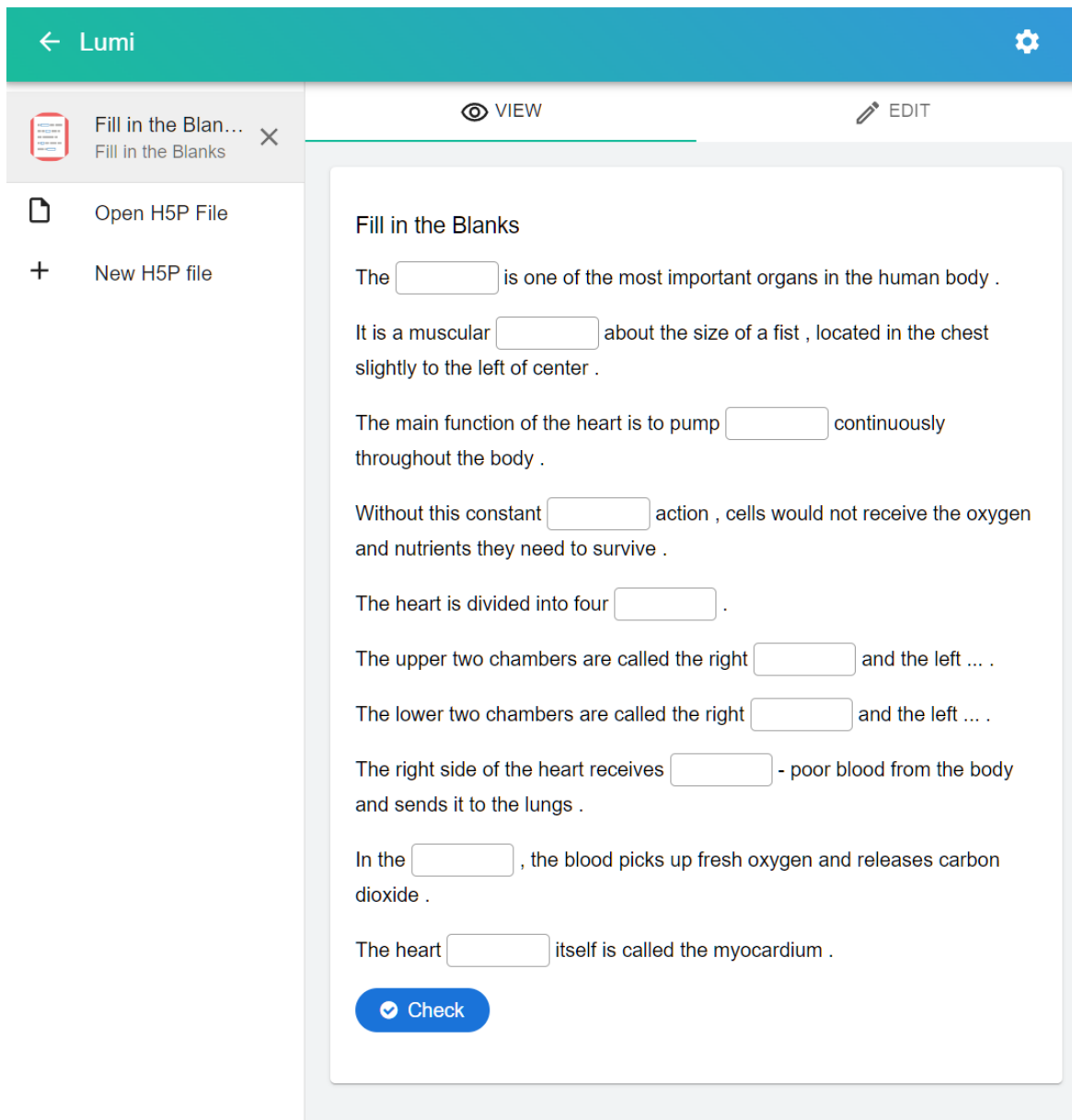
Παράλληλα με τα H5P αρχεία, παράγονται και απλά JSON dumps των items (`single_choice_content.json`, `fill_in_content.json`) στο ίδιο directory. Δεν είναι μέρος του H5P format, αλλά εργαλεία επιθεώρησης που επιτρέπουν στον developer να ελέγξει το περιεχόμενο χωρίς να ανοίξει Lumi. Για την αξιολόγηση του επόμενου σταδίου, το `evaluate.py` διαβάζει ξεχωριστά το αρχείο `system_questions.txt` (μία ερώτηση ανά γραμμή).

4.5 Χρήση σε Lumi και Moodle

Έχοντας εγκατεστημένη την εφαρμογή του Lumi και τα αρχεία .h5p, δίνεται η άμεση δυνατότητα ανοίγματος των υπόψη αρχείων προκειμένου να προεπισκοπηθούν πριν την εισαγωγή τους σε οποιοδήποτε LMS (π.χ. Moodle).



Εικόνα 37 Preview αρχείου single choice



← Lumi

VIEW EDIT

Fill in the Blanks

Open H5P File

New H5P file

Fill in the Blanks

The is one of the most important organs in the human body .

It is a muscular about the size of a fist , located in the chest slightly to the left of center .

The main function of the heart is to pump continuously throughout the body .

Without this constant action , cells would not receive the oxygen and nutrients they need to survive .

The heart is divided into four .

The upper two chambers are called the right and the left

The lower two chambers are called the right and the left

The right side of the heart receives - poor blood from the body and sends it to the lungs .

In the , the blood picks up fresh oxygen and releases carbon dioxide .

The heart itself is called the myocardium .

✓ Check

Εικόνα 38 Preview αρχείου fill in the blanks

Αντίστοιχα δίνεται η δυνατότητα τροποποίησης και επεξεργασίας των αποτελεσμάτων σε περίπτωση διαπίστωσης σφάλματος ή έλλειψης σε distractor. Το γεγονός αυτό έρχεται να επικαλύψει τις αδυναμίες του συστήματος, ειδικά κατά την παραγωγή ελληνικών ερωτήσεων, όπου εμφανίζονται τα περισσότερα σφάλματα.

Single Choice...
Single Choice Set

Open H5P File

New H5P file

VIEW

EDIT

H5P Single Choice Set

Tutorial Example Copy Paste & Replace

Single Choice Set

Title * Metadata

Used for searching, reports and copyright information

Single Choice Quiz

List of questions * Textual Default

What was one of the greatest festivals of anc...

Question *

What was one of the greatest festivals of ancient Greece?

Alternatives - first alternative is the correct one. *

Alternative

The Olympic Games

Alternative

Ancient Olympic Games

Alternative

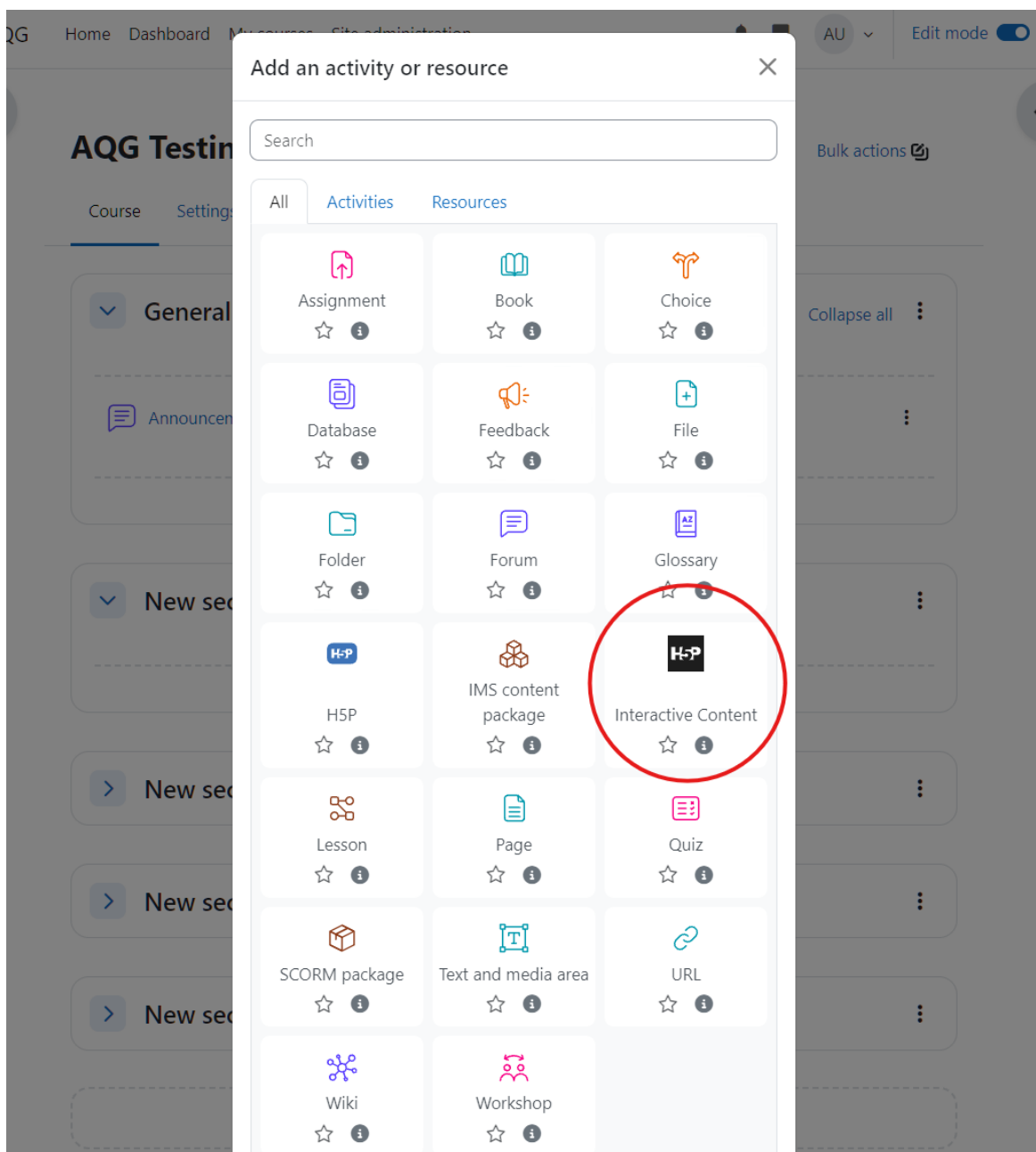
The Games

Alternative

the Olympic Truce

Εικόνα 39 Επεξεργασία αρχείου .h5p

Κατόπιν των παραπάνω τοπικών ρυθμίσεων το αρχείο δύναται να εισαχθεί στο Moodle course, αφού έχει ήδη προστεθεί το activity/library σύμφωνα με το documentation του ιστοτόπου moodle.org για την υποδοχή αυτών των αρχείων.



Εικόνα 40 Επιλογή activity σε Moodle course

Αφού επιλεγεί το activity, δίνεται η δυνατότητα περαιτέρω τροποποιήσεων και επεξεργασίας βάσει της επιθυμίας του κάθε διδάσκοντα.

**Fill in the Blanks****Fill in the Blanks**

The is one of the most important organs in the human body .

It is a muscular about the size of a fist , located in the chest slightly to the left of center .

The main function of the heart is to pump continuously throughout the body .

Without this constant action , cells would not receive the oxygen and nutrients they need to survive .

The heart is divided into four .

The upper two chambers are called the right and the left

The lower two chambers are called the right and the left

The right side of the heart receives - poor blood from the body and sends it to the lungs .

In the , the blood picks up fresh oxygen and releases carbon dioxide .

The heart itself is called the myocardium .

Check

Εικόνα 41 Προεπισκόπηση σε Moodle course

5. Συμπεράσματα

5.1 Σύνοψη

Στόχος της παρούσας Διπλωματικής Εργασίας ήταν η ανάπτυξη ενός συστήματος αυτόματης δημιουργίας ερωτήσεων (AQG) από εκπαιδευτικό κείμενο, με αξιοποίηση τεχνικών Επεξεργασίας Φυσικής Γλώσσας και μοντέλων βαθιάς μάθησης, και τελική παραγωγή διαδραστικών κουίζ σε μορφή H5P, άμεσα αξιοποιήσιμων σε Συστήματα Διαχείρισης Μάθησης όπως το Moodle. Συγκρίνοντας τους αρχικούς στόχους με το τελικό αποτέλεσμα, διαπιστώνεται ότι καλύφθηκαν στο σύνολό τους:

Αυτοματοποιημένος αγωγός (pipeline) εισόδου txt/pdf. Υλοποιήθηκε πλήρης ροή που δέχεται κείμενο και αρχεία txt/pdf, εκτελεί ανίχνευση γλώσσας και προετοιμασία, και παράγει ερωτήσεις δύο τύπων, συμπλήρωσης κενού (fill-in-the-blanks) και μονής επιλογής (single choice).

Γλωσσολογική ανάλυση με Stanza. Ενσωματώθηκε η Stanza για tokenization, POS tagging, λημματοποίηση και αναγνώριση ονομαστικών οντοτήτων (NER), παρέχοντας τη βάση για την επιλογή κατάλληλων υποψήφιων όρων ως απαντήσεων.

Παραγωγή ερωτήσεων με T5. Αξιοποιήθηκε το προεκπαιδευμένο μοντέλο valhalla/t5-base-qg-hl με τη λογική του highlight format, ώστε η παραγωγή να εστιάζει στον προεπιλεγμένο answer span.

Παραγωγή distractors. Σχεδιάστηκε υβριδικός μηχανισμός παραγωγής εναλλακτικών επιλογών που συνδυάζει αλγοριθμικά φίλτρα, σημασιολογικούς πόρους (WordNet), εξωτερική γνώση (Wikipedia API) και δεξαμενή οντοτήτων από το ίδιο το κείμενο (cross-sentence pooling).

Εξαγωγή έγκυρων αρχείων H5P. Το σύστημα παράγει πραγματικά αρχεία .h5p (μορφή ZIP) για τους τύπους H5P.Single Choice Set και H5P.Blanks, τα οποία ελέγχθηκαν επιτυχώς στο Lumi Desktop και είναι συμβατά με το Moodle.

Φιλικό γραφικό περιβάλλον. Αναπτύχθηκε διεπαφή μέσω Gradio που επιτρέπει άμεση χρήση, ενώ παράλληλα διατηρήθηκε και διεπαφή γραμμής εντολών (CLI).

Ποσοτική αξιολόγηση και συζήτηση περιορισμών. Διενεργήθηκε συστηματική αξιολόγηση με τη μετρική ROUGE σε εκπαιδευτικά κείμενα, συνοδευόμενη από κριτική συζήτηση των ορίων της αυτόματης μετρικής.

Συνολικά, το σύστημα που υλοποιήθηκε καλύπτει ολόκληρη την αλυσίδα από το ακατέργαστο κείμενο έως το έτοιμο προς χρήση διαδραστικό κουίζ, επιβεβαιώνοντας τη βιωσιμότητα της προτεινόμενης υβριδικής προσέγγισης.

5.2 Περιορισμοί και αποτίμηση

Το παρόν σύστημα, ως διπλωματική εργασία με περιορισμένους πόρους, δεν φιλοδοξεί να ανταγωνιστεί εμπορικά συστήματα AQG ή υπάρχοντα συστήματα παρόμοιων δυνατοτήτων. Αντίθετα η εργασία έχει εκπαιδευτικό και ερευνητικό χαρακτήρα, όπου μέσα από τον σχεδιασμό και την υλοποίηση ενός ολοκληρωμένου αγωγού Αυτόματης Δημιουργίας Ερωτήσεων, επιδιώχθηκε η κατανόηση και πρακτική εφαρμογή τεχνικών Επεξεργασίας Φυσικής Γλώσσας και AQG.

Πέρα από την επίτευξη των παραπάνω σκοπών εντοπίζονται τρεις συγκεκριμένοι περιορισμοί:

Πρώτον, η εξάρτηση από προεκπαιδευμένα μοντέλα γενικού σκοπού. Το μοντέλο T5 που χρησιμοποιείται είναι μικρό και δεν έχει εκπαιδευτεί στο συγκεκριμένο εκπαιδευτικό περιεχόμενο, με αποτέλεσμα να παράγει περιστασιακά ερωτήσεις με γραμματικές ατέλειες ή ασαφή διατύπωση. Οι κανονιστικοί έλεγχοι και ο προαιρετικός έλεγχος μέσω LLM μετριάζουν αλλά δεν εξαλείφουν πλήρως αυτές τις περιπτώσεις.

Δεύτερον, η ευαισθησία στην ποιότητα της μετάφρασης για τα ελληνικά. Η αρχιτεκτονική «sandwich» καθιστά εφικτή την υποστήριξη ελληνικών, όμως η διπλή διέλευση από τη μηχανική μετάφραση εισάγει σημεία αστάθειας, όπως όροι που δύναται να αλλοιωθούν, και ορισμένες περιπτώσεις δεν καλύπτονται πλήρως ούτε από το από τις κανονιστικές διορθώσεις. Ως εκ τούτου, η ποιότητα στα ελληνικά κείμενα εμφανίζει μεγαλύτερη διακύμανση σε σχέση με τα αγγλικά, και τα παραγόμενα αποτελέσματα ωφελούνται από ανθρώπινη επιθεώρηση πριν τη χρήση.

Τρίτον, η μη σταθερή φύση του προαιρετικού LLM σταδίου. Η διαμόρφωση με Groq εμφανίζει μικρή διακύμανση μεταξύ εκτελέσεων, αναλόγως κάθε υπηρεσίας LLM. Αυτό

σημαίνει ότι, με ενεργοποιημένο το LLM στάδιο, δύο εκτελέσεις στο ίδιο κείμενο ενδέχεται να μην δώσουν πανομοιότυπες ερωτήσεις. Πρόκειται για συνειδητό συμβιβασμό σύμφωνα με το οποίο το LLM στάδιο προσφέρει βελτίωση ποιότητας με αντάλλαγμα την προβλεψιμότητα αυτής.

Είναι σημαντικό να τονιστεί ότι κανένας από αυτούς τους περιορισμούς δεν αναιρεί τη χρηστικότητα του συστήματος για τον σκοπό που σχεδιάστηκε και λειτουργούν ως δείκτες κατεύθυνσης για μελλοντική βελτίωση.

5.3 Προτάσεις για μελλοντική εργασία

Από τους παραπάνω περιορισμούς προκύπτουν και οι κατευθύνσεις περαιτέρω ανάπτυξης, εκ των οποίων οι σημαντικότερες είναι:

Η αντικατάσταση του γενικού T5 με μοντέλο εξειδικευμένα εκπαιδευμένο (fine-tuned) σε ζεύγη εκπαιδευτικού κειμένου–ερωτήσεων που αναμένεται να μειώσει σημαντικά τις γραμματικές ατέλειες και να βελτιώσει τη συνάφεια, ελαττώνοντας την εξάρτηση από το τους κανονιστικούς σχεδιασμούς διόρθωσης.

Η αξιοποίηση ελληνικών ή πολυγλωσσικών μοντέλων παραγωγής ερωτήσεων που ωριμάζουν με την πάροδο των ετών, θα εξάλειφε την αρχιτεκτονική «sandwich» και τις αλλοιώσεις της διπλής μετάφρασης, αντιμετωπίζοντας στη ρίζα την αστάθεια στα ελληνικά.

Η μετάβαση από το cloud Groq σε τοπικά εκτελούμενο LLM (π.χ. μέσω Ollama) θα αφαιρούσε την εξάρτηση από εξωτερικό API και τη σχετική μεταβλητότητα δικτύου, καθιστώντας το σύστημα πλήρως αυτόνομο.

Βιβλιογραφία

Ακολουθούν οι βιβλιογραφικές αναφορές (πηγές) της Εργασίας.

Banerjee, S., & Lavie, A. (2005). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization (pp. 65–72). Association for Computational Linguistics.

Das, B., Majumder, M., Phadikar, S., & Sekh, A. A. (2021). Automatic question generation and answer assessment: A survey. Research and Practice in Technology Enhanced Learning, 16(1), Article 5. <https://doi.org/10.1186/s41039-021-00151-1>

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 4171–4186). Association for Computational Linguistics. <https://doi.org/10.18653/v1/N19-1423>

Dougiamas, M., & Taylor, P. C. (2003). Moodle: Using learning communities to create an open source course management system. In Proceedings of EdMedia 2003 (pp. 171–178). Association for the Advancement of Computing in Education (AACE).

Explosion AI. (n.d.). spaCy: Industrial-strength natural language processing. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://spacy.io/>

Fellbaum, C. (2010). WordNet. In R. Poli, M. Healy, & A. Kameas (Eds.), Theory and applications of ontology: Computer applications (pp. 231–243). Springer. https://doi.org/10.1007/978-90-481-8847-5_10

Gamage, S. H. P. W., Ayres, J. R., & Behrend, M. B. (2022). A systematic review on trends in using Moodle for teaching and learning. International Journal of STEM Education, 9(1), Article 9. <https://doi.org/10.1186/s40594-021-00323-x>

Google. (n.d.). mT5-base [Model card]. Hugging Face. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://huggingface.co/google/mt5-base>

Grattafiori, A., et al. (2024). The Llama 3 herd of models (arXiv:2407.21783). arXiv. <https://arxiv.org/abs/2407.21783>

Groq. (n.d.). Groq documentation: Fast AI inference. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://console.groq.com/docs>

Guo, S., Liao, L., Li, C., & Chua, T.-S. (2024). A survey on neural question generation: Methods, applications, and prospects. In K. Larson (Ed.), Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI-24) (pp. 8038–8047). International Joint Conferences on Artificial Intelligence. <https://doi.org/10.24963/ijcai.2024/889>

H5P Group. (n.d.-a). Fill in the Blanks. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://h5p.org/fill-in-the-blanks>

H5P Group. (n.d.-b). Package definition (h5p.json). Ανακτήθηκε στις 16 Μαΐου 2026, από <https://h5p.org/documentation/developers/json-file-definitions>

H5P Group. (n.d.-c). Single Choice Set. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://h5p.org/single-choice-set>

H5P Group. (n.d.-d). The .h5p specification. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://h5p.org/documentation/developers/h5p-specification>

Helsinki-NLP. (n.d.). OPUS-MT / MarianMT model pages. Hugging Face. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://huggingface.co/Helsinki-NLP>

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>

Hugging Face. (n.d.-a). Hugging Face Hub documentation. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://huggingface.co/docs/hub>

Hugging Face. (n.d.-b). Transformers documentation. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://huggingface.co/docs/transformers>

iarfmoose. (n.d.). iarfmoose/t5-base-question-generator [Model card]. Hugging Face. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://huggingface.co/iarfmoose/t5-base-question-generator>

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of hallucination in natural language generation. ACM Computing Surveys, 55(12), Article 248. <https://doi.org/10.1145/3571730>

Junczys-Dowmunt, M., Grundkiewicz, R., Dwojak, T., Hoang, H., Heafield, K., Neckermann, T., Seide, F., Hermann, U., Aji, A. F., Bogoychev, N., Martins, A. F. T., & Birch, A. (2018). Marian: Fast neural machine translation in C++. In Proceedings of ACL 2018, System Demonstrations (pp. 116–121). Association for Computational Linguistics. <https://doi.org/10.18653/v1/P18-4020>

Koutsikakis, J., Chalkidis, I., Malakasiotis, P., & Androutsopoulos, I. (2020). GREEK-BERT: The Greeks visiting Sesame Street. In 11th Hellenic Conference on Artificial Intelligence (SETN 2020) (pp. 110–117). Association for Computing Machinery. <https://doi.org/10.1145/3411408.3411440>

Kurdi, G., Leo, J., Parsia, B., Sattler, U., & Al-Emari, S. (2020). A systematic review of automatic question generation for educational purposes. International Journal of Artificial Intelligence in Education, 30(1), 121–204. <https://doi.org/10.1007/s40593-019-00186-y>

Le, N.-T., Kojiri, T., & Pinkwart, N. (2014). Automatic question generation for educational applications – The state of art. In T. van Do, H. A. L. Thi, & N. T. Nguyen (Eds.), Advanced Computational Methods for Knowledge Engineering (Advances in Intelligent Systems and Computing, Vol. 282, pp. 325–338). Springer. https://doi.org/10.1007/978-3-319-06569-4_24

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (pp. 7871–7880). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2020.acl-main.703>

Lin, C.-Y. (2004). ROUGE: A package for automatic evaluation of summaries. In Text Summarization Branches Out (pp. 74–81). Association for Computational Linguistics.

Lumi Education. (n.d.). H5P offline: Create content without Internet. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://lumi.education/>

Manning, C. D., & Schütze, H. (1999). Foundations of statistical natural language processing. MIT Press.

Miller, G. A. (1995). WordNet: A lexical database for English. Communications of the ACM, 38(11), 39–41. <https://doi.org/10.1145/219717.219748>

- Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., & Gebru, T. (2019). Model cards for model reporting. In Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT '19)* (pp. 220–229). Association for Computing Machinery. <https://doi.org/10.1145/3287560.3287596>
- Moodle. (n.d.-a). H5P. MoodleDocs. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://docs.moodle.org/en/H5P>
- Moodle. (n.d.-b). H5P activity. MoodleDocs. Ανακτήθηκε στις 16 Μαΐου 2026, από https://docs.moodle.org/en/H5P_activity
- Mulla, N., & Gharpure, P. (2023). Automatic question generation: A review of methodologies, datasets, evaluation metrics, and applications. *Progress in Artificial Intelligence*, 12(1), 1–32. <https://doi.org/10.1007/s13748-023-00295-9>
- Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajič, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., & Zeman, D. (2016). Universal Dependencies v1: A multilingual treebank collection. In Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016) (pp. 1659–1666). European Language Resources Association.
- Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). BLEU: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (pp. 311–318). Association for Computational Linguistics. <https://doi.org/10.3115/1073083.1073135>
- Qi, P., Zhang, Y., Zhang, Y., Bolton, J., & Manning, C. D. (2020). Stanza: A Python natural language processing toolkit for many human languages. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations (pp. 101–108). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2020.acl-demos.14>
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), 1–67.
- Stanford NLP Group. (n.d.). Stanza documentation: Tokenize, MWT, POS, lemma, depparse, NER. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://stanfordnlp.github.io/stanza/>

- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems* (Vol. 27, pp. 3104–3112). Curran Associates.
- Tiedemann, J., & Thottingal, S. (2020). OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation* (pp. 479–480). European Association for Machine Translation.
- Valhalla. (n.d.). valhalla/t5-base-qg-hl [Model card]. Hugging Face. Ανακτήθηκε στις 16 Μαΐου 2026, από <https://huggingface.co/valhalla/t5-base-qg-hl>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (Vol. 30, pp. 5998–6008). Curran Associates.
- Vedantam, R., Zitnick, C. L., & Parikh, D. (2015). CIDEr: Consensus-based image description evaluation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 4566–4575). IEEE.
- Wikimedia Foundation. (2024). MediaWiki API documentation. Ανακτήθηκε στις 16 Μαΐου 2026, από https://www.mediawiki.org/wiki/API:Main_page
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., ... Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 38–45). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2020.emnlp-demos.6>
- Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., & Raffel, C. (2021). mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 483–498). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2021.naacl-main.41>

Παράρτημα Α: «Κώδικας»

Στο παρόν παράρτημα παρατίθεται ορισμένα από τα βασικά αρχεία/modules κώδικα του συστήματος αυτόματης δημιουργίας ερωτήσεων που αναπτύχθηκε στο πλαίσιο της Διπλωματικής Εργασίας.

Αρχείο: `main.py`

```
import argparse
import sys
from pathlib import Path

if sys.stdout.encoding and sys.stdout.encoding.lower() != "utf-8":
    sys.stdout.reconfigure(encoding="utf-8")

from qq.config import MAX_QUESTIONS
from qq.file_io import load_input_file
from qq.pipeline import run_pipeline
from qq.h5p_export import (
    make_fill_in_content_json,
    make_single_choice_content_json,
    save_json,
    export_single_choice_h5p,
    export_fill_in_h5p,
)

def main():

    parser = argparse.ArgumentParser(
        description="AQG: Fill in the blanks + Single Choice (stanza +
T5) "
    )

    parser.add_argument(
        "input_path",
        help="Διαδρομή σε .txt ή .pdf αρχείο με εκπαιδευτικό κείμενο.",
    )

    parser.add_argument(
        "--run_name",
        default="run1",
        help="Όνομα υποφακέλου μέσα στο data/output/ για τα
αποτελέσματα.",
    )

    parser.add_argument(
        "--max_questions",
        type=int,
        default=None,
        help=f"Μέγιστος αριθμός ερωτήσεων ανά τύπο (default από config:
{MAX_QUESTIONS}).",
    )

    args = parser.parse_args()
```

```
if args.max_questions is not None and args.max_questions > 0:
    print(f"[INFO] MAX_QUESTIONS override από CLI:
{args.max_questions}")

print(f"[INFO] Φόρτωση αρχείου: {args.input_path}")
text = load_input_file(args.input_path)

input_lang, n_sentences, fill_in_items, single_choice_items,
_timings = run_pipeline(
    text=text,
    max_questions=args.max_questions,
    verbose=True,
)

print("\n" + "=" * 50)
print("[TIMING] Ανάλυση χρόνου ανά στάδιο:")
greek_mode = (input_lang == "el")
if greek_mode:
    print(f"  Μετάφραση el→en : {_timings.get('translate_in', 0)}s")
print(f"  stanza           : {_timings.get('stanza', 0)}s")
print(f"  T5 + Groq         : {_timings.get('generation', 0)}s")
if greek_mode:
    print(f"  Μετάφραση en→el : {_timings.get('translate_out',
0)}s")
print(f"  ΣΥΝΟΛΟ           : {_timings.get('total', 0)}s")
print("=" * 50)

print(f"\n[INFO] Σύνολο fill-in items: {len(fill_in_items)}")
print(f"[INFO] Σύνολο single-choice items:
{len(single_choice_items)}")

_SCRIPT_DIR = Path(__file__).resolve().parent
base_out_dir = _SCRIPT_DIR / "data" / "output" / args.run_name
base_out_dir.mkdir(parents=True, exist_ok=True)

input_stem = Path(args.input_path).stem

sc_h5p_path = base_out_dir / f"{input_stem}_single_choice.h5p"
fill_h5p_path = base_out_dir / f"{input_stem}_fill_in.h5p"

export_single_choice_h5p(
    single_choice_items,
    str(sc_h5p_path),
    title=f"{input_stem} - Single Choice Quiz",
)
export_fill_in_h5p(
    fill_in_items,
    str(fill_h5p_path),
    title=f"{input_stem} - Fill in the Blanks",
)

fill_in_content = make_fill_in_content_json(fill_in_items)
single_choice_content =
make_single_choice_content_json(single_choice_items)
```

```
save_json(fill_in_content, str(base_out_dir /
"fill_in_content.json"))
save_json(single_choice_content, str(base_out_dir /
"single_choice_content.json"))

system_questions = [item["question"] for item in
single_choice_items]

questions_txt_path = base_out_dir / "system_questions.txt"

with questions_txt_path.open("w", encoding="utf-8") as f:
    for q in system_questions:
        f.write(q.strip() + "\n")

print(f"[OK] Αποθηκεύτηκαν οι system questions στο:
{questions_txt_path}")
print("\n[DONE] Το AQG pipeline ολοκληρώθηκε. Δες τα αποτελέσματα
στο data/output/")

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

Αρχείο: pipeline.py

```
from typing import Optional, Tuple, List, Dict, Callable
import time
import random as _rnd

from .config import (
    MAX_SENTENCES as _CFG_MAX_SENTENCES,
    MAX_QUESTIONS as _CFG_MAX_QUESTIONS,
    USE_API as _CFG_USE_API,
    GROQ_API_KEY as _CFG_GROQ_API_KEY,
    GROQ_MODEL as _CFG_GROQ_MODEL,
    USE_TRANSLATION as _CFG_USE_TRANSLATION,
    NUM_DISTRACTORS as _CFG_NUM_DISTRACTORS,
)
from .nlp_pipelines import split_text_to_sentences, analyze_sentence,
generate_questions_batch
from .question_logic import build_fill_in_items, prepare_sc_inputs,
finalize_sc_item
from .api_postprocess import (
    postprocess_batch_fill_in,
    postprocess_batch_sc,
    postprocess_batch_greek_fill_in,
    postprocess_batch_greek_sc,
    _has_content_overlap,
)
from .translation import translate_to_english,
translate_questions_to_greek
from .dedup_utils import answer_already_seen, is_circular_question
from .lang_detect import detect_language

def run_pipeline(
    text: str,
```

```
max_questions: Optional[int] = None,
max_sentences: Optional[int] = None,
use_api: Optional[bool] = None,
groq_api_key: Optional[str] = None,
groq_model: Optional[str] = None,
use_translation: Optional[bool] = None,
verbose: bool = False,
log_fn: Optional[Callable[[str], None]] = None,
) -> Tuple[str, int, List[Dict], List[Dict], Dict[str, float]]:

    _cap = max_questions if max_questions else _CFG_MAX_QUESTIONS
    _max_sent = max_sentences if max_sentences else _CFG_MAX_SENTENCES
    _use_api = _CFG_USE_API if use_api is None else use_api
    _groq_key = _CFG_GROQ_API_KEY if groq_api_key is None else
groq_api_key
    _groq_model = _CFG_GROQ_MODEL if groq_model is None else groq_model
    _use_translation = _CFG_USE_TRANSLATION if use_translation is None
else use_translation

    def _log(msg: str) -> None:
        if log_fn is not None:
            log_fn(msg)
        elif verbose:
            print(msg)

    timings: Dict[str, float] = {}
    _t = time.time()

    input_lang = detect_language(text)
    _log(f"[INFO] Ανιχνεύθηκε γλώσσα: {input_lang}")

    greek_mode = (input_lang == "el" and _use_translation)

    anchor_dict: Dict[str, str] = {}
    original_greek_text = text
    if greek_mode:
        _log(f"[INFO] : Χτίσιμο anchor dictionary από πρωτότυπο
κείμενο...")
        try:
            from .anchor_dict import build_anchor_dict
            anchor_dict = build_anchor_dict(text)
            _log(f"[INFO] : Anchor dictionary με {len(anchor_dict)}
όρους.")
        except Exception as exc:
            _log(f"[WARN] : Αποτυχία χτισίματος anchor dict: {exc}")
            anchor_dict = {}

    if greek_mode:
        _log(f"[INFO] Ελληνικό κείμενο – μετάφραση el→en για το
pipeline...")
        text = translate_to_english(text)
        lang = "en"
        _log(f"[INFO] Μετάφραση el→en ολοκληρώθηκε.")

    try:
        from .anchor_dict import build_context_anchor_dict
```

```
ctx_dict = build_context_anchor_dict(original_greek_text,
english_text=text)

    for k, v in ctx_dict.items():
        if k not in anchor_dict:
            anchor_dict[k] = v
            _log(f"[INFO] : +{len(ctx_dict)} context anchors → σύνολο
{len(anchor_dict)}".)
        except Exception as exc:
            _log(f"[WARN] : Αποτυχία context anchor dict: {exc}")
    else:
        lang = input_lang
        timings["translate_in"] = round(time.time() - _t, 1) if greek_mode
    else 0.0
    _t = time.time()

sentences = split_text_to_sentences(text, lang)[:_max_sent]
n_sentences = len(sentences)
_log(f"[INFO] Βρέθηκαν {n_sentences} προτάσεις (max {_max_sent}).")

_BARRIER = {"text": "", "lemma": None, "upos": None, "xpos": None,
              "ner": "O", "start_char": None, "end_char": None}
global_token_pool: List[Dict] = []
for s in sentences:
    global_token_pool.extend(analyze_sentence(s, lang=lang))
    global_token_pool.append(dict(_BARRIER))
timings["stanza"] = round(time.time() - _t, 1)
_t = time.time()

fill_items: List[Dict] = []
sc_items: List[Dict] = []
seen_questions: set = set()
seen_fill_answers: set = set()
seen_sc_answers: set = set()

fill_raw: List[Dict] = []
seen_fill_raw: set = set()
_fill_buffer_cap = _cap * 3
fill_leftovers: List[Dict] = []
for sent in sentences:
    if len(fill_raw) >= _fill_buffer_cap:
        break
    items = build_fill_in_items(sent, lang=lang, k=2)
    first_taken = False
    for pos, fi in enumerate(items):
        ans_key = fi["answer"].strip().lower()
        if ans_key in seen_fill_raw:
            continue
        if not first_taken:
            seen_fill_raw.add(ans_key)
            fill_raw.append(fi)
            first_taken = True
            if len(fill_raw) >= _fill_buffer_cap:
                break
        else:
            fill_leftovers.append(fi)
```

```
for fi in fill_leftovers:
    if len(fill_raw) >= _fill_buffer_cap:
        break
    ans_key = fi["answer"].strip().lower()
    if ans_key in seen_fill_raw:
        continue
    seen_fill_raw.add(ans_key)
    fill_raw.append(fi)

sc_prepared: List[Dict] = []
_sc_buffer_cap = _cap * 3
sc_leftovers: List[Dict] = []
for idx, sent in enumerate(sentences):
    if len(sc_prepared) >= _sc_buffer_cap:
        break
    parts: List[str] = []
    if idx > 0:
        parts.append(sentences[idx - 1])
    parts.append(sent)
    if idx + 1 < len(sentences):
        parts.append(sentences[idx + 1])
    context = " ".join(parts)
    preps = prepare_sc_inputs(sent, lang=lang,
global_token_pool=global_token_pool, context=context, k=2)
    if preps:
        sc_prepared.append(preps[0])
        sc_leftovers.extend(preps[1:])

for prep in sc_leftovers:
    if len(sc_prepared) >= _sc_buffer_cap:
        break
    sc_prepared.append(prep)
    _log(f"[INFO] SC candidates: {len(sc_prepared)} (hybrid 1-per-
sentence + fallback).")

batch_pairs = [(p["t5_input"], p["correct_answer"]) for p in
sc_prepared]
batch_qs = generate_questions_batch(batch_pairs) if batch_pairs else
[]

results_by_idx: Dict[int, Optional[Dict]] = {}
for i, (prep, raw_q) in enumerate(zip(sc_prepared, batch_qs)):
    if not raw_q:
        continue
    try:
        results_by_idx[i] = finalize_sc_item(prep, raw_q)
    except Exception as exc:
        _log(f" [WARN] SC candidate {i+1}: {exc}")
        results_by_idx[i] = None

if _use_api and _groq_key:
    if not greek_mode:
        _log(f"[INFO] Batch Groq fill-in ({len(fill_raw)}
items)...")
        fi_groq = postprocess_batch_fill_in(fill_raw, _groq_key,
_groq_model)
```

```
for orig, improved in zip(fill_raw, fi_groq):
    if len(fill_items) >= _cap:
        break
    ans_key = orig["answer"].strip().lower()
    if answer_already_seen(ans_key, seen_fill_answers):
        continue
    seen_fill_answers.add(ans_key)
    if improved and "text_with_blank" in improved:
        fill_items.append({
            "type": "fill_in",
            "original_sentence": orig["original_sentence"],
            "answer": improved["answer"],
            "text_with_blank": improved["text_with_blank"],
        })
    else:
        fill_items.append(orig)
else:

    for fi in fill_raw:
        if len(fill_items) >= _cap:
            break
        ans_key = fi["answer"].strip().lower()
        if answer_already_seen(ans_key, seen_fill_answers):
            continue
        seen_fill_answers.add(ans_key)
        fill_items.append(fi)

    sc_batch_order = [(i, results_by_idx[i]) for i in
sorted(results_by_idx) if results_by_idx.get(i)]
    if sc_batch_order:
        _log(f"[INFO] Batch Groq SC ({len(sc_batch_order)}
items)...")
        sc_groq = postprocess_batch_sc(
            [item for _, item in sc_batch_order], _groq_key,
            _groq_model
        )
        import re as _re_local
        for j, (i, base) in enumerate(sc_batch_order):
            gr = sc_groq[j] if j < len(sc_groq) else None
            if gr and gr.get("rejected"):
                results_by_idx[i] = None
            elif gr and base:

                corrected_answer = gr.get("answer", "").strip()
                original_answer = base.get("correct_answer",
                "").strip()

                original_sentence = base.get("original_sentence",
                "")

                if corrected_answer and corrected_answer.lower() !=
original_answer.lower():
                    src_lower = original_sentence.lower()
                    new_lower = corrected_answer.lower()
                    has_in_source = bool(
                        _re_local.search(r"\b" +
_re_local.escape(new_lower) + r"\b", src_lower)
                    ) or (new_lower in src_lower and len(new_lower)
>= 4)

                    if has_in_source:
```

```

        base["correct_answer"] = corrected_answer
    else:
        _log(f" [SC guard] KEEP original:
'{corrected_answer}' not in source, keeping T5 '{original_answer}'")
        gr = {"question": base["question"],
"distractors": base.get("distractor_pool") or base.get("distractors",
[])}

        new_distractors = list(gr.get("distractors",
[]))[:3]
        pool = base.get("distractor_pool") or
base.get("distractors", [])
        if len(new_distractors) < 3:
            correct_lower =
base["correct_answer"].strip().lower()
            existing_lower = {d.strip().lower() for d in
new_distractors}

            for d in pool:
                dl = d.strip().lower()
                if dl and dl != correct_lower and dl not in
existing_lower:

                    new_distractors.append(d)
                    existing_lower.add(dl)
                    if len(new_distractors) >= 3:
                        break

            correct_lower =
base["correct_answer"].strip().lower()
            seen_opts = {correct_lower}
            unique_distractors = []
            for d in new_distractors:
                dl = d.strip().lower()
                if dl and dl not in seen_opts:
                    seen_opts.add(dl)
                    unique_distractors.append(d)
            options = [{"text": base["correct_answer"],
"correct": True}]
            for d in unique_distractors:
                options.append({"text": d, "correct": False})
            _rnd.shuffle(options)

            new_q = gr.get("question", base["question"])
            if new_q and new_q != base["question"]:
                if
_has_content_overlap(base.get("original_sentence", ""), new_q):
                    base["question"] = new_q
                else:
                    _log(f" [EN SC guard] KEEP original
question (no content overlap)")
                    base["options"] = options
                    results_by_idx[i] = base

            else:

                for fi in fill_raw:
                    ans_key = fi["answer"].strip().lower()
                    if not answer_already_seen(ans_key, seen_fill_answers):
                        seen_fill_answers.add(ans_key)
                        fill_items.append(fi)

```

```
        if len(fill_items) >= _cap:
            break

    for i in list(results_by_idx):
        base = results_by_idx.get(i)
        if not base:
            continue
        correct = base.get("correct_answer", "")
        correct_lower = correct.strip().lower()

        chosen = [o["text"] for o in base.get("options", []) if not
o.get("correct")]
        chosen_lower = {d.strip().lower() for d in chosen}

        for d in (base.get("distractor_pool") or []):
            if len(chosen) >= _CFG_NUM_DISTRACTORS:
                break
            dl = d.strip().lower()
            if dl and dl != correct_lower and dl not in
chosen_lower:
                chosen.append(d)
                chosen_lower.add(dl)

        options = [{"text": correct, "correct": True}]
        for d in chosen[:_CFG_NUM_DISTRACTORS]:
            options.append({"text": d, "correct": False})
        _rnd.shuffle(options)
        base["options"] = options

    for i in sorted(results_by_idx):
        if len(sc_items) >= _cap:
            break
        sc = results_by_idx[i]
        if not sc:
            continue
        if is_circular_question(sc["question"], sc.get("correct_answer",
"")):
            continue
        q_key = sc["question"].strip().lower()
        ans_key = sc.get("correct_answer", "").strip().lower()
        if q_key not in seen_questions and (not ans_key or not
answer_already_seen(ans_key, seen_sc_answers)):
            seen_questions.add(q_key)
            if ans_key:
                seen_sc_answers.add(ans_key)
            sc_items.append(sc)

    timings["generation"] = round(time.time() - _t, 1)
    _t = time.time()

    if greek_mode:

        fill_items = translate_questions_to_greek(fill_items, "fill_in",
anchor_dict=anchor_dict)
        sc_items = translate_questions_to_greek(sc_items,
"single_choice", anchor_dict=anchor_dict)
```

```

    if _use_api and _groq_key:
        _log("[INFO] Batch Groq GR fill-in review...")
        gr-fi = postprocess_batch_greek_fill_in(fill_items,
        _groq_key, _groq_model)
        for j, reviewed in enumerate(gr-fi):
            if reviewed and j < len(fill_items):
                new_text = reviewed["text_with_blank"]
                new_answer = reviewed["answer"]
                expected_blank = f"[{new_answer}]"
                if expected_blank not in new_text:
                    _log(f" [GR fill-in guard] SKIP item {j}: blank
not in reviewed text")
                    continue
                fill_items[j]["text_with_blank"] = new_text
                fill_items[j]["answer"] = new_answer

        _log("[INFO] Batch Groq GR SC review...")
        gr-sc = postprocess_batch_greek_sc(sc_items, _groq_key,
        _groq_model)
        for j, reviewed in enumerate(gr-sc):
            if reviewed and j < len(sc_items):
                sc_items[j]["question"] = reviewed.get("question",
sc_items[j]["question"])
                new_dist = list(reviewed.get("distractors", []))[:3]
                translated_correct = next(
                    (opt["text"] for opt in
sc_items[j].get("options", []) if opt.get("correct")),
                    sc_items[j]["correct_answer"],
                )
                correct_lower = (translated_correct or
"".strip().lower()
                if len(new_dist) < 3:
                    existing_lower = {d.strip().lower() for d in
new_dist}

                    for d in sc_items[j].get("distractor_pool", []):
                        dl = d.strip().lower()
                        if dl and dl != correct_lower and dl not in
existing_lower:

                            new_dist.append(d)
                            existing_lower.add(dl)
                            if len(new_dist) >= 3:
                                break
                    if len(new_dist) < 3:
                        for opt in sc_items[j].get("options", []):
                            if opt.get("correct"):
                                continue
                            dt = opt.get("text", "")
                            dl = dt.strip().lower()
                            if dl and dl != correct_lower and dl not
in existing_lower:

                                new_dist.append(dt)
                                existing_lower.add(dl)
                                if len(new_dist) >= 3:
                                    break
                seen_opts = {correct_lower}
                unique_dist = []
                for d in new_dist:
                    dl = d.strip().lower()
                    if dl and dl not in seen_opts:

```

```
        seen_opts.add(dl)
        unique_dist.append(d)
    if unique_dist:
        options = [{"text": translated_correct,
"correct": True}]
        for d in unique_dist:
            options.append({"text": d, "correct":
False})
        _rnd.shuffle(options)
        sc_items[j]["options"] = options

    seen_gr_q: set = set()
    sc_items_deduped = []
    for sc in sc_items:
        qk = sc.get("question", "").strip().lower()
        if qk in seen_gr_q:
            continue
        correct_opt = next((o["text"] for o in sc.get("options", []))
if o.get("correct")), "")
        if is_circular_question(sc.get("question", ""),
correct_opt):
            continue
        seen_gr_q.add(qk)
        sc_items_deduped.append(sc)
    sc_items = sc_items_deduped

    seen_gr_twbl: set = set()
    fill_items_deduped = []
    for fi in fill_items:
        twbl_k = fi.get("text_with_blank", "").strip().lower()
        if twbl_k not in seen_gr_twbl:
            seen_gr_twbl.add(twbl_k)
            fill_items_deduped.append(fi)
    fill_items = fill_items_deduped

    timings["translate_out"] = round(time.time() - _t, 1) if greek_mode
else 0.0
    timings["total"] = round(sum(timings.values()), 1)

    return input_lang, n_sentences, fill_items, sc_items, timings
```

Αρχείο: question_logic.py

```
from typing import List, Dict, Optional
import re
import random

from .nlp_pipeline import analyze_sentence
from .config import NUM_DISTRACTORS
from .distractor_sources import get_wordnet_distractors,
get_wikipedia_distractors

NUMBER_WORDS = {
    "zero", "one", "two", "three", "four", "five", "six", "seven",
    "eight",
```

```
"nine", "ten", "eleven", "twelve", "thirteen", "fourteen",  
"fifteen",  
"sixteen", "seventeen", "eighteen", "nineteen", "twenty", "thirty",  
"forty", "fifty", "sixty", "seventy", "eighty", "ninety",  
"hundred", "thousand", "million",  
}  
  
STOPWORDS_EN = {  
  
    "the", "a", "an", "this", "that", "these", "those", "one", "ones",  
  
    "he", "she", "it", "they", "his", "her", "their", "him", "them",  
    "its",  
    "we", "us", "our", "you", "your", "my", "me", "i",  
  
    "of", "to", "for", "in", "on", "at", "from", "by", "with", "into",  
    "onto", "upon", "over", "under", "about", "around", "through",  
    "between",  
    "among", "along", "against", "during", "within", "without",  
    "beyond",  
  
    "and", "or", "but", "as", "if", "so", "yet", "nor",  
  
    "is", "are", "was", "were", "be", "been", "being",  
    "have", "has", "had", "do", "does", "did", "will", "would",  
    "can", "could", "may", "might", "shall", "should", "must",  
  
    "who", "what", "where", "when", "which", "how",  
  
    "today", "now", "then", "there", "here", "also", "just", "only",  
    "still", "even", "ever", "never", "always", "often", "every",  
    "already", "soon", "again", "once", "twice", "both", "all",  
    "some", "such", "many", "much", "more", "most", "other", "same",  
    "well", "back", "away", "rather", "quite", "very", "too",  
  
    "city", "place", "area", "world", "form", "play", "work",  
    "location", "region", "country", "period", "part", "century",  
    "number", "type", "kind", "sort", "example", "use", "role",  
  
    "size", "function", "action", "side", "way", "thing", "fact",  
    "idea", "ideas", "story", "point", "case", "group", "level",  
    "process", "system", "result", "amount", "side", "step",  
}  
  
EPITHET_BLACKLIST = {  
    "great", "little", "elder", "younger",  
    "southeast", "northeast", "northwest", "southwest",  
    "eastern", "western", "northern", "southern", "central",  
    "empire",  
}  
  
def _normalize_text(text: Optional[str]) -> str:  
    if text is None:  
        return ""  
    return str(text).strip()
```

```
def _is_valid_answer_text(text: Optional[str], token_info:
Optional[Dict] = None) -> bool:
    t = _normalize_text(text)
    if not t:
        return False

    low = t.lower()

    if len(low) <= 2:
        return False

    if low in STOPWORDS_EN:
        return False

    if low in NUMBER_WORDS:
        return False

    if low in EPITHET_BLACKLIST:
        return False

    if re.fullmatch(r"\W+", t):
        return False

    _ner_base = ""
    if token_info is not None:
        _ner_base = (token_info.get("ner") or "").upper()
        for _pfx in ("S-", "B-", "I-", "E-"):
            if _ner_base.startswith(_pfx):
                _ner_base = _ner_base[2:]
                break

    is_date_ner = _ner_base in {"DATE", "TIME"}
    is_quantity_ner = _ner_base in {"QUANTITY", "PERCENT", "MONEY"}

    if not is_date_ner and not is_quantity_ner:
        if re.fullmatch(r"[\dA-Z]{1,4}(st|nd|rd|th)?", t):
            return False

        if
re.search(r"\b\d{2,4}\s*(μ\.X\.|π\.X\.|AD\.|BC\.|BCE\.)\b", t,
re.IGNORECASE):
            return False
        if re.fullmatch(r"(μ\.X\.|π\.X\.|AD\.|BC\.|BCE\.)", t,
re.IGNORECASE):
            return False
        if re.fullmatch(r"\d{3,4}\.?", t):
            return False
        elif is_date_ner:

            if re.fullmatch(r"(μ\.X\.|π\.X\.|AD\.|BC\.|BCE\.|CE\.)", t,
re.IGNORECASE):
                return False
            if re.fullmatch(r"\d{1,2}", t):
                return False
        elif is_quantity_ner:
```

```
if re.fullmatch(r"[\d\.\,\\s]+", t):
    return False

if token_info is not None and _ner_base not in {"NORP"}:
    upos = (token_info.get("upos") or "").upper()
    xpos = (token_info.get("xpos") or "").upper()
    if upos == "ADJ" or xpos in {"JJ", "JJR", "JJS"}:
        return False

return True

def _is_valid_distractor_text(text: Optional[str], correct_answer: str,
token_info: Optional[Dict] = None) -> bool:
    t = _normalize_text(text)

    if re.search(r"\s+['\"]s?\b", t) or t.endswith("'") or
t.endswith('"'):
        return False

    if not _is_valid_answer_text(t, token_info=token_info):
        return False

    if re.fullmatch(r"\d[\\d\\s]*", t):
        return False

    ans_norm = _normalize_text(correct_answer)
    if t.lower() == ans_norm.lower():
        return False

    t_lower = t.lower()
    ans_lower = ans_norm.lower()
    if ans_lower and ans_lower in t_lower.split():
        return False

    short, long_ = (ans_lower, t_lower) if len(ans_lower) <=
len(t_lower) else (t_lower, ans_lower)
    if len(short) >= 4 and long_.startswith(short) and len(long_) -
len(short) <= 2:
        return False

    if len(ans_lower.split()) > 1:
        ans_words = set(ans_lower.split())
        dist_words = set(t_lower.split())
        if dist_words and dist_words.issubset(ans_words):
            return False

    return True

def _make_span_dict(tokens: List[Dict], ner_type: str) -> Dict:
    text = " ".join(t.get("text", "") for t in tokens)
    return {
        "text": text,
        "ner": ner_type,
        "upos": tokens[0].get("upos"),
        "xpos": tokens[0].get("xpos"),
        "lemma": tokens[0].get("lemma"),
```

```
        "start_char": tokens[0].get("start_char"),
        "end_char": tokens[-1].get("end_char"),
    }

def _group_ner_spans(tokens_info: List[Dict]) -> List[Dict]:
    spans: List[Dict] = []
    current_tokens: List[Dict] = []
    current_type: Optional[str] = None

    for token in tokens_info:
        ner = (token.get("ner") or "O").upper()

        if ner == "O":
            if current_tokens:
                spans.append(_make_span_dict(current_tokens,
current_type))
                current_tokens = []
                current_type = None
            continue

        if len(ner) > 2 and ner[1] == "-":
            prefix, base_type = ner[0], ner[2:]
        else:
            prefix, base_type = "S", ner

        if prefix == "S":
            if current_tokens:
                spans.append(_make_span_dict(current_tokens,
current_type))
                current_tokens = []
                current_type = None
            spans.append(_make_span_dict([token], base_type))

        elif prefix == "B":
            if current_tokens:
                spans.append(_make_span_dict(current_tokens,
current_type))
                current_tokens = [token]
                current_type = base_type

        elif prefix in ("I", "E"):
            if current_tokens and current_type == base_type:
                current_tokens.append(token)
                if prefix == "E":
                    spans.append(_make_span_dict(current_tokens,
current_type))
                    current_tokens = []
                    current_type = None
            else:
                if current_tokens:
                    spans.append(_make_span_dict(current_tokens,
current_type))
```

```
current_tokens = [token]
current_type = base_type

if current_tokens:
    spans.append(_make_span_dict(current_tokens, current_type))

for span in spans:
    words = span["text"].split()
    n = len(words)
    if n >= 2 and n % 2 == 0:
        half = n // 2
        if words[:half] == words[half:]:
            span["text"] = " ".join(words[:half])

MERGE_TYPES = {"PERSON", "PER", "ORG", "ORGANIZATION",
               "LOC", "GPE", "FAC", "LOCATION",
               "EVENT",
               "NORP",
               "WORK_OF_ART"}

merged: List[Dict] = []
for span in spans:
    ner_type = (span.get("ner") or "").upper()
    _gap = (
        span["start_char"] - merged[-1]["end_char"]
        if (merged
            and span.get("start_char") is not None
            and merged[-1].get("end_char") is not None)
        else None
    )
    if (
        merged
        and merged[-1].get("ner") == span.get("ner")
        and ner_type in MERGE_TYPES
        and _gap is not None

        and 0 <= _gap <= 2
    ):

        prev_text = merged[-1]["text"].lower()
        new_text = span["text"].lower()
        would_be = prev_text + " " + new_text

        words = would_be.split()
        n = len(words)
        half = n // 2
        is_doubled = (n >= 2 and n % 2 == 0 and words[:half] ==
words[half:])

        is_prefix_overlap = prev_text.startswith(new_text) or
new_text.startswith(prev_text)

        if is_doubled or is_prefix_overlap:
            merged.append(span)
        else:
            merged[-1]["text"] = merged[-1]["text"] + " " +
span["text"]
```

```

        merged[-1]["end_char"] = span["end_char"]
    else:
        merged.append(span)

    return merged

def _find_named_entities(tokens_info: List[Dict]) -> List[Dict]:
    spans = _group_ner_spans(tokens_info)
    return [s for s in spans if _is_valid_answer_text(s.get("text"),
token_info=s)]

def _find_nouns(tokens_info: List[Dict]) -> List[Dict]:
    nouns: List[Dict] = []
    for t in tokens_info:
        upos = (t.get("upos") or "").upper()
        xpos = (t.get("xpos") or "").upper()
        is_noun = upos in {"NOUN", "PROPN"} or xpos.startswith("NN")
        if is_noun:
            nouns.append(t)
    return nouns

def _answer_span_key(token: Dict) -> tuple:
    start = token.get("start_char")
    end = token.get("end_char")
    if start is not None and end is not None:
        return ("span", start, end)
    return ("text", (token.get("text") or "").strip().lower())

def _spans_overlap(a: Dict, b: Dict) -> bool:
    a_start, a_end = a.get("start_char"), a.get("end_char")
    b_start, b_end = b.get("start_char"), b.get("end_char")
    if None in (a_start, a_end, b_start, b_end):
        ta = (a.get("text") or "").strip().lower()
        tb = (b.get("text") or "").strip().lower()
        return bool(ta) and bool(tb) and (ta == tb or ta in tb or tb in
ta)
    return not (a_end <= b_start or b_end <= a_start)

def select_answer_tokens(tokens_info: List[Dict], k: int = 3) ->
List[Dict]:
    named_entities = _find_named_entities(tokens_info)

    def _ner_base(t: Dict) -> str:
        n = (t.get("ner") or "").upper()
        for pfx in ("S-", "B-", "I-", "E-"):
            if n.startswith(pfx):
                return n[2:]
        return n

    _HIGH_VALUE = {
        "PERSON", "PER",
        "LOC", "GPE", "FAC", "LOCATION",
        "ORG", "ORGANIZATION",
        "EVENT",
        "NORP",
        "WORK_OF_ART",

```

```
}

tiers: List[List[Dict]] = [[], [], [], [], []]

for t in named_entities:
    base = _ner_base(t)
    if not _is_valid_answer_text(t.get("text"), token_info=t):
        continue
    if base in _HIGH_VALUE:
        tiers[0].append(t)
    elif base in {"DATE", "TIME"}:
        tiers[1].append(t)
    elif base in {"QUANTITY", "PERCENT", "MONEY"}:
        tiers[2].append(t)

for t in _find_nouns(tokens_info):
    if _is_valid_answer_text(t.get("text"), token_info=t):
        tiers[3].append(t)

for t in tokens_info:
    upos = (t.get("upos") or "").upper()
    xpos = (t.get("xpos") or "").upper()
    if upos in {"NOUN", "PROPN"} or xpos.startswith("NN"):
        if _is_valid_answer_text(t.get("text"), token_info=t):
            tiers[4].append(t)

selected: List[Dict] = []
seen_keys: set = set()

for tier in tiers:
    for t in tier:
        key = _answer_span_key(t)
        if key in seen_keys:
            continue

        if any(_spans_overlap(t, s) for s in selected):
            continue
        seen_keys.add(key)
        selected.append(t)
        if len(selected) >= k:
            return selected

return selected

def select_answer_token(tokens_info: List[Dict]) -> Optional[Dict]:
    results = select_answer_tokens(tokens_info, k=1)
    return results[0] if results else None

def _ner_semantic_group(ner: str) -> str:
    n = (ner or "O").upper()

    if len(n) > 2 and n[1] == "-":
        n = n[2:]

    if n in {"PERSON", "PER"}:
        return "person"
```

```
if n in {"LOC", "GPE", "LOCATION", "FAC"}:
    return "location"
if n in {"ORG", "ORGANIZATION"}:
    return "organization"
if n in {"DATE", "TIME"}:
    return "date"
if n in {"CARDINAL", "ORDINAL", "QUANTITY", "PERCENT", "MONEY"}:
    return "quantity"
if n in {"EVENT"}:
    return "event"
if n in {"NORP"}:
    return "norpp"
if n in {"WORK_OF_ART"}:
    return "work_of_art"
return "other"

def _get_distractor_candidates(
    answer_token: Dict,
    tokens_info: List[Dict],
    global_pool: Optional[List[Dict]] = None,
) -> List[str]:
    answer_text = _normalize_text(answer_token.get("text"))
    ans_ner = answer_token.get("ner") or "O"
    ans_pos = (
        (answer_token.get("upos") or answer_token.get("pos") or
         answer_token.get("xpos") or "")
        .upper()
    )

    def _collect_from_pool(pool: List[Dict], seen: set) -> tuple:
        same: List[str] = []
        other: List[str] = []

        if ans_ner != "O":
            target_group = _ner_semantic_group(ans_ner)
            for t in _find_named_entities(pool):
                txt = _normalize_text(t.get("text"))
                if not _is_valid_distractor_text(txt, answer_text,
                    token_info=t) or txt.lower() in seen:
                    continue

            if _ner_semantic_group(t.get("ner") or "O") !=
target_group:
                continue
            if (t.get("ner") or "O") == ans_ner:
                same.append(txt)
                seen.add(txt.lower())
            else:
                other.append(txt)
                seen.add(txt.lower())
        else:
            for t in _find_nouns(pool):
                txt = _normalize_text(t.get("text"))
                if not _is_valid_distractor_text(txt, answer_text,
                    token_info=t) or txt.lower() in seen:
```

```
        continue
    seen.add(txt.lower())
    t_pos = (
        (t.get("upos") or t.get("pos") or t.get("xpos") or
"" ).upper()
    )
    if ans_pos and t_pos == ans_pos:
        same.append(txt)
    else:
        other.append(txt)

    return same, other

seen_texts: set = set()
candidates: List[str] = []

same_local, _ = _collect_from_pool(tokens_info, seen_texts)
candidates.extend(same_local)

if ans_ner == "O":
    wn_results = get_wordnet_distractors(answer_text,
n=NUM_DISTRACTORS * 3)
    for w in wn_results:
        if w.lower() not in seen_texts and
_is_valid_distractor_text(w, answer_text):
            seen_texts.add(w.lower())
            candidates.append(w)

    _ans_ner_base = ans_ner.upper()
    for _pfx in ("S-", "B-", "I-", "E-"):
        if _ans_ner_base.startswith(_pfx):
            _ans_ner_base = _ans_ner_base[2:]
        break

    _ans_is_year = bool(re.search(r"\d", answer_text))
    if _ans_ner_base in {"DATE", "TIME"} and global_pool:
        for t in _find_named_entities(global_pool):
            ner_base = (t.get("ner") or "O").upper()
            for _pfx in ("S-", "B-", "I-", "E-"):
                if ner_base.startswith(_pfx):
                    ner_base = ner_base[2:]
                break
            if ner_base not in {"DATE", "TIME"}:
                continue
            txt = _normalize_text(t.get("text"))
            if not txt or txt.lower() == answer_text.lower() or
txt.lower() in seen_texts:
                continue
            if re.fullmatch(r"(μ\..X\.|π\..X\.|AD\..?|BC\..?|BCE\..?|CE\..?)",
txt, re.IGNORECASE):
                continue

            _txt_is_year = bool(re.search(r"\d", txt))
            if _ans_is_year != _txt_is_year:
                continue
            seen_texts.add(txt.lower())
            candidates.append(txt)
```

```
        if len(candidates) >= NUM_DISTRACTORS:
            break

    elif _ans_ner_base in {"QUANTITY", "PERCENT", "MONEY"} and
global_pool:
    for t in _find_named_entities(global_pool):
        ner_base = (t.get("ner") or "O").upper()
        for _pfx in ("S-", "B-", "I-", "E-"):
            if ner_base.startswith(_pfx):
                ner_base = ner_base[2:]
                break
        if ner_base not in {"QUANTITY", "PERCENT", "MONEY"}:
            continue
        txt = _normalize_text(t.get("text"))
        if not txt or txt.lower() == answer_text.lower() or
txt.lower() in seen_texts:
            continue

        if re.fullmatch(r"[\d\.\, \s]+", txt):
            continue
        seen_texts.add(txt.lower())
        candidates.append(txt)
        if len(candidates) >= NUM_DISTRACTORS:
            break

    if _ans_ner_base in {"DATE", "TIME"} and _ans_is_year and
len(candidates) < NUM_DISTRACTORS:
        m = re.search(r"(\d{1,4})", answer_text)
        if m:
            base_year = int(m.group(1))
            era_m = re.search(r"(μ\.X\.|π\.X\.|BCE|BC|AD|CE)",
answer_text, re.IGNORECASE)
            era = (" " + era_m.group(1)) if era_m else ""

            for off in (12, -23, 35, -8, 50, -41):
                cand_year = base_year + off
                if cand_year <= 0:
                    continue
                cand = f"{cand_year}{era}"
                if cand.lower() in seen_texts or cand.lower() ==
answer_text.lower():
                    continue
                seen_texts.add(cand.lower())
                candidates.append(cand)
                if len(candidates) >= NUM_DISTRACTORS:
                    break

    _NO_WIKI = {"DATE", "TIME", "QUANTITY", "PERCENT", "MONEY"}
    if ans_ner != "O" and _ans_ner_base not in _NO_WIKI:
        wiki_results = get_wikipedia_distractors(answer_text, ans_ner,
n=NUM_DISTRACTORS * 3)
        for w in wiki_results:
            if w.lower() not in seen_texts and
_is_valid_distractor_text(w, answer_text):
                seen_texts.add(w.lower())
                candidates.append(w)
```

```
if len(candidates) < NUM_DISTRACTORS and global_pool:
    same_g, other_g = _collect_from_pool(global_pool, seen_texts)
    candidates.extend(same_g)
    if len(candidates) < NUM_DISTRACTORS:
        candidates.extend(other_g)

if len(candidates) < NUM_DISTRACTORS and global_pool and ans_ner !=
"O":
    ans_subtype = _ans_ner_base
    for t in _find_named_entities(global_pool):
        txt = _normalize_text(t.get("text"))
        if not _is_valid_distractor_text(txt, answer_text,
token_info=t):
            continue
        if txt.lower() in seen_texts:
            continue
        t_sub = (t.get("ner") or "O").upper()
        for _pfx in ("S-", "B-", "I-", "E-"):
            if t_sub.startswith(_pfx):
                t_sub = t_sub[2:]
            break
        if t_sub != ans_subtype:
            continue
        seen_texts.add(txt.lower())
        candidates.append(txt)
        if len(candidates) >= NUM_DISTRACTORS:
            break

if len(candidates) < NUM_DISTRACTORS and global_pool:
    ans_group = _ner_semantic_group(ans_ner)
    for t in _find_named_entities(global_pool) +
_find_nouns(global_pool):
        txt = _normalize_text(t.get("text"))
        if not _is_valid_distractor_text(txt, answer_text,
token_info=t):
            continue
        if txt.lower() in seen_texts:
            continue
        t_group = _ner_semantic_group(t.get("ner") or "O")

        _strict_groups = {"date", "quantity", "event", "norp",
"work_of_art"}
        if ans_group in _strict_groups and t_group != ans_group:
            continue
        if ans_group not in _strict_groups and t_group in
_strict_groups:
            continue
        seen_texts.add(txt.lower())
        candidates.append(txt)
        if len(candidates) >= NUM_DISTRACTORS:
            break

return candidates

def _get_answer_type(answer_token: Dict) -> str:
    ner = (answer_token.get("ner") or "").upper()
    pos = (
```

```
(answer_token.get("upos") or answer_token.get("pos") or
answer_token.get("xpos") or "")
    .upper()
)

ner_base = ner
for _pfx in ("S-", "B-", "I-", "E-"):
    if ner_base.startswith(_pfx):
        ner_base = ner_base[2:]
        break

if ner_base in {"PERSON", "PER"}:
    return "person"
if ner_base in {"LOC", "LOCATION", "GPE", "FAC"}:
    return "location"
if ner_base in {"DATE", "TIME"}:
    return "date"
if ner_base in {"QUANTITY", "PERCENT", "MONEY", "CARDINAL"}:
    return "quantity"
if ner_base in {"EVENT"}:
    return "event"
if ner_base in {"NORP"}:
    return "norp"
if ner_base in {"WORK_OF_ART"}:
    return "work_of_art"
if ner_base in {"ORG", "ORGANIZATION"}:
    return "organization"

if pos in {"PROPN"}:
    return "person"
if pos in {"NUM"}:
    return "quantity"

return "other"

def postprocess_question(
    raw_question: str,
    answer_text: str,
    answer_token: Dict,
) -> str:
    q = _normalize_text(raw_question)
    a = _normalize_text(answer_text)

    if not q:
        q = ""

    q = q.replace("answer>", " ")
    q = q.replace("context>", " ")
    q = q.replace("<answer>", " ")
    q = q.replace("<context>", " ")

    q = re.sub(r"\s+", " ", q).strip()

    q = re.sub(r"\?+\s*\?+", "?", q)

    if q and not q.endswith("?"):
```

```
q = q.rstrip(".")
q = q.strip()
if q:
    q = q + "?"

wh_words = {"who", "what", "where", "when", "why", "how", "which"}
tokens = q.split() if q else []
has_wh = any(t.lower() in wh_words for t in tokens)

if len(tokens) >= 3 and has_wh:
    return q

answer_type = _get_answer_type(answer_token)

if answer_type == "person":
    return f"Who is {a}?"
if answer_type == "location":
    return f"Where is {a}?"
if answer_type == "date":
    return f"When did this happen regarding {a}?" if a else "When
did this happen?"
if answer_type == "quantity":
    return f"What is the measurement of {a}?" if a else "How much is
there?"
if answer_type == "event":
    return f"What was the {a}?"
if answer_type == "norp":
    return f"Which group was {a}?"
if answer_type == "work_of_art":
    return f"What is {a}?"
if answer_type == "organization":
    return f"What is {a}?"

return f"What is {a}?"

def _clean_fill_in_answer(answer_text: str) -> str:

    answer_text = re.sub(r"\s*['\"]s\b", "", answer_text).strip()
    answer_text = re.sub(r"\s*['\"]s*$", "", answer_text).strip()
    if not answer_text:
        return ""

    _leading_articles_en = {"the", "a", "an"}
    words = answer_text.split()
    while words and words[0].lower() in _leading_articles_en:
        words = words[1:]
    return " ".join(words) if words else ""

FILL_IN_MAX_ANSWER_WORDS = 2

_WEAK_FILL_IN_NOUNS = {
    "king", "queen", "leader", "ruler", "emperor", "philosopher",
    "teacher",
    "writer", "author", "poet", "general", "soldier", "scientist",
    "doctor",
    "president", "member", "group", "people", "man", "woman", "child",
```

```
}

def _make_fill_in_item(sentence: str, answer_token: Dict) ->
Optional[Dict]:
    answer_text = _normalize_text(answer_token.get("text"))
    if not answer_text:
        return None

    answer_text = _clean_fill_in_answer(answer_text)
    if not answer_text or len(answer_text) < 4:
        return None

    _ner = (answer_token.get("ner") or "O").upper()
    _is_entity = _ner != "O"
    if not _is_entity and answer_text.lower() in _WEAK_FILL_IN_NOUNS:
        return None

    if len(answer_text.split()) > FILL_IN_MAX_ANSWER_WORDS:
        return None

    count = sentence.count(answer_text)
    marker = f"[[{answer_text}]]"
    text_with_blank = sentence.replace(answer_text, marker, 1)
    if count > 1:

        head, sep, tail = text_with_blank.partition(marker)
        tail = tail.replace(answer_text, "...")
        text_with_blank = head + sep + tail

    return {
        "type": "fill_in",
        "original_sentence": sentence,
        "answer": answer_text,
        "text_with_blank": text_with_blank,
    }

def build_fill_in_items(sentence: str, lang: str = "en", k: int = 3) ->
List[Dict]:
    tokens_info = analyze_sentence(sentence, lang=lang)
    if not tokens_info:
        return []

    items: List[Dict] = []
    seen_answers: set = set()
    for answer_token in select_answer_tokens(tokens_info, k=max(k * 2, k
+ 2)):
        item = _make_fill_in_item(sentence, answer_token)
        if item is None:
            continue
        key = item["answer"].strip().lower()
        if key in seen_answers:
            continue
        seen_answers.add(key)
        items.append(item)
        if len(items) >= k:
            break
    return items
```

```
def _make_sc_input(
    sentence: str,
    answer_token: Dict,
    tokens_info: List[Dict],
    context: Optional[str],
    global_token_pool: Optional[List[Dict]],
) -> Optional[Dict]:
    correct_answer = _normalize_text(answer_token.get("text"))
    if not correct_answer:
        return None

    correct_answer = re.sub(r"\s+'s$", "", correct_answer).strip()
    correct_answer = re.sub(r"\s+'s*$", "", correct_answer).strip()
    if not correct_answer:
        return None

    return {
        "t5_input": context if context else sentence,
        "correct_answer": correct_answer,
        "answer_token": answer_token,
        "tokens_info": tokens_info,
        "sentence": sentence,
        "context": context,
        "global_token_pool": global_token_pool or [],
    }

def prepare_sc_inputs(
    sentence: str,
    lang: str = "en",
    global_token_pool: Optional[List[Dict]] = None,
    context: Optional[str] = None,
    k: int = 3,
) -> List[Dict]:
    if len(sentence.split()) < 6:
        return []

    tokens_info = analyze_sentence(sentence, lang=lang)
    if not tokens_info:
        return []

    results: List[Dict] = []
    seen_answers: set = set()
    for answer_token in select_answer_tokens(tokens_info, k=max(k * 2, k
+ 2)):
        prep = _make_sc_input(sentence, answer_token, tokens_info,
context, global_token_pool)
        if prep is None:
            continue
        key = prep["correct_answer"].strip().lower()
        if key in seen_answers:
            continue
        seen_answers.add(key)
        results.append(prep)
        if len(results) >= k:
            break
    return results
```

```
def finalize_sc_item(prepared: Dict, raw_question: str) ->
Optional[Dict]:
    sentence = prepared["sentence"]
    correct_answer = prepared["correct_answer"]
    answer_token = prepared["answer_token"]
    tokens_info = prepared["tokens_info"]
    global_token_pool = prepared["global_token_pool"]

    question_text = postprocess_question(raw_question, correct_answer,
answer_token)
    question_text = _normalize_text(question_text)
    if not question_text:
        return None

    candidates = _get_distractor_candidates(answer_token, tokens_info,
global_pool=global_token_pool)

    q_lower = question_text.lower()
    candidates = [c for c in candidates
                    if not re.search(r'\b' + re.escape(c.lower()) + r'\b',
q_lower)]

    def _near_dup(a: str, b: str) -> bool:
        a, b = a.strip().lower(), b.strip().lower()
        if a == b:
            return True
        short, long_ = (a, b) if len(a) <= len(b) else (b, a)
        return len(short) >= 4 and long_.startswith(short) and
len(long_) - len(short) <= 2

    deduped: List[str] = []
    for c in candidates:
        if any(_near_dup(c, d) for d in deduped):
            continue
        deduped.append(c)
    candidates = deduped

    if len(candidates) < NUM_DISTRACTORS:
        seen_lower = {c.strip().lower() for c in candidates}
        seen_lower.add(correct_answer.strip().lower())
        wn_extra = get_wordnet_distractors(correct_answer,
n=NUM_DISTRACTORS * 4)
        for w in wn_extra:
            wl = w.strip().lower()
            if wl in seen_lower:
                continue
            if not _is_valid_distractor_text(w, correct_answer):
                continue

            if re.search(r'\b' + re.escape(wl) + r'\b', q_lower):
                continue
            if any(_near_dup(w, d) for d in candidates):
                continue
            seen_lower.add(wl)
            candidates.append(w)
        if len(candidates) >= NUM_DISTRACTORS:
```

```
        break

distractors = candidates[:NUM_DISTRACTORS]

distractor_pool = candidates[:NUM_DISTRACTORS * 2]

if len(distractors) == 0:
    return None

options = [{"text": correct_answer, "correct": True}]
for d in distractors:
    options.append({"text": d, "correct": False})
random.shuffle(options)

return {
    "type": "single_choice",
    "original_sentence": sentence,
    "question": question_text,
    "correct_answer": correct_answer,
    "distractors": distractors,
    "distractor_pool": distractor_pool,
    "options": options,
}
```

Αρχείο: `distractor_sources.py`

```
from __future__ import annotations
import re
from typing import List, Optional

_BLACKLIST = {
    "potemkin", "potyokin", "grigori", "salutatorian", "salutatory",
    "dr.", "dr", "nr", "viz",
}

def _clean_wn_lemma(name: str) -> Optional[str]:

    name = name.replace("_", " ").strip()

    if not name or len(name) <= 2:
        return None

    if name.lower() in _BLACKLIST:
        return None

    if re.search(r"[0-9\(\)\[\]\@:/\#\&*\]", name):
        return None

    if len(name.split()) > 3:
        return None

    words = name.split()
    if len(words) > 1 and all(w[0].isupper() for w in words if w):
```

```
    return None

    return name

def get_wordnet_distractors(word: str, n: int = 8) -> List[str]:
    try:
        import nltk
        from nltk.corpus import wordnet as wn

        try:
            wn.synsets("test")
        except LookupError:
            nltk.download("wordnet", quiet=True)
            nltk.download("omw-1.4", quiet=True)

        word_lower = word.lower()
        seen: set = {word_lower}
        single_word: List[str] = []
        multi_word: List[str] = []

        synsets = wn.synsets(word_lower, pos=wn.NOUN)
        if not synsets:
            return []

        def _collect_hyponyms(synset) -> None:
            for hyponym in synset.hyponyms():
                for lemma in hyponym.lemmas():
                    cleaned = _clean_wn_lemma(lemma.name())
                    if not cleaned or cleaned.lower() in seen:
                        continue

                    if cleaned[0].isupper():
                        continue
                    seen.add(cleaned.lower())
                    if " " not in cleaned:
                        single_word.append(cleaned)
                    else:
                        multi_word.append(cleaned)

        for syn in synsets[:3]:
            for hypernym in syn.hypernyms()[1:2]:
                _collect_hyponyms(hypernym)
                _collect_hyponyms(syn)
            if len(single_word) >= n:
                break

        candidates = single_word + multi_word
        return candidates[:n]

    except Exception:
        return []

_PERSON_TYPES = {"PERSON", "PER"}
_LOCATION_TYPES = {"LOC", "GPE", "LOCATION", "FAC"}
_ORG_TYPES = {"ORG", "ORGANIZATION"}
```

```
def _ner_group(ner_type: str) -> str:
    ner = ner_type.upper()

    if len(ner) > 2 and ner[1] == "-":
        ner = ner[2:]
    if ner in _PERSON_TYPES:
        return "PERSON"
    if ner in _LOCATION_TYPES:
        return "LOCATION"
    if ner in _ORG_TYPES:
        return "ORG"
    return "OTHER"

def get_wikipedia_distractors(entity: str, ner_type: str, n: int = 8) ->
List[str]:
    target_group = _ner_group(ner_type)

    if target_group not in {"LOCATION", "ORG"}:
        return []

    try:
        import wikipedia
        wikipedia.set_lang("en")

        from .nlp_pipelines import analyze_sentence

        results = wikipedia.search(entity, results=3)
        if not results:
            return []

        ent_low = entity.lower()
        chosen_title = None
        for r in results:
            rl = r.lower()
            if ent_low in rl or rl in ent_low:
                chosen_title = r
                break
        if chosen_title is None:
            chosen_title = results[0]

        try:
            summary = wikipedia.summary(chosen_title, sentences=12,
auto_suggest=False)
        except Exception:
            return []

        tokens = analyze_sentence(summary, lang="en")
        seen: set = {entity.lower()}

        freq: dict = {}
        first_pos: dict = {}
        for pos, t in enumerate(tokens):
            ner = (t.get("ner") or "O").upper()
            text = (t.get("text") or "").strip()
            upos = (t.get("upos") or "").upper()
```

```
if not text or len(text) <= 2:
    continue
if text.lower() in seen:
    continue

if _ner_group(ner) != target_group:
    continue

if upos not in {"PROPN", "NOUN"}:
    continue

if text.isupper():
    continue

key = text.lower()
freq[key] = freq.get(key, 0) + 1
if key not in first_pos:
    first_pos[key] = pos

if not freq:
    return []

display: dict = {}
for t in tokens:
    text = (t.get("text") or "").strip()
    k = text.lower()
    if k in freq and k not in display:
        display[k] = text

ranked = sorted(freq.keys(), key=lambda k: (-freq[k],
first_pos[k]))
candidates = [display[k] for k in ranked if k in display]
return candidates[:n]

except Exception:
    return []
```

Αρχείο: translation.py

```
from __future__ import annotations
from functools import lru_cache
from typing import Optional
import re
from .config import DEVICE, USE_API, GROQ_API_KEY, GROQ_MODEL

_EL_EN_MODEL = "Helsinki-NLP/opus-mt-tc-big-el-en"
_EN_EL_MODEL = "Helsinki-NLP/opus-mt-en-el"
_CHUNK_SIZE = 450

@lru_cache(maxsize=1)
def _get_el_en():
    from transformers import MarianMTModel, MarianTokenizer
```

```

tok = MarianTokenizer.from_pretrained(_EL_EN_MODEL)
model = MarianMTModel.from_pretrained(_EL_EN_MODEL).to(DEVICE)
return tok, model

@lru_cache(maxsize=1)
def _get_en_el():
    from transformers import MarianMTModel, MarianTokenizer
    tok = MarianTokenizer.from_pretrained(_EN_EL_MODEL)
    model = MarianMTModel.from_pretrained(_EN_EL_MODEL).to(DEVICE)
    return tok, model

def _split_chunks(text: str, size: int = _CHUNK_SIZE) -> list[str]:
    if len(text) <= size:
        return [text]

    chunks = []
    while text:
        if len(text) <= size:
            chunks.append(text)
            break
        cut = text.rfind(".", 0, size)
        if cut == -1:
            cut = text.rfind("\n", 0, size)
        if cut == -1:
            cut = size
        chunks.append(text[:cut + 1])
        text = text[cut + 1:].lstrip()

    return chunks

def _clean(text: str) -> str:

    text = re.sub(r'City name \(optional,?\s*probably does not need a
translation\) ', '', text)

    text = re.sub(r'\s+', ' ', text).strip()
    return text

_CONTEXTUAL_FIXES = [

    (r"^\s*(το|τα)?\s*κεφάλαι[οα]\s*$", "πρωτεύουσα", "πρωτεύουσα"),

    (r"^\s*Ρόμαν\s*$", "Ρωμαϊκ", "Ρωμαίος"),

    (r"^\s*(το)?\s*δικαστήριο\s*$", "αυλή", "αυλή"),

]

_ACCEPTABLE_LATIN = {
    "dna", "rna", "atp", "adp", "mrna", "trna", "ph", "co2", "o2",
    "corpus", "juris", "civilis",
}

def _is_untranslated_latin(text: str) -> bool:
    if not text:
        return False

```

```
latin_words = [w for w in re.findall(r"[A-Za-z]{2,}", text)
                if w.lower() not in _ACCEPTABLE_LATIN]
if not latin_words:
    return False
greek_chars = len(re.findall(r"[Α-Ωα-ωΑΕΗΙΟΥΩϊϋϋάέήίούώ]", text))
latin_chars = sum(len(w) for w in latin_words)

return latin_chars >= max(greek_chars, 3)

def _fix_contextual_mistranslation(option_text: str, question_text: str)
-> str:
    q_lower = (question_text or "").lower()
    for pattern, clue, correct in _CONTEXTUAL_FIXES:
        if re.fullmatch(pattern, option_text, re.IGNORECASE) and
clue.lower() in q_lower:
            return correct
    return option_text

_KNOWN_TERMS_EN_EL = [
    ("Edict of Milan", "Διάταγμα των Μεδιολάνων"),
    ("Mediolanum", "Μεδιόλανα"),
    ("Corpus Juris Civilis", "Corpus Juris Civilis"),
    ("Hagia Sophia", "Αγία Σοφία"),
    ("Hokkaido", "Χοκάιντο"),
    ("Honshu", "Χονσού"),
    ("Kyushu", "Κιούσου"),
    ("Shikoku", "Σικόκου"),
]

def _apply_known_terms(text: str) -> str:
    for en, el in sorted(_KNOWN_TERMS_EN_EL, key=lambda kv: len(kv[0]),
reverse=True):
        text = re.sub(r"(?!\\w)" + re.escape(en) + r"(?!\\w)", el, text,
flags=re.IGNORECASE)
    return text

def _preprocess_for_greek(text: str) -> str:

    text = _apply_known_terms(text)

    roman = {"I": "Α'", "II": "Β'", "III": "Γ'", "IV": "Δ'", "V": "Ε'",
            "VI": "ΣΤ'", "VII": "Ζ'", "VIII": "Η'", "IX": "Θ'", "X":
            "Ι'",
            "XI": "ΙΑ'", "XII": "ΙΒ'"}
    for roman_num, greek_num in roman.items():

        text = re.sub(rf'\b([A-Z][a-z]+)\s+{roman_num}\b', rf'\1
{greek_num}', text)

        text = re.sub(r'\bAD\.\?\b', 'μ.Χ.', text)
        text = re.sub(r'\bBC\.\?\b', 'π.Χ.', text)

    return text

def _translate(text: str, tok, model) -> str:
    import torch
```

```
inputs = tok(text, return_tensors="pt", padding=True,
truncation=True, max_length=512)
inputs = {k: v.to(DEVICE) for k, v in inputs.items()}
with torch.no_grad():
    translated = model.generate(**inputs)
    return _clean(tok.decode(translated[0], skip_special_tokens=True))

def translate_to_english(text: str) -> str:
    tok, model = _get_el_en()
    chunks = _split_chunks(text)
    return " ".join(_translate(chunk, tok, model) for chunk in chunks)

def translate_to_greek(text: str) -> str:
    tok, model = _get_en_el()
    text = _preprocess_for_greek(text)
    chunks = _split_chunks(text)
    return " ".join(_translate(chunk, tok, model) for chunk in chunks)

def translate_questions_to_greek(
    questions: list[dict],
    q_type: str,
    anchor_dict: Optional[dict] = None,
) -> list[dict]:

    if anchor_dict:
        try:
            from .anchor_dict import apply_anchor_dict as _apply_anchor
        except Exception:
            _apply_anchor = None
            anchor_dict = None
    else:
        _apply_anchor = None

    def _maybe_anchor(text: str) -> str:
        if _apply_anchor and anchor_dict and text:
            return _apply_anchor(text, anchor_dict)
        return text

    _EN_COMMON = {
        "date", "mission", "release", "city", "state", "empire",
        "church",
        "north", "south", "east", "west", "year", "time", "name",
        "part",
        "system", "world", "history", "people", "place", "area",
        "region",
        "language", "religion", "century", "period", "group", "number",
    }
    _pre_anchor_dict = {}
    if anchor_dict:
        for k, v in anchor_dict.items():
            if not v or not k:
                continue
            first = v.strip()[0] if v.strip() else ""
            is_propn = ("A" <= first <= "Q") or first in "ΑΕΗΙΟΥΩΪΨ"
            if is_propn and len(k) >= 5 and k.lower() not in _EN_COMMON:
                _pre_anchor_dict[k] = v
```

```
def _pre_anchor_en(text: str) -> str:
    if _apply_anchor and _pre_anchor_dict and text:
        return _apply_anchor(text, _pre_anchor_dict)
    return text

translated = []

for item in questions:
    new_item = dict(item)

    if q_type == "fill_in":

        original = item.get("original_sentence") or
item.get("text_with_blank", "")

        original_clean = re.sub(r"\[\\[(.+?)\\]\\]", r"\1", original)

        translated_sentence =
translate_to_greek(_pre_anchor_en(original_clean))
        translated_answer =
translate_to_greek(_pre_anchor_en(item["answer"])).strip()

        translated_sentence = _maybe_anchor(translated_sentence)
        translated_answer =
_maybe_anchor(translated_answer).strip()

        text_with_blank = None

        if translated_answer and translated_answer in
translated_sentence:

            pos = translated_sentence.find(translated_answer)
            before_ok = (pos == 0 or not translated_sentence[pos -
1].isalpha())
            if before_ok:
                text_with_blank = (
                    translated_sentence[:pos]
                    + f"[{translated_answer}]"
                    + translated_sentence[pos +
len(translated_answer):]
                )
            else:
                text_with_blank = None
            elif len(translated_answer) >= 4:
                prefix = translated_answer[:4]

                m = re.search(r"(?!\\w)" + re.escape(prefix) + r"[α-ώά-
ώΑ-Ωά-ώα-zA-Z]*", translated_sentence)
                if m:
                    found_form = m.group(0)
                    text_with_blank = translated_sentence.replace(
                        found_form, f"[{found_form}]", 1
                    )
                    translated_answer = found_form

        if text_with_blank is None:
```

```
        text_with_blank =
translate_to_greek(item["text_with_blank"])

        new_item["text_with_blank"] = text_with_blank
        new_item["answer"] = translated_answer

    elif q_type == "single_choice":

        gr_question =
translate_to_greek(_pre_anchor_en(item["question"]))

        gr_question = _maybe_anchor(gr_question)
        new_options = []
        for opt in item["options"]:
            new_opt = dict(opt)
            raw_translated =
translate_to_greek(_pre_anchor_en(opt["text"]))

            raw_translated = _maybe_anchor(raw_translated)

            new_opt["text"] =
_fix_contextual_mistranslation(raw_translated, gr_question)

            if not new_opt.get("correct") and
_is_untranslated_latin(new_opt["text"]):
                continue
            new_options.append(new_opt)

        translated_pool: list = []
        seen_pool: set = set()
        for d in item.get("distractor_pool", []):
            td_raw = translate_to_greek(_pre_anchor_en(d))

            td_raw = _maybe_anchor(td_raw)
            td = _fix_contextual_mistranslation(td_raw, gr_question)
            if not td:
                continue

            if _is_untranslated_latin(td):
                continue
            tdl = td.strip().lower()
            if tdl and tdl not in seen_pool:
                seen_pool.add(tdl)
                translated_pool.append(td)

        new_item["question"] = gr_question
        new_item["options"] = new_options
        new_item["distractor_pool"] = translated_pool

    translated.append(new_item)

return translated
```

Αρχείο: anchor_dict.py

```
from __future__ import annotations
import re
from typing import Dict, List, Optional

_MIN_GREEK_LEN = 5
_MIN_PROPN_LEN = 3

_GREEK_FUNCTION_WORDS = {

    "τότε", "τώρα", "πάντα", "ποτέ", "συχνά", "σπάνια",

    "αυτός", "αυτή", "αυτό", "αυτοί", "αυτές", "αυτά",
    "εκείνος", "εκείνη", "εκείνο",

    "μέρος", "πράγμα", "τρόπος", "φορά", "είδος",
    "σημείο", "θέμα", "λόγος", "χρόνος",
}

_HIGH_VALUE_NER = {
    "PERSON", "PER",
    "LOC", "GPE", "FAC", "LOCATION",
    "ORG", "ORGANIZATION",
    "EVENT",
    "NORP",
    "WORK_OF_ART",
    "DATE", "TIME",
    "QUANTITY", "PERCENT", "MONEY",
    "PRODUCT",
}

_DATE_REGEX = re.compile(
    r"\b\d{1,4}\s*(?:μ\.?X\.?|π\.?X\.?|μX|πX|MX|ΠX)?",
    re.UNICODE,
)

_NAME_ORDINAL_REGEX = re.compile(
    r"[Α-ΩΑΕΗΤΟΥΩ] [α-ωάέήίόύώϊϋϗ] + (?:\s+ [Α-ΩΑΕΗΤΟΥΩ] [α-ωάέήίόύώϊϋϗ] *) * \s+ [ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ] {1,3} [' ']",
    re.UNICODE,
)

def _is_greek_capitalized(word: str) -> bool:
    if not word:
        return False
    first = word[0]
    return ("Α" <= first <= "Ω") or first in "ΑΕΗΤΟΥΩΪΥ"

def _extract_capitalized_phrases(sentence: str) -> List[str]:
    if not sentence:
        return []
```

```
words = sentence.split()
if len(words) <= 1:
    return []

phrases: List[str] = []

i = 1
while i < len(words):
    w = words[i].strip(".,;:!?\"'() ")
    if _is_greek_capitalized(w) and len(w) >= 3:

        group = [w]
        j = i + 1
        while j < len(words):
            nw = words[j].strip(".,;:!?\"'() ")

            if _is_greek_capitalized(nw) and len(nw) >= 2:
                group.append(nw)
                j += 1
            elif
re.fullmatch(r"[ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ]{1,3}['`']?", nw):

                group.append(nw)
                j += 1
            else:
                break
        phrase = " ".join(group)

        if phrase and len(phrase) >= _MIN_PROPN_LEN:
            phrases.append(phrase)
            i = j
        else:
            i += 1
    return phrases

def _extract_dates(text: str) -> List[str]:
    matches = _DATE_REGEX.findall(text)

    return [m.strip() for m in matches if m and m.strip() and
any(c.isdigit() for c in m)]

def _extract_greek_anchors(greek_text: str) -> List[str]:

    try:
        from .nlp_pipelines import analyze_sentence,
split_text_to_sentences
    except Exception:
        return []

    if not greek_text or not isinstance(greek_text, str):
        return []

    seen_lower: set = set()
    anchors: List[str] = []
```

```
def _add(term: str) -> None:
    if not term:
        return

    first_word = term.split()[0] if term.split() else term
    min_len = _MIN_PROPN_LEN if _is_greek_capitalized(first_word)
else _MIN_GREEK_LEN
    if len(term) < min_len:
        return
    key = term.lower()
    if key in seen_lower or key in _GREEK_FUNCTION_WORDS:
        return
    seen_lower.add(key)
    anchors.append(term)

try:

    for line in greek_text.splitlines():
        line = line.strip()
        if not line:
            continue
        raw_sentences = re.split(r"(?<=[.!?])\s+", line)
        for sent in raw_sentences:
            sent = sent.strip()
            if not sent:
                continue
            for phrase in _extract_capitalized_phrases(sent):
                _add(phrase)
    except Exception:
        pass

try:
    for date_str in _extract_dates(greek_text):
        _add(date_str)
    except Exception:
        pass

try:
    for word_match in re.finditer(r"\b[α-ωάέήίόύώϊϋϗ]{6,}\b",
greek_text):
        word = word_match.group(0)
        _add(word)
    except Exception:
        pass

try:
    sentences = split_text_to_sentences(greek_text, lang="el")
    except Exception:
        sentences = []

for sentence in sentences:
    try:
        tokens = analyze_sentence(sentence, lang="el")
    except Exception:
        continue
```

```
i = 0
while i < len(tokens):
    tok = tokens[i]
    ner_raw = (tok.get("ner") or "O").upper()
    ner_base = ner_raw
    for pfx in ("S-", "B-", "I-", "E-"):
        if ner_base.startswith(pfx):
            ner_base = ner_base[2:]
            break

    if ner_base in _HIGH_VALUE_NER:
        group_texts = [tok.get("text", "")]
        j = i + 1
        while j < len(tokens):
            next_ner = (tokens[j].get("ner") or "O").upper()
            for pfx in ("S-", "B-", "I-", "E-"):
                if next_ner.startswith(pfx):
                    next_ner = next_ner[2:]
                    break
            if next_ner == ner_base and next_ner != "O":
                group_texts.append(tokens[j].get("text", ""))
                j += 1
            else:
                break

        full_text = " ".join(t for t in group_texts if
t).strip()
        _add(full_text)
        i = j
        continue

    upos = (tok.get("upos") or "").upper()
    text = (tok.get("text") or "").strip()
    if upos in ("NOUN", "PROPN") and len(text) >=
_MIN_GREEK_LEN:
        _add(text)
        i += 1

return anchors

def build_anchor_dict(greek_text: str, max_anchors: int = 200) ->
Dict[str, str]:
    if not greek_text or not isinstance(greek_text, str):
        return {}

    anchors = _extract_greek_anchors(greek_text)
    if not anchors:
        return {}

    if len(anchors) > max_anchors:
        anchors = anchors[:max_anchors]

    try:
        from .translation import translate_to_english,
translate_to_greek
    except Exception:
        return {}
```

```
anchor_dict: Dict[str, str] = {}

original_greek_lowercase = {a.lower() for a in anchors}

for greek_term in anchors:

    try:
        english = translate_to_english(greek_term).strip()
    except Exception:
        continue
    if not english:
        continue

    english = english.strip().strip(".,;:!?")
    if len(english) < 2:
        continue
    if "(" in english or ")" in english:
        english = re.sub(r"\(.*?\)", "", english).strip()
        if not english:
            continue

    english_lower = english.lower()

    if english_lower not in anchor_dict and english_lower !=
greek_term.lower():
        anchor_dict[english_lower] = greek_term

    try:
        roundtrip_gr = translate_to_greek(english).strip()
    except Exception:
        roundtrip_gr = ""

    if not roundtrip_gr:
        continue
    roundtrip_gr = roundtrip_gr.strip().strip(".,;:!?")
    if "(" in roundtrip_gr or ")" in roundtrip_gr:
        roundtrip_gr = re.sub(r"\(.*?\)", "", roundtrip_gr).strip()

    if not roundtrip_gr:
        continue

    roundtrip_lower = roundtrip_gr.lower()
    original_lower = greek_term.lower()

    if roundtrip_lower == original_lower:
        continue

    if roundtrip_lower in original_greek_lowercase and
roundtrip_lower != original_lower:
        continue

    if roundtrip_lower in _GREEK_FUNCTION_WORDS:
        continue
```

```
if len(roundtrip_gr) < _MIN_PROPN_LEN:
    continue

if roundtrip_lower not in anchor_dict:
    anchor_dict[roundtrip_lower] = greek_term

return anchor_dict

def build_context_anchor_dict(
    greek_text: str,
    english_text: Optional[str] = None,
) -> Dict[str, str]:
    if not greek_text or not isinstance(greek_text, str):
        return {}

    try:
        from .translation import translate_to_english
    except Exception:
        return {}

    anchors = _extract_greek_anchors(greek_text)
    if not anchors:
        return {}

    propn_anchors = [a for a in anchors if
        _is_greek_capitalized(a.split()[0] if a.split() else a)]

    en_text_lower = (english_text or "").lower()
    result: Dict[str, str] = {}

    for anchor in propn_anchors:
        try:
            en = translate_to_english(anchor).strip().strip(".,;:!?")
        except Exception:
            continue
        if not en or len(en) < 3:
            continue
        if "(" in en or ")" in en:
            en = re.sub(r"\(.*?\)", "", en).strip()
            if not en:
                continue

        en_phrase = en.lower().strip()

        en_phrase_clean = re.sub(r"^(the|a|an)\s+", "", en_phrase)
        if en_text_lower and len(en_phrase_clean) >= 4 and
            en_phrase_clean in en_text_lower:
            if en_phrase_clean not in result:
                result[en_phrase_clean] = anchor

        en_words = [w for w in re.findall(r"[A-Za-z]+", en)
            if len(w) > 3 and w.lower() not in {"the", "and",
            "from", "with", "this", "that", "for"}]
        for ew in en_words:
            ewl = ew.lower()
            if en_text_lower and ewl not in en_text_lower:
```

```
        continue
    if ewl in result:
        continue
    result[ewl] = anchor

    return result

def apply_anchor_dict(text: str, anchor_dict: Optional[Dict[str, str]])
-> str:
    if not text or not anchor_dict:
        return text

    result = text

    sorted_keys = sorted(anchor_dict.keys(), key=len, reverse=True)

    for english_lower in sorted_keys:
        greek_original = anchor_dict[english_lower]
        if not english_lower or not greek_original:
            continue

        pattern = r"(?!\\w)" + re.escape(english_lower) + r"(?!\\w)"
        try:
            result = re.sub(pattern, greek_original, result,
flags=re.IGNORECASE)
        except re.error:
            continue

    return result
```

Αρχείο: h5p_export.py

```
import json
import re
import uuid
import zipfile
from pathlib import Path
from typing import List, Dict

def _new_id() -> str:
    return str(uuid.uuid4())

def _h5p_meta(main_library: str, major: int, minor: int, title: str) ->
Dict:
    return {
        "title": title,
        "language": "en",
        "mainLibrary": main_library,
        "license": "U",
        "embedTypes": ["iframe"],
        "preloadedDependencies": [
            {
                "machineName": main_library,
                "majorVersion": str(major),
```

```
        "minorVersion": str(minor),
    },
],
}

def _write_h5p_zip(out_path: str, h5p_meta: Dict, content: Dict) ->
None:
    path = Path(out_path)

    path.parent.mkdir(parents=True, exist_ok=True)

    with zipfile.ZipFile(path, "w", zipfile.ZIP_DEFLATED) as zf:

        zf.writestr(
            "h5p.json",
            json.dumps(h5p_meta, ensure_ascii=False, indent=2),
        )

        zf.writestr(
            "content/content.json",
            json.dumps(content, ensure_ascii=False, indent=2),
        )

    print(f"[OK] Αποθηκεύτηκε .h5p στο: {path}")

def export_single_choice_h5p(
    sc_items: List[Dict],
    out_path: str,
    title: str = "Single Choice Quiz",
) -> None:
    choices = []

    for item in sc_items:

        correct = [o["text"] for o in item["options"] if
o["correct"]]
        wrong = [o["text"] for o in item["options"] if not
o["correct"]]

        answers = [f"<p>{a}</p>" for a in correct + wrong]

        choices.append(
            {
                "subContentId": _new_id(),
                "question": f"<p>{item['question']}</p>",
                "answers": answers,
            }
        )

    content = {
        "choices": choices,
        "behaviour": {
            "autoContinue": True,
            "timeoutCorrect": 2000,
            "timeoutWrong": 3000,
            "soundEffectsEnabled": True,
```

```
        "enableRetry": True,
        "enableSolutionsButton": True,
        "passPercentage": 100,
    },
    "l10n": {
        "nextButtonLabel": "Next question",
        "showSolutionButtonLabel": "Show solution",
        "retryButtonLabel": "Retry",
        "solutionViewTitle": "Solution list",
        "correctText": "Correct!",
        "incorrectText": "Incorrect!",
        "muteButtonLabel": "Mute sound effects",
        "closeButtonLabel": "Close",
        "slideOfTotal": "Slide :num of :total",
        "scoreBarLabel": "You got :num out of :total points",
        "solutionListTitle": "Correct answers",
    },
}

_write_h5p_zip(
    out_path,
    _h5p_meta("H5P.SingleChoiceSet", 1, 11, title),
    content,
)

def export_fill_in_h5p(
    fill_items: List[Dict],
    out_path: str,
    title: str = "Fill in the Blanks",
) -> None:
    questions = []

    for item in fill_items:
        answer = item["answer"]

        sentence = _fix_brackets(item.get("text_with_blank") or
item.get("original_sentence", ""))

        q_text = re.sub(r'\[([.+?])\]', r'*\1*', sentence)

        if '*' not in q_text:
            q_text = sentence.replace(answer, f"*{answer}*", 1)

        questions.append(f"<p>{q_text}</p>")

    content = {
        "media": {"disableImageZooming": False},
        "text": f"<p>{title}</p>",
        "questions": questions,
        "behaviour": {
            "enableRetry": True,
            "enableSolutionsButton": True,
            "enableCheckButton": True,
            "autoCheck": False,
            "caseSensitive": False,
            "showSolutionsRequiresInput": True,
            "separateLines": False,
```

```
    },
    "l10n": {
        "checkAnswerButton": "Check",
        "submitAnswerButton": "Submit",
        "tryAgainButton": "Retry",
        "showSolutionButton": "Show solution",
        "notFilledOut": "Please fill in all blanks",
        "answerIsCorrect": "':ans' is correct",
        "answerIsWrong": "':ans' is wrong",
        "answeredCorrectly": "Answered correctly",
        "answeredIncorrectly": "Answered incorrectly",
        "solutionLabel": "Correct answer:",
        "inputLabel": "Blank input @num of @total",
        "inputHasTipLabel": "Tip available",
        "tipLabel": "Tip",
    },
}

_write_h5p_zip(
    out_path,
    _h5p_meta("H5P.Blanks", 1, 14, title),
    content,
)

def _fix_brackets(text: str) -> str:
    import re

    text = re.sub(r'\[([^\[]+)\](?!\\)', r'[[1]]', text)

    text = re.sub(r'(?![\[]\[([^\[]+)\]\]', r'[[1]]', text)

    text = re.sub(r'\[\\s*\]', '', text)
    return text

def make_fill_in_content_json(fill_in_items: List[Dict]) -> Dict:

    texts = [_fix_brackets(item["text_with_blank"]) for item in
fill_in_items]
    return {
        "text": "\n\n".join(texts),
        "behaviour": {},
        "ui": {},
        "media": {},
    }

def make_single_choice_content_json(single_choice_items: List[Dict]) ->
Dict:
    questions_for_h5p = []
    for item in single_choice_items:
        questions_for_h5p.append(
            {
                "question": item["question"],
                "answers": [
                    {"text": o["text"], "correct": o["correct"]}
                    for o in item["options"]
                ],
                "behaviour": {},
            }
        )
```

```
        "l10n": {},  
    }  
)  
return {  
    "questions": questions_for_h5p,  
    "behaviour": {},  
    "l10n": {},  
    "media": {},  
}  
  
def save_json(content: Dict, out_path: str) -> None:  
    path = Path(out_path)  
    path.parent.mkdir(parents=True, exist_ok=True)  
    path.write_text(  
        json.dumps(content, ensure_ascii=False, indent=2),  
        encoding="utf-8",  
    )  
    print(f"[OK] Αποθηκεύτηκε JSON στο: {path}")
```

Υπεύθυνη Δήλωση Συγγραφέα:

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