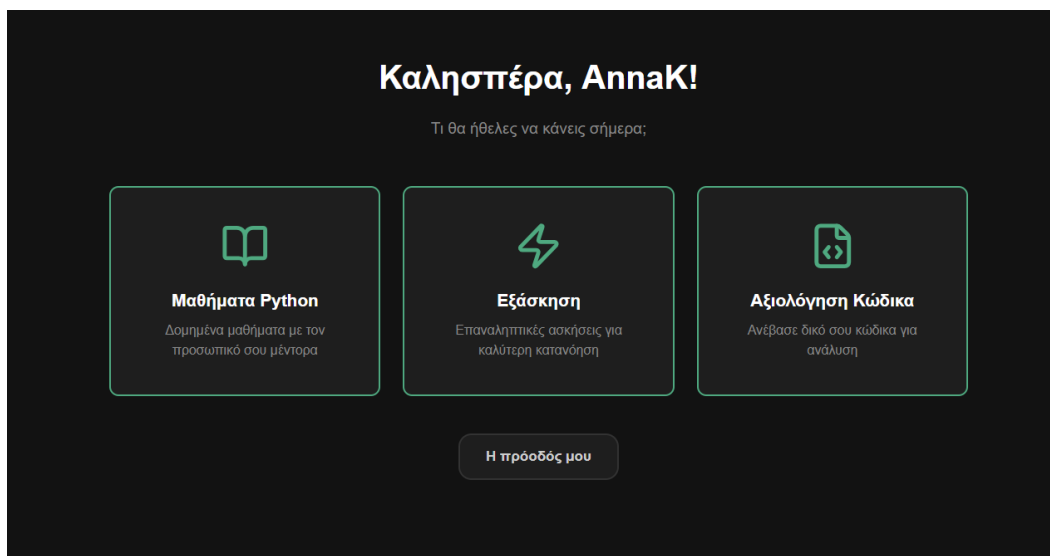




ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Σχεδίαση και Υλοποίηση Αυτόνομου Πράκτορα
(Agentic AI) για την Εξατομικευμένη Υποστήριξη και
Αξιολόγηση Φοιτητών στον Προγραμματισμό.»



Της φοιτήτριας
Άννα Καργαλίδου
Αρ. Μητρώου: 2021057

Επιβλέπων
Ονοματεπώνυμο Καλλιόπη Κράβαρη
Βαθμίδα Επίκουρη Καθηγήτρια

Ημερομηνία 2/9/2026

Τίτλος Δ.Ε. Σχεδίαση και Υλοποίηση Αυτόνομου Πράκτορα (Agentic AI) για την Εξατομικευμένη
Υποστήριξη και Αξιολόγηση Φοιτητών στον Προγραμματισμό

Κωδικός Δ.Ε. 26137

Ονοματεπώνυμο φοιτητή/τών Άννα Καργαλίδου

Ονοματεπώνυμο εισηγητή Καλλιόπη Κράβαρη

Ημερομηνία ανάληψης Δ.Ε 11/3/2026

Ημερομηνία περάτωσης Δ.Ε 2/9/2026

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία της φοιτήτριας Άννα Καργαλίδου που την εκπόνησε/αν. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραίτητα και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Έχοντας βιώσει τα πολυπληθή τμήματα των πρώτων βασικών μαθημάτων προγραμματισμού, και τη δυσκολία εκμάθησης με την οποία συναρτούνται, η υλοποίηση ενός συστήματος που μπορεί να προσφέρει εξατομικευμένη προσοχή στην διαδικασία της διδασκαλίας μου φάνηκε σαν μια πολύ καλή απάντηση σε αυτό το πρόβλημα. Η ιδέα ενός εξειδικευμένου συστήματος που συνδυάζει την ευελιξία των Μεγάλων Γλωσσικών Μοντέλων με ντετερμινιστικούς, ελέγξιμους μηχανισμούς, συνάδει εξαιρετικά με τις ανάγκες της εποχής, προσφέροντας ένα πολύ ενδιαφέρον αντικείμενο μελέτης. Μέσα από την εκπόνηση αυτής της εργασίας μου δόθηκε η ευκαιρία να δουλέψω σε βάθος με μια πολυπρακτορική αρχιτεκτονική, από τον αρχικό σχεδιασμό μέχρι την τελική αξιολόγηση, κάτι που ξεπερνά κατά πολύ τη θεωρητική μελέτη τέτοιων συστημάτων. Ασχολήθηκα εξίσου με ζητήματα σχεδίασης λογισμικού, παιδαγωγικής θεωρίας και μεθοδολογίας αξιολόγησης. Η εμπειρία αυτή μου έδειξε πρακτικά τις δυνατότητες που μπορεί να προσφέρει ένα καλά σχεδιασμένο σύστημα, βοηθώντας μελλοντικά ακόμα και τους πιο άπειρους φοιτητές στα πρώτα τους βήματα εκμάθησης προγραμματισμού.

Περίληψη

Η εξατομικευμένη διδασκαλία βελτιώνει μετρήσιμα τα μαθησιακά αποτελέσματα σε σχέση με τη συμβατική διδασκαλία σε τάξη, δεν κλιμακώνεται όμως εύκολα σε τάξεις με πολλούς φοιτητές, ιδίως σε εισαγωγικά μαθήματα προγραμματισμού. Τα Μεγάλα Γλωσσικά Μοντέλα (LLM) προσφέρονται ως πιθανή λύση, αλλά δεν έχουν σχεδιαστεί ειδικά για εκπαιδευτικά πλαίσια και τείνουν να παράγουν πειστική αλλά εσφαλμένη πληροφορία.

Η παρούσα εργασία σχεδιάζει και υλοποιεί ένα σύστημα Agentic AI για την υποστήριξη φοιτητών στην εκμάθηση της Python, μέσω πολυπρακτορικής αρχιτεκτονικής τριών εξειδικευμένων πρακτόρων: έναν πράκτορα καθοδήγησης που παρουσιάζει θεωρία και παράγει υποδείξεις, έναν πράκτορα εντοπισμού σφαλμάτων που αναλύει στατικά τον κώδικα του φοιτητή, και έναν πράκτορα αξιολόγησης που κρίνει την ικανοποίηση κριτηρίων και αποφασίζει την πρόοδο. Και οι τρεις ακολουθούν το ίδιο μοτίβο σχεδίασης: ντετερμινιστική θεμελίωση των ευρημάτων πριν από κάθε κρίση LLM, ώστε η γλωσσική ικανότητα του μοντέλου να στηρίζεται σε επαληθευμένα δεδομένα και όχι σε εικασίες. Το σύστημα υλοποιήθηκε με FastAPI, React και LangGraph, και διατηρεί ανοιχτό μοντέλο μαθητή (Open Learner Model) που εκθέτει στον φοιτητή την πρόοδό του ανά ενότητα.

Η αξιολόγηση συνδύασε σύγκριση με ένα απλούστερο σύστημα μίας κλήσης LLM και πειράματα αφαίρεσης πάνω στο ίδιο το σύστημα, σε δείγματα κώδικα σχεδιασμένα να ελέγχουν ορθότητα κρίσης, ανθεκτικότητα σε μη κανονικές υποβολές και ευαισθησία σε επιφανειακά μοτίβα. Το πολυπρακτορικό σύστημα ισοφάρισε την ακρίβεια κρίσης του απλούστερου συστήματος, παρήγαγε όμως παιδαγωγική απόφαση προόδου στηριγμένη στο ιστορικό του φοιτητή, την οποία το απλούστερο σύστημα δεν προσφέρει καν, και θεμελίωσε πλήρως όλες τις εξόδους του στον πραγματικό κώδικα, έναντι σημαντικού ποσοστού αστήρικτων ισχυρισμών χωρίς τη δομημένη πληροφορία. Το κόστος αυτής της αξιοπιστίας είναι μετρήσιμο, με τον χρόνο απόκρισης να πολλαπλασιάζεται σημαντικά σε σχέση με τη μία κλήση.

Η εργασία δείχνει ότι ο συνδυασμός ντετερμινιστικής θεμελίωσης και κρίσης LLM, επαναλαμβανόμενος σε κάθε πράκτορα, παράγει μετρήσιμα πιο αξιόπιστη συμπεριφορά από μια ανεξέλεγκτη μοναδική κλήση.

«Design and Implementation of an Agentic AI for Personalized Student Support and Assessment in Programming»

«Anna Kargalidou»

Abstract

Personalized instruction measurably improves learning outcomes over conventional classroom teaching, but does not scale easily to large classes, particularly in introductory programming courses. Large Language Models (LLMs) are often proposed as a solution, yet they are not purpose-built for educational contexts and tend to produce convincing but incorrect information.

This thesis designs and implements an Agentic AI system for supporting students learning Python, through a multi-agent architecture of three specialized agents: a Mentoring Agent that presents theory and produces hints, a Debugging Agent that statically analyzes student code, and an Assessment Agent that judges criterion satisfaction and decides on progression. All three follow the same design pattern: deterministic grounding of findings before any LLM judgment, so that the model's language ability is always anchored to verified data rather than assumptions. The system was implemented with FastAPI, React, and LangGraph, and maintains an Open Learner Model that exposes the student's progress per lesson.

Evaluation combined comparison with a simpler single-call LLM system and ablation experiments within the system itself, on code samples designed to test judgment correctness, robustness to non-standard submissions, and sensitivity to surface patterns. The multi-agent system matched the judgment accuracy of the simpler system, while producing a pedagogical progression decision grounded in the student's history that the simpler system does not offer at all, and fully grounding all of its outputs in the actual code, against a substantial share of unsupported claims without the structured information. The cost of this reliability is measurable, with response time increasing considerably relative to the single call.

The thesis shows that combining deterministic grounding with LLM judgment, repeated at every agent, produces measurably more reliable behavior than an unchecked single call.

Περιεχόμενα

Πρόλογος.....	iii
Περίληψη.....	iv
Abstract	v
Περιεχόμενα	vi
Κατάλογος Σχημάτων	viii
Κατάλογος Πινάκων.....	viii
Συντομογραφίες.....	ix
Κεφάλαιο 1ο: Εισαγωγή.....	1
1.1 Πλαίσιο και Κίνητρο	1
1.2 Σκοπός και Στόχοι	1
1.3 Δομή της Εργασίας.....	2
Κεφάλαιο 2ο: Βιβλιογραφική Ανασκόπηση.....	2
2.1 Εισαγωγή	2
2.2 Θεμελιώδης Θεωρία Πρακτόρων	2
2.3 Πολυπρακτορικά Συστήματα και LLM-based Orchestration.....	3
2.4 Έξυπνα Συστήματα Διδασκαλίας (ITS)	5
2.5 LLMs σε Εκπαιδευτικές Εφαρμογές	5
2.6 Hallucinations.....	6
2.7 Επίλογος	7
Κεφάλαιο 3ο: Αρχιτεκτονική	7
3.1 Εισαγωγή	7
3.2 Γενική Αρχιτεκτονική Συστήματος.....	7
3.3 Ντετερμινιστική Θεμελίωση και Κρίση LLM	11
3.4 Mentoring Agent	12
3.4.1 Παρουσίαση περιεχομένου και ανάθεση ασκήσεων	12
3.4.2 Παραγωγή και κλιμάκωση υποδείξεων	13
3.4.3 Συμπληρωματικές λειτουργίες	16
3.5 Debugging Agent	17
3.6 Assessment Agent	20
3.7 Open Learner Model & Αυτορρύθμιση.....	23
3.8 Επίλογος	25
Κεφάλαιο 4ο: Υλοποίηση.....	26

4.1	Εισαγωγή.....	26
4.2	Τεχνολογική Στοιβά.....	27
4.3	Σχεδιασμός Βάσης Δεδομένων.....	29
4.4	Σχεδιασμός Prompts.....	31
4.5	Λογική Καθοδήγησης (Mentoring Agent).....	34
4.6	Λογική Εντοπισμού Σφαλμάτων (Debugging Agent).....	36
4.7	Λογική Κριτηρίων (Assessment Agent).....	39
4.8	Επίπεδο Ενορχήστρωσης.....	42
4.8.1	Ο κύκλος της κύριας ροής.....	42
4.8.2	Μεταβάσεις και προσαρμογή δυσκολίας.....	43
4.8.3	Εκτέλεση και αποθήκευση.....	43
4.8.4	Εξάσκηση, Αξιολόγηση Κώδικα και έναρξη συνεδρίας.....	44
4.9	Open Learner Model – Υλοποίηση.....	45
4.10	Frontend & Διεπαφή Χρήστη.....	46
4.10.1	Πιστοποίηση και αρχική οθόνη.....	46
4.10.2	Ο επεξεργαστής κώδικα και ο χρονιστής αδράνειας.....	48
4.10.3	Συμπληρωματικές οθόνες και ιστορικό.....	52
4.11	Προκλήσεις Υλοποίησης.....	53
4.12	Επίλογος.....	54
Κεφάλαιο 5ο:	Αξιολόγηση.....	55
5.1	Εισαγωγή.....	55
5.2	Μέθοδοι Αξιολόγησης.....	56
5.3	Δείγματα.....	58
5.4	Πρώτη Μέθοδος: Σύγκριση με το Σύστημα Αναφοράς.....	59
5.5	Δεύτερη Μέθοδος: Πειράματα Αφαίρεσης.....	61
5.6	Κόστος και Χρόνος Απόκρισης.....	64
5.7	Σύγκριση, Συμπεράσματα και Περιορισμοί.....	65
Κεφάλαιο 6ο:	Συμπεράσματα και Μελλοντική Εργασία.....	67
6.1	Εισαγωγή.....	67
6.2	Ανακεφαλαίωση και Συνεισφορά.....	68
6.3	Επέκταση του Μηχανισμού Επικύρωσης σε Σύνθετες Δομικές Συνθήκες.....	69
6.4	Κατευθύνσεις Μελλοντικής Εργασίας.....	69
6.5	Επίλογος.....	70
BIBΛΙΟΓΡΑΦΙΑ.....		71
ΠΑΡΑΡΤΗΜΑ Α : ΑΠΟΣΠΑΣΜΑΤΑ ΚΩΔΙΚΑ.....		74

Κατάλογος Σχημάτων

Σχήμα 2.1. Τρόποι οργάνωσης πολυπρακτορικών συστημάτων.....	4
Σχήμα 3.1. Αρχιτεκτονικά επίπεδα του συστήματος.....	8
Σχήμα 3.2. Ροή δρομολόγησης ανά λειτουργία	10
Σχήμα 3.3. Ντετερμινιστική θεμελίωση και κρίση LLM	11
Σχήμα 3.4. Απόφαση του Mentor για υπόδειξη	15
Σχήμα 4.1. Σχεδιασμός βάσης δεδομένων	29
Σχήμα 4.2. Οθόνη εγγραφής.....	46
Σχήμα 4.3. Οθόνη σύνδεσης.....	47
Σχήμα 4.4. Αρχική οθόνη	48
Σχήμα 4.5. Μοντέλο προόδου με ένδειξη δυσκολίας.....	48
Σχήμα 4.6. Ενεργή άσκηση, ο φοιτητής γράφει κώδικα	49
Σχήμα 4.7. Κλειδωμένος editor όσο δεν υπάρχει ενεργή άσκηση	49
Σχήμα 4.8. Παράδειγμα συγκεκριμένης υπόδειξης.....	51
Σχήμα 4.9. Επιτυχής ολοκλήρωση άσκησης.....	51
Σχήμα 4.10. Οθόνη Εξάσκησης	52
Σχήμα 4.11. Οθόνη Αξιολόγησης Κώδικα	52
Σχήμα A.1. Απόσπασμα prompt με ρητή θεμελίωση ευρημάτων (Debugger)	74
Σχήμα A.2. Παραγωγή άσκησης από πρότυπο	75
Σχήμα A.3. Έλεγχος πιθανού ατέρμονου βρόχου while	75
Σχήμα A.4. Αντιστοίχιση κριτηρίου σε δομικό εύρημα.....	76

Κατάλογος Πινάκων

Πίνακας 4.1. Χρονικά όρια του χρονιστή αδράνειας ανά στάδιο	50
Πίνακας 5.1. Δείγματα του συνόλου απόφασης προόδου (D1-D4).....	58
Πίνακας 5.2. Χρήση κάθε συνόλου δειγμάτων ανά ενότητα αξιολόγησης.....	59
Πίνακας 5.3. Ορθότητα κρίσης ανά σύνολο δειγμάτων	60
Πίνακας 5.4. Μέσο μήκος ανατροφοδότησης ανά σύνολο δειγμάτων	61
Πίνακας 5.5. Αποτελέσματα πειράματος αφαίρεσης στην απόφαση προόδου	62
Πίνακας 5.6. Θεμελίωση εξόδων με και χωρίς δομημένη πληροφορία	63
Πίνακας 5.7. Σύγκριση τήρησης πολιτικής: πλήρες σύστημα έναντι ενιαίου prompt	64
Πίνακας 5.8. Κόστος και χρόνος απόκρισης ανά υποβολή.....	65
Πίνακας 5.9. Κατανομή πλήθους κλήσεων μοντέλου ανά περίπτωση.....	65

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
AST	Abstract Syntax Tree
API	Application Programming Interface
ITS	Intelligent Tutoring System
LLM	Large Language Model
OLM	Open Learner Model
RAG	Retrieval-Augmented Generation

Κεφάλαιο 1ο: Εισαγωγή

1.1 Πλαίσιο και Κίνητρο

Η εκμάθηση προγραμματισμού αποτελεί για πολλούς φοιτητές τμημάτων Πληροφορικής μια εξαιρετικά απαιτητική διαδικασία. Αυτό οφείλεται στην ανάγκη ταυτόχρονης κατανόησης αφηρημένων εννοιών, ανάπτυξης δομημένης, λογικής σκέψης καθώς και ικανότητας εντοπισμού και διόρθωσης σφαλμάτων [7]. Η ανάγκη για βελτίωση της διδακτικής μεθοδολογίας είναι διαπιστωμένη εδώ και δεκαετίες. Ήδη από το 1984 ο ψυχολόγος Benjamin Bloom ανέδειξε το λεγόμενο “Πρόβλημα των Δυο Τυπικών Αποκλίσεων” (2-Sigma Problem), σύμφωνα με το οποίο η εξατομικευμένη, ένα προς ένα διδασκαλία οδηγεί σε καλύτερα μαθησιακά αποτελέσματα σε σχέση με τη συμβατική διδασκαλία σε τάξη, όπως επισημαίνεται στη σχετική βιβλιογραφία [1]. Η ατομικά προσαρμοσμένη καθοδήγηση, ωστόσο, δεν είναι εύκολα εφαρμόσιμη σε τάξεις με μεγάλο αριθμό φοιτητών [1]. Το πρόβλημα αυτό κλιμακώνεται στα πλαίσια εισαγωγικών μαθημάτων προγραμματισμού όπου ο αριθμός των φοιτητών είναι δυσανάλογος του διδακτικού προσωπικού, όπως αποδεικνύεται από μελέτες βασισμένες σε πραγματικές συνθήκες τάξης [2].

Παράλληλα, η ραγδαία εξέλιξη των Μεγάλων Γλωσσικών Μοντέλων (Large Language Models - LLMs) τα τελευταία χρόνια έχει οδηγήσει σε ευρεία υιοθέτηση τους ως εργαλεία υποστήριξης της εκπαιδευτικής διαδικασίας. Αυτό συμβαίνει συχνά με την παραδοχή ότι μπορούν να λειτουργήσουν ως αυτόνομα συστήματα εξατομικευμένης διδασκαλίας. Όμως, σύμφωνα με πρόσφατη έρευνα, η παραδοχή αυτή είναι εσφαλμένη. Τα LLMs δεν έχουν σχεδιαστεί ειδικά για εκπαιδευτικά ή παιδαγωγικά πλαίσια, με αποτέλεσμα να παρουσιάζουν ασυνεπείς απαντήσεις με χρήση πλαστών πληροφοριών, φαινόμενο γνωστό και ως “ψευδαισθήσεις” (hallucinations) [3]. Η έρευνα αυτή καταλήγει συμπερασματικά στην υβριδική ενσωμάτωση της δομημένης γνώσης σε συνδυασμό με τα LLM ως προτιμότερη λύση, θέση η οποία λειτούργησε σαν έναυσμα για την υλοποίηση της παρούσας εργασίας.

Με βάση τα παραπάνω, γίνεται φανερό η ανάγκη για ένα σύστημα το οποίο θα προσφέρει την εξατομικευμένη υποστήριξη που λείπει από τις παραδοσιακές μεθόδους διδασκαλίας, ενώ παράλληλα θα περιορίζει την αναξιοπιστία των LLM με χρήση ορισμένης βάσης γνώσης. Η παρούσα διπλωματική εργασία προτείνει μια τέτοια προσέγγιση μέσω μιας πολυπρακτορικής (multi-agent) αρχιτεκτονικής Agentic AI (Πρακτορική Τεχνητή Νοημοσύνη), όπου εξειδικευμένοι πράκτορες συνδυάζουν την ευελιξία των LLMs με ντετερμινιστικούς μηχανισμούς επαλήθευσης.

1.2 Σκοπός και Στόχοι

Βασικός σκοπός αυτής της εργασίας είναι ο σχεδιασμός και η υλοποίηση του προτεινόμενου συστήματος ώστε να λειτουργήσει ως ψηφιακός βοηθός διδασκαλίας για φοιτητές προγραμματισμού, συγκεκριμένα στην γλώσσα Python. Το σύστημα αποτελείται από τρεις εξειδικευμένους πράκτορες, ο καθένας υπεύθυνος για διαφορετικό κομμάτι της εκπαιδευτικής διαδικασίας. Ένας πράκτορας αναλαμβάνει την καθοδήγηση, παρέχοντας σημασιολογικές υποδείξεις χωρίς να αποκαλύπτει απευθείας τη λύση (Mentoring Agent). Ένας άλλος πράκτορας αναλύει και εντοπίζει σφάλματα στον κώδικα του φοιτητή (Debugging Agent). Ο τελευταίος είναι πράκτορας αξιολόγησης, που παρακολουθεί και κρίνει την πρόοδο του μαθητή (Assessment Agent).

Πέρα από την υλοποίηση καθαυτή, η εργασία θέτει δύο πιο συγκεκριμένους ερευνητικούς στόχους. Ο πρώτος αφορά τη μελέτη της αποτελεσματικότητας των Agentic Workflows στην εκπαιδευτική διαδικασία. Η εργασία δεν περιορίζεται στην περιγραφή του πως λειτουργεί μια πολυπρακτορική αρχιτεκτονική, αλλά προχωρά και στη σύγκρισή της με μια πιο απλή εναλλακτική, ώστε να διαπιστωθεί αν η προσέγγιση αυτή προσφέρει πραγματικό όφελος. Ο δεύτερος στόχος αφορά τη μείωση των ψευδαισθήσεων των Μεγάλων Γλωσσικών Μοντέλων μέσω της χρήσης δομημένης γνώσης. Όπου είναι εφικτό, η ελεύθερη παραγωγή κειμένου του LLM αντικαθιστάται από ντετερμινιστική, ελέγξιμη ανάλυση, για πιο αιτιοκρατικά αποτελέσματα.

1.3 Δομή της Εργασίας

Η εργασία αυτή οργανώνεται σε έξι κεφάλαια, ξεκινώντας από το θεωρητικό υπόβαθρο και καταλήγοντας στην αξιολόγηση της υλοποίησης. Το Κεφάλαιο 2 παρουσιάζει τη σχετική βιβλιογραφική έρευνα πάνω στην οποία στηρίζεται η εργασία. Το Κεφάλαιο 3 αναλύει την αρχιτεκτονική του προτεινόμενου συστήματος και το Κεφάλαιο 4 τις τεχνικές λεπτομέρειες της υλοποίησης. Το Κεφάλαιο 5 παρουσιάζει τη μεθοδολογία και τα αποτελέσματα της αξιολόγησης. Τέλος, το Κεφάλαιο 6 συνοψίζει τη συνεισφορά της εργασίας και προτείνει κατευθύνσεις για μελλοντική έρευνα.

Κεφάλαιο 2ο: Βιβλιογραφική Ανασκόπηση

2.1 Εισαγωγή

Το παρόν κεφάλαιο εξετάζει το θεωρητικό υπόβαθρο στο οποίο βασίζονται η αρχιτεκτονική και οι σχεδιαστικές επιλογές της υλοποίησης της εργασίας. Στην ενότητα 2.2, παρουσιάζεται η θεμελιώδης θεωρία πρακτόρων, ώστε να οριστεί η έννοια του “πράκτορα” και τα χαρακτηριστικά που τον διαφοροποιούν τόσο από απλά λογισμικά συστήματα όσο και από τα Μεγάλα Γλωσσικά Μοντέλα (LLM). Στη συνέχεια, η ενότητα 2.3 εστιάζει στα πολυπρακτορικά συστήματα βασισμένα σε Μεγάλα Γλωσσικά Μοντέλα, καταγράφοντας τις σύγχρονες τάσεις οργάνωσης πολλαπλών πρακτόρων. Η ενότητα 2.4 στρέφεται στα Έξυπνα Συστήματα Διδασκαλίας (ITS), το ευρύ πεδίο εφαρμογής της εργασίας, ενώ η ενότητα 2.5 εξετάζει ειδικότερα τη χρήση LLMs σε εκπαιδευτικές εφαρμογές. Τέλος η 2.6 παρουσιάζει το πρόβλημα των “ψευδαισθήσεων” (hallucinations), και τις κύριες τεχνικές αντιμετώπισης του, ενώ η ενότητα 2.7 συνοψίζει τα βασικά συμπεράσματα του κεφαλαίου.

2.2 Θεμελιώδης Θεωρία Πρακτόρων

Ο όρος “πράκτορας” (agent) στην τεχνητή νοημοσύνη ορίζεται ως ένα υπολογιστικό σύστημα το οποίο είναι τοποθετημένο σε ένα περιβάλλον και έχει την ικανότητα να δρα αυτόνομα, με ευέλικτο τρόπο, προκειμένου να επιτύχει τους στόχους για τους οποίους έχει σχεδιαστεί [4]. Από αυτόν τον ορισμό προκύπτουν τέσσερις βασικές ιδιότητες που πρέπει να διαθέτει ένας πράκτορας: αυτονομία, δηλαδή έλεγχος της δικής του κατάστασης και συμπεριφοράς χωρίς άμεση εξωτερική παρέμβαση. Αντιδραστικότητα, όπου είναι η ικανότητα αντίληψης του περιβάλλοντος και η έγκαιρη ανταπόκριση σε αυτό. Προδραστικότητα, δηλαδή συμπεριφορά που πηγάζει από δομημένους στόχους και δεν περιορίζεται σε απλή αντίδραση σε γεγονότα, και κοινωνική ικανότητα, δηλαδή δυνατότητα συνεργασίας με άλλους πράκτορες ή με ανθρώπους [4].

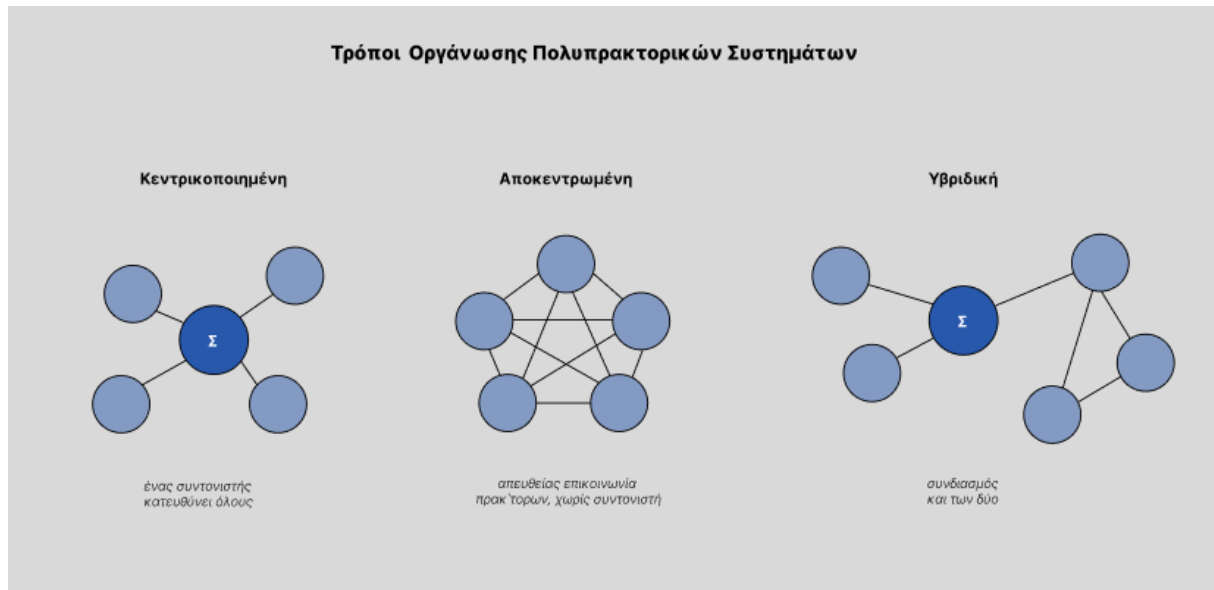
Η διαφορά ανάμεσα σε έναν πράκτορα και σε ένα απλό αντικείμενο λογισμικού (object) είναι σημαντική. Ένα αντικείμενο διαθέτει μεθόδους οι οποίες μπορούν να κληθούν οποτεδήποτε, χωρίς το ίδιο να ελέγχει πότε ή γιατί θα εκτελεστεί κάποια ενέργεια. Από την άλλη, ένας πράκτορας αποφασίζει από μόνος του πότε και υπό ποιες συνθήκες θα δράσει με βάση τους δικούς του στόχους, καθιστώντας τον πολύ πιο ικανό για ευέλικτη προσαρμογή σε απρόβλεπτες καταστάσεις [5].

Μια ακόμα σημαντική διάκριση αφορά τα Μεγάλα Γλωσσικά Μοντέλα (LLMs). Ένα LLM είναι ένα μοντέλο βαθιάς μάθησης (deep learning), εκπαιδευμένο πάνω σε τεράστιο όγκο κειμενικών δεδομένων ώστε να κατανοεί και να παράγει φυσική γλώσσα, προσεγγίζοντας γενικού τύπου νοημοσύνη [3]. Από μόνο του, ένα LLM δεν αποτελεί πράκτορα με την έννοια που ορίστηκε πάνω. Αν και είναι μια ισχυρή μηχανή συλλογισμού και παραγωγής κειμένου, δεν έχει την ικανότητα για αντίληψη του περιβάλλοντος, αλλά ούτε και μνήμη ανάμεσα σε διαφορετικές κλήσεις [10]. Μπορεί να χαρακτηριστεί ως πράκτορας μόνο όταν ενσωματωθεί μέσα σε μια ευρύτερη αρχιτεκτονική που του προσδίδει τις ιδιότητες που ήδη αναφέρθηκαν. Το LLM λειτουργεί δηλαδή ως το συλλογιστικό κομμάτι του πράκτορα, ενώ χάρη στην αρχιτεκτονική του αποκτά ικανότητες αυτονομίας, αντιδραστικότητας, προδραστικότητας και κοινωνικότητας.

2.3 Πολυπρακτορικά Συστήματα και LLM-based Orchestration

Ο όρος πολυπρακτορικό σύστημα αναφέρεται σε μια ομάδα αυτόνομων πρακτόρων που δρουν μέσα στο ίδιο περιβάλλον, με σκοπό την επίτευξη κοινών ή ατομικών στόχων. Οι πράκτορες αυτοί μπορούν είτε να συνεργάζονται μεταξύ τους, είτε και να ανταγωνίζονται, ανάλογα με την φύση των στόχων που έχει θέσει ο καθένας [4].

Τα τελευταία χρόνια η ανάπτυξη των Μεγάλων Γλωσσικών Μοντέλων έδωσε νέα ώθηση σε αυτή την ιδέα, καθιστώντας εφικτή την οργάνωση πολλών LLM-based πρακτόρων μαζί, τον καθένα με διαφορετικό ρόλο, ώστε να συνεργάζονται για την επίλυση ενός προβλήματος. Αυτός ο συνδυασμός (πολλαπλοί πράκτορες βασισμένοι σε LLM, οργανωμένοι ώστε να συνεργάζονται σε ένα κοινό έργο) αναφέρεται συχνά στη βιβλιογραφία με τον όρο Agentic AI (Πρακτορική Τεχνητή Νοημοσύνη) ή Agentic Workflows [8]. Οι τρόποι οργάνωσης τέτοιων συστημάτων χωρίζονται συνήθως σε τρεις κατηγορίες: κεντριοποιημένη, όπου ένας κεντρικός μηχανισμός ή ένας συντονιστής πράκτορας συντονίζει όλους τους υπόλοιπους, αποκεντρωμένη, όπου πράκτορες επικοινωνούν απευθείας μεταξύ τους χωρίς κάποιο συντονιστή, και υβριδική, η οποία συνδυάζει στοιχεία και των δυο (Σχήμα 2.1) [8].



Σχήμα 2.1. Τρόποι οργάνωσης πολυπρακτορικών συστημάτων

Η διάκριση ενός πολυπρακτορικού συστήματος από ένα μεμονωμένο μοντέλο δεν είναι αυτονόητη και αξίζει να γίνει κατανοητή. Ένα LLM μπορεί να λάβει οδηγία να υιοθετήσει διαδοχικά διαφορετικούς ρόλους μέσα στο ίδιο prompt, δηλαδή μέσα στο ίδιο κείμενο οδηγιών και δεδομένων που δίνεται ως είσοδος στο μοντέλο, και το αποτέλεσμα μοιάζει εξωτερικά με συνεργασία. Οι ρόλοι εκεί δεν διαθέτουν δική τους κατάσταση, δεν ενεργοποιούνται από ρητή δρομολόγηση, και τίποτα δεν εξασφαλίζει ότι ο ένας θα σεβαστεί τα όρια του άλλου, αφού όλοι παράγονται από την ίδια κλήση. Σε ένα πολυπρακτορικό σύστημα κάθε πράκτορας αποτελεί ξεχωριστό υπολογιστικό βήμα, με δικό του prompt, δική του είσοδο, και δικό του αποτέλεσμα που καταγράφεται πριν περάσει στον επόμενο.

Η ιδέα της ανάθεσης ρητά ορισμένων ρόλων σε πράκτορες μέσα σε εκπαιδευτικά συστήματα προηγείται κατά πολύ των Μεγάλων Γλωσσικών Μοντέλων. Ήδη από τα μέσα της δεκαετίας του 2000 έχει προταθεί η τυπική περιγραφή του ρόλου ενός πράκτορα διδασκαλίας μέσα από τις δεξιότητες και τις αρμοδιότητες που τον συνοδεύουν, με διακριτούς ρόλους για την υποβολή ερωτήσεων, την κρίση της απάντησης του μαθητή, και την εξατομικευμένη καθοδήγηση [6].

Τέτοια συστήματα ωστόσο συνοδεύονται από αρκετά προβλήματα. Έρευνες δείχνουν ότι συχνά οι ρόλοι των πρακτόρων επικαλύπτονται και ενδέχεται να μην υπάρχει ευθυγράμμιση στον κοινό στόχο. Αυτές οι δυσκολίες δεν οφείλονται τόσο στους περιορισμούς του ίδιου του LLM, όσο στον τρόπο σχεδίασης της αρχιτεκτονικής [9]. Για την αντιμετώπιση αυτού του ζητήματος έχει προταθεί η έννοια του «bounded determinism», σύμφωνα με την οποία, συγκεκριμένα τμήματα ενός υβριδικού συστήματος πρέπει να λειτουργούν με προβλέψιμο και ελέγξιμο τρόπο, ενώ άλλα μπορούν να παραμείνουν ευέλικτα μέσω LLM [9].

Η λογική αυτή έχει οδηγήσει και σε πιο συγκεκριμένες αρχιτεκτονικές προτάσεις. Μια από αυτές διαχωρίζει το σύστημα σε δυο επίπεδα: ένα επίπεδο συντονισμού των πρακτόρων, και ένα ξεχωριστό επίπεδο αποθήκευσης δομημένης γνώσης [10]. Ανάλογη λογική διαχωρισμού συναντάται σε πληθώρα πεδίων εφαρμογής, ανάμεσα στα οποία βρίσκονται και τα συστήματα διαχείρισης δορυφορικών δικτύων [11], γεγονός που υποδεικνύει πως πρόκειται για ένα αναγνωρισμένο αρχιτεκτονικό μοτίβο. Επιπλέον, σε σχετική μελέτη, ένα πολυπρακτορικό σύστημα συγκρίθηκε τόσο με ένα απλό LLM όσο

και με ένα LLM που αντλούσε επιπλέον πληροφορίες από εξωτερική πηγή πριν απαντήσει. Και στις δυο περιπτώσεις το πολυπρακτορικό σύστημα σημείωσε σημαντικά καλύτερα αποτελέσματα [12].

2.4 Έξυπνα Συστήματα Διδασκαλίας (ITS)

Πριν μιλήσουμε για την ενσωμάτωση των LLMs σε εκπαιδευτικά συστήματα, χρειάζεται να κάνουμε μια σύντομη αναφορά στο ευρύτερο πεδίο των Έξυπνων Συστημάτων Διδασκαλίας (Intelligent Tutoring Systems - ITS), από το οποίο προέρχεται και εξελίσσεται η παρούσα έρευνα. Τα ITS ορίζονται ως προγράμματα υπολογιστή που εφαρμόζουν τεχνικές τεχνητής νοημοσύνης για να αυτοματοποιήσουν και να εξατομικεύσουν τη διδασκαλία, προσπαθώντας να κατανοήσουν πως μαθαίνει ο φοιτητής μέσα από την ίδια την αλληλεπίδραση του με το σύστημα [7], [13].

Η κλασική αρχιτεκτονική ενός ITS αποτελείται κυρίως από τέσσερα βασικά κομμάτια. Αρχικά, έχουμε το expert/domain module, το οποίο περιέχει τη γνώση του πεδίου και χρησιμεύει ως πρότυπο αξιολόγησης. Έπειτα, το student model, που αναπαριστά τη διαρκώς εξελισσόμενη γνώση και τις δεξιότητες του μαθητή. Μετά το tutoring module που σχεδιάζει και ελέγχει τον τρόπο παράδοσης της διδασκαλίας, και τέλος το user interface, υπεύθυνο για την επικοινωνία του μαθητή με το σύστημα [13]. Η επιτυχία τέτοιων συστημάτων βασίζεται συνήθως σε προσεκτικά σχεδιασμένη προσαρμοστικότητα, ανατροφοδότηση και έλεγχο των διδακτικών στρατηγικών [14].

Παρά την τεκμηριωμένη αποτελεσματικότητα τους, τα κλασικά ITS παρουσιάζουν ένα σοβαρό πρόβλημα κλιμάκωσης. Λόγω της εξειδίκευσης και του χρόνου που απαιτεί η κατασκευή των μοντέλων, η επεκτασιμότητα τους σε πολλαπλά εκπαιδευτικά περιβάλλοντα είναι εξαιρετικά περιορισμένη [14]. Συνάμα, στην κλασική τους μορφή, τα συστήματα αυτού του τύπου στηρίζουν την ανατροφοδότηση και την αλληλεπίδραση με τον φοιτητή σε αμιγώς αλγοριθμικές μεθόδους [13]. Το πώς μεταβάλλεται η εικόνα όταν στο ίδιο ακριβώς πλαίσιο προστεθεί ένα Μεγάλο Γλωσσικό Μοντέλο εξετάζεται στην επόμενη ενότητα.

2.5 LLMs σε Εκπαιδευτικές Εφαρμογές

Πέρα από τη γενική αρχιτεκτονική των ITS, τα τελευταία χρόνια έχουν γίνει πολλές έρευνες πάνω στο πως τα LLMs μπορούν να ενσωματωθούν σε εκπαιδευτικά συστήματα, σε διαφορετικά γνωστικά αντικείμενα. Στον προγραμματισμό, για παράδειγμα, έχει αποδειχτεί πειραματικά ότι LLMs όπως το Codex εντοπίζουν και διορθώνουν σφάλματα σε κώδικα πιο αποτελεσματικά από παραδοσιακά εργαλεία αυτόματης επιδιόρθωσης που βασίζονται σε βαθιά μάθηση [15]. Μια μελέτη πάνω στην πλατφόρμα CodeHelp έδειξε επίσης ότι, μετά από δώδεκα εβδομάδες χρήσης σε πραγματικά μαθήματα, οι φοιτητές που χρησιμοποιούσαν συχνά έναν βοηθό προγραμματισμού βασισμένο σε LLM πετύχαιναν καλύτερους βαθμούς [15].

Το φαινόμενο αυτό το βλέπουμε και σε άλλα γνωστικά αντικείμενα. Σε συστήματα διδασκαλίας SQL, ένα LLM αναλαμβάνει να παράγει μόνο του τη σωστή απάντηση σε μια άσκηση και να συγκρίνει την απάντηση του φοιτητή με αυτήν, εξηγώντας του γιατί είναι σωστή ή λάθος. Αυτή η εργασία παλιότερα γινόταν με πολύπλοκους αλγόριθμους που δυσκολεύονταν με ερωτήματα μέτριας πολυπλοκότητας και δεν έδιναν καμιά επεξήγηση [16]. Παράλληλα, σε συστήματα εκμάθησης ξένης γλώσσας, ένα μόνο LLM αναλαμβάνει πολλούς διαφορετικούς ρόλους μέσα στο ίδιο ITS. Πρόκειται για την περίπτωση που η ενότητα 2.3 διέκρινε από ένα πραγματικό πολυπρακτορικό σύστημα. Η ποιότητα του συστήματος αυτού βελτιώθηκε σημαντικά μέσω few-shot prompting, δηλαδή δίνοντάς του λίγα έτοιμα παραδείγματα μέσα στο ίδιο το prompt αντί να επανεκπαιδευτεί [17].

Ξεχωριστή θέση σε αυτή τη συζήτηση έχει η αυτόματη ανατροφοδότηση σε αρχάριους προγραμματιστές, το πεδίο στο οποίο εντάσσεται άμεσα η παρούσα εργασία. Η έρευνα εδώ προηγείται των Μεγάλων Γλωσσικών Μοντέλων. Έχει δειχθεί σε πραγματικές συνθήκες τάξης ότι η προσαρμοσμένη ανατροφοδότηση σε σφάλματα μεταγλώττισης προσφέρει μετρήσιμο λειτουργικό όφελος στην επίλυση σφαλμάτων από αρχάριους [2], ενώ εργαλεία που ταξινομούν τα σφάλματα σε προκαθορισμένες κατηγορίες [30] ή που παράγουν διορθωμένη εκδοχή του προγράμματος συγκρίνοντάς το με λύση αναφοράς [29] αναπτύχθηκαν και αξιολογήθηκαν χωρίς καθόλου Μεγάλα Γλωσσικά Μοντέλα. Οι προσεγγίσεις αυτές εξετάζονται αναλυτικότερα στην ενότητα 3.5, όπου συγκρίνονται με τις επιλογές του δικού μας συστήματος.

Σε όλα αυτά τα παραδείγματα ωστόσο υπάρχει κάτι κοινό. Το LLM δεν αποφασίζει ποτέ μόνο του, χωρίς κανένα έλεγχο, για το αν κάτι είναι σωστό. Ένα ενδεικτικό παράδειγμα αυτού βρίσκεται σε ένα σύστημα διδασκαλίας γραμματικής, όπου ένας απλός αλγόριθμος ελέγχει μαθηματικά αν η γραμματική του φοιτητή είναι σωστή, και το LLM χρησιμοποιείται μόνο στο τέλος, για να εξηγήσει το αποτέλεσμα με απλά λόγια [13]. Αυτό δεν είναι σύμπτωση, καθώς όσο περισσότερη εμπειρία αποκτιέται από την ενσωμάτωση LLMs σε ITS, τόσο πιο ξεκάθαρο γίνεται ότι τα LLMs, όσο καλά και αν παράγουν φυσική γλώσσα, δεν εγγυούνται πλήρη αυτονομία. Σε αξιολογήσεις άλγεβρας για παράδειγμα, βρέθηκε πως μόνο οι μισοί περίπου διάλογοι που παράγει ένα LLM είναι χωρίς λάθη, ενώ η ποιότητα της απάντησης εξαρτάται από το τι πληροφορία του δίνεται μέσα στο prompt [14]. Για αυτό και οι πιο πρόσφατες αρχιτεκτονικές αντιμετωπίζουν το LLM σαν συμπληρωματικό κομμάτι και όχι σαν αυτόνομο. Δεν αντικαθιστά τα δομημένα θεμέλια ενός ITS, απλά τα ενισχύει [14]. Αυτή η ανάγκη για έναν μηχανισμό ελέγχου πέρα από την καθαρά γλωσσική ικανότητα ενός LLM συνδέεται άμεσα με το πρόβλημα των «ψευδαισθήσεων» και τις τεχνικές αντιμετώπισης του, στο οποίο αναφερόμαστε στην επόμενη ενότητα.

2.6 Hallucinations

Το φαινόμενο των «ψευδαισθήσεων» αναφέρεται στην τάση ενός LLM να παράγει πληροφορία που ακούγεται πειστική αλλά στην πραγματικότητα είναι εσφαλμένη, ανεπιβεβαίωτη ή εντελώς κατασκευασμένη. Αυτό συμβαίνει διότι το μοντέλο παράγει κείμενο με βάση στατιστικά που έχει μάθει από τα δεδομένα εκπαίδευσης του, χωρίς ενσωματωμένο μηχανισμό επαλήθευσης της αλήθειας όσων παράγει. Σε εκπαιδευτικό πλαίσιο, το πρόβλημα αυτό αποκτά ιδιαίτερη βαρύτητα, καθώς μια λανθασμένη αλλά πειστική απάντηση μπορεί να παραπλανήσει τον φοιτητή αντί να τον βοηθήσει [19].

Στην βιβλιογραφία έχουν προταθεί διάφορες τεχνικές αντιμετώπισης, όπως η θεμελίωση της απάντησης σε εξωτερική γνώση μέσω ανάκτησης σχετικού υλικού (RAG) [19], η υποστήριξη του LLM από συμβολικούς ή ντετερμινιστικούς μηχανισμούς [21], η συνεργασία πολλαπλών μοντέλων που αξιολογούν απαντήσεις με βάση τον βαθμό αβεβαιότητας τους [22] και η αυτοδιόρθωση της απάντησης σε επίπεδο πρότασης, όπου το ίδιο το μοντέλο εντοπίζει και αναθεωρεί τους ισχυρισμούς του πριν τους παρουσιάσει [20]. Επίσης έχουμε τις μετρικές εντοπισμού βασισμένες στην αβεβαιότητα του ίδιου του μοντέλου [23], καθώς και τεχνικές συγγραφής prompt όπως ο περιορισμός της εξόδου σε αυστηρά δομημένη μορφή [24].

Η δική μας προσέγγιση αξιοποιεί δύο από αυτές. Την υποστήριξη του μοντέλου από ντετερμινιστικούς μηχανισμούς [21], που αποτελεί το κεντρικό μοτίβο σχεδίασης ολόκληρου του

συστήματος και αναλύεται στην ενότητα 3.3, και τον περιορισμό της εξόδου σε αυστηρά δομημένη μορφή [24], του οποίου η εφαρμογή σε κάθε σημείο κλήσης περιγράφεται στην ενότητα 4.4.

2.7 Επίλογος

Το κεφάλαιο αυτό κάλυψε το θεωρητικό υπόβαθρο πάνω στο οποίο στηρίζεται η υλοποίηση της εργασίας. Ξεκίνησε από τη θεμελιώδη θεωρία πρακτόρων (2.2) και την οργάνωση του σε πολυπρακτορικά συστήματα βασισμένα σε LLM (2.3), πέρασε στο ευρύτερο πεδίο εφαρμογής των Έξυπνων Συστημάτων Διδασκαλίας (2.4) και στην ενσωμάτωση των LLMs σε αυτά (2.5), και κατέληξε στο πρόβλημα των ψευδαισθήσεων και τις τεχνικές αντιμετώπισης του (2.6).

Από αυτή τη βιβλιογραφική διαδρομή προκύπτει ένα επαναλαμβανόμενο συμπέρασμα. Τα LLMs προσφέρουν πραγματική αξία σε εκπαιδευτικά συστήματα, αλλά μόνο όταν ο ρόλος τους είναι σαφώς οριοθετημένος και συνοδεύεται από κάποιες μορφής ντετερμινιστικό έλεγχο, είτε αυτός προέρχεται από τον αρχιτεκτονικό σχεδιασμό, είτε από τεχνικές θεμελίωσης και επαλήθευσης. Η παρούσα εργασία χτίζει πάνω σε ακριβώς αυτό το συμπέρασμα. Η διαφορά βρίσκεται στην έκταση της εφαρμογής, καθώς ο έλεγχος αυτός δεν τοποθετείται σε ένα μόνο σημείο της ροής αλλά επαναλαμβάνεται σε καθέναν από τους τρεις πράκτορες. Χρησιμοποιώντας την πολυπρακτορική αρχιτεκτονική που περιγράφεται παρακάτω, συνδυάζει την ευελιξία των LLMs με μηχανισμούς επαλήθευσης, ακολουθώντας τη λογική που η βιβλιογραφία επανειλημμένα επιβεβαιώνει ως αποτελεσματική.

Κεφάλαιο 3ο: Αρχιτεκτονική

3.1 Εισαγωγή

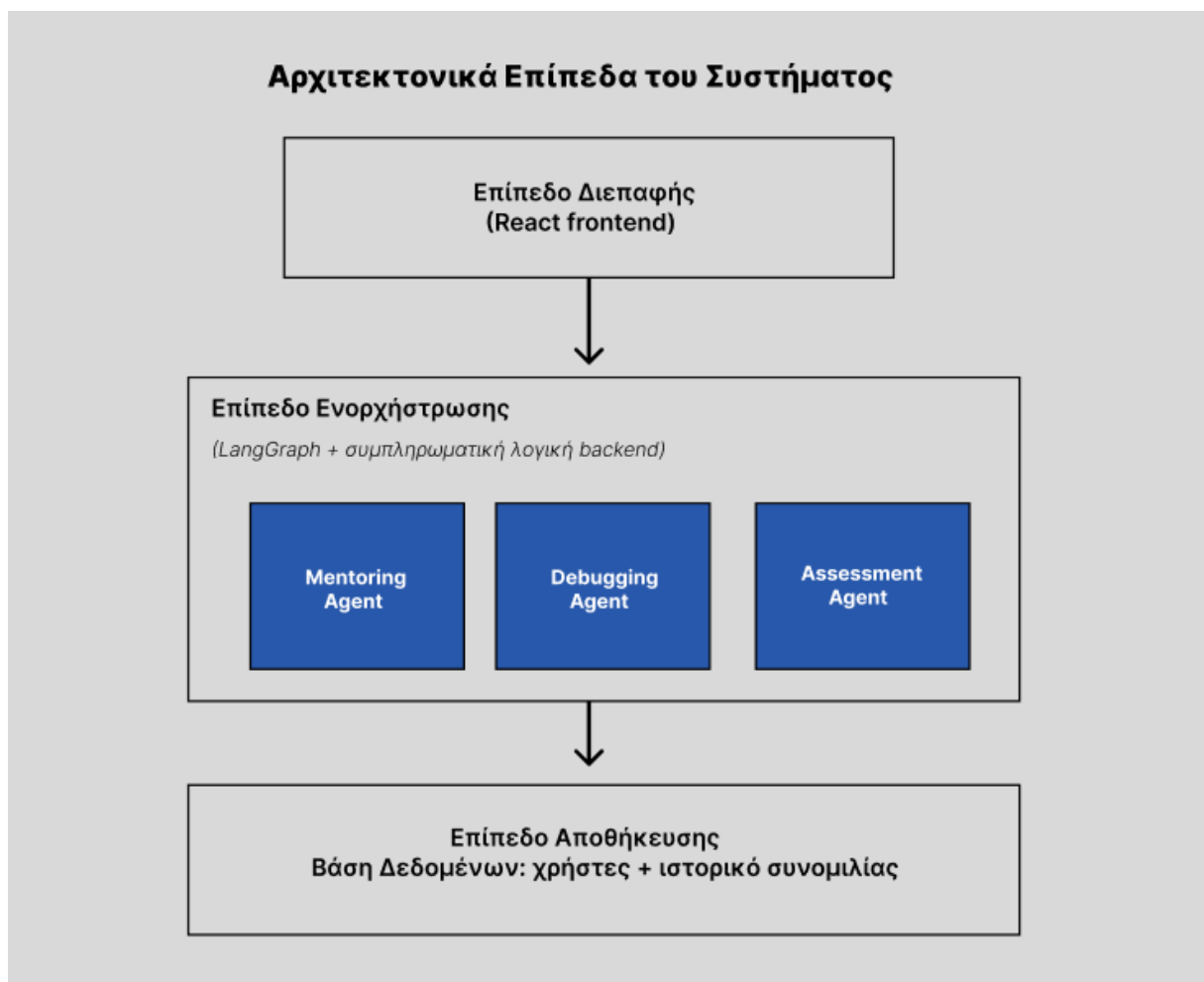
Το κεφάλαιο αυτό αναλύει την αρχιτεκτονική του προτεινόμενου συστήματος, χτίζοντας πάνω στο θεωρητικό υπόβαθρο του Κεφαλαίου 2. Η ενότητα 3.2 παρουσιάζει τη γενική αρχιτεκτονική του συστήματος, περιγράφοντας πως οργανώνονται και συνεργάζονται οι τρεις πράκτορες. Στη συνέχεια η ενότητα 3.3 αναλύει το κεντρικό μοτίβο σχεδίασης πάνω στο οποίο βασίζεται όλο το σύστημα, τον συνδυασμό της ντετερμινιστικής θεμελίωσης με την κρίση ενός LLM. Οι ενότητες 3.4, 3.5 και 3.6 εξετάζουν αναλυτικά τον καθένα από τους τρεις πράκτορες τεκμηριώνοντας τις επιμέρους σχεδιαστικές τους επιλογές μέσα από τις σχετικές παιδαγωγικές και τεχνικές θεωρίες. Η ενότητα 3.7 παρουσιάζει πως το σύστημα μοντελοποιεί τη γνώση και την πρόοδο του φοιτητή, ενισχύοντας παράλληλα την ικανότητα του για αυτορρύθμιση. Τέλος η ενότητα 3.8 συνοψίζει τα βασικά σημεία του κεφαλαίου.

3.2 Γενική Αρχιτεκτονική Συστήματος

Το σύστημα υποστηρίζει τρεις λειτουργικούς τρόπους χρήσης. Ο πυρήνας της εργασίας βρίσκεται στην κύρια ροή διδασκαλίας, η οποία περιλαμβάνει τη δομημένη αλληλεπίδραση θεωρίας, άσκησης, εντοπισμού σφαλμάτων και αξιολόγησης μέσα από τους τρεις πράκτορες (Mentoring, Debugging και Assessment Agent). Δυο ακόμα λειτουργίες προσφέρουν συμπληρωματικές δυνατότητες, ανεξάρτητες από τις βασικές απαιτήσεις της εργασίας, που υλοποιήθηκαν για παιδαγωγικούς σκοπούς. Συγκεκριμένα, υπάρχει η δυνατότητα για επαναληπτική εξάσκηση σε ασκήσεις ήδη διδαγμένης ύλης,

καθώς και η δυνατότητα ελεύθερης υποβολής κώδικα προς ανάλυση. Η επιλογή αυτών των δύο επιπλέον λειτουργιών δεν είναι αυθαίρετη. Σε πρόσφατη εφαρμογή σε πρώτο μάθημα προγραμματισμού, η βαθμιαία, επαναληπτική εξάσκηση στα βασικά συνδυάστηκε με μια δεύτερη φάση ανοιχτής εξερεύνησης, διαθέσιμη μόνο αφού ο φοιτητής είχε πρώτα δείξει επαρκή κατανόηση των θεμελιωδών δεξιοτήτων [1]. Η επαναληπτική εξάσκηση του παρόντος συστήματος εξυπηρετεί το πρώτο σκέλος, δίνοντας στον φοιτητή την ευκαιρία να παγιώσει όσα έχει ήδη διδαχθεί με τον δικό του ρυθμό. Η ελεύθερη υποβολή κώδικα εξυπηρετεί το δεύτερο, επιτρέποντας εξερεύνηση πέρα από τη σταθερή ακολουθία ασκήσεων, μόλις η βασική ύλη έχει καλυφθεί. Η γενική αρχιτεκτονική που ακολουθεί (Σχήμα 3.1) είναι κοινή και για τις τρεις λειτουργίες, οι επόμενες ενότητες όμως εστιάζουν ειδικά στο πως αυτή εφαρμόζεται στην κύρια ροή διδασκαλίας.

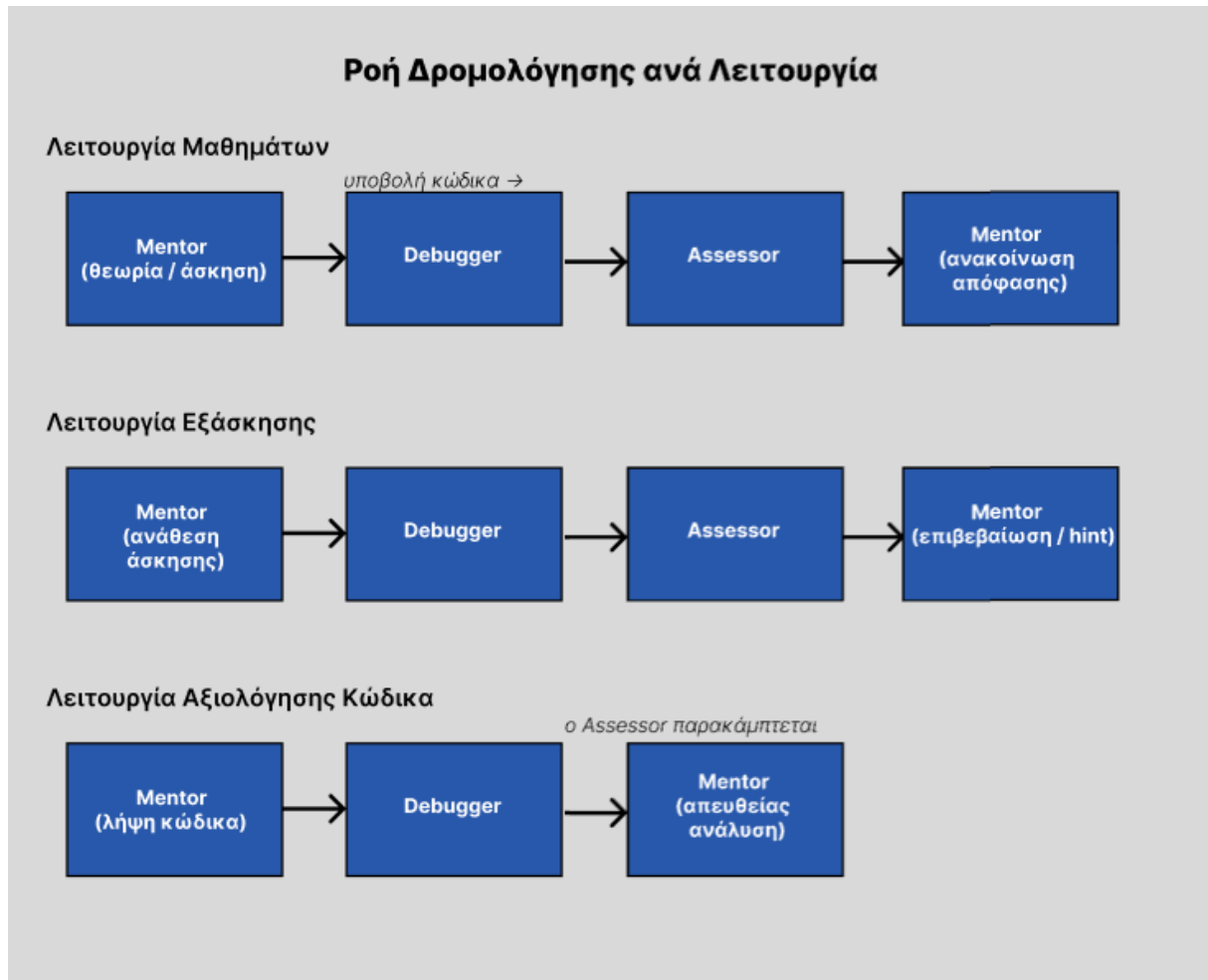
Το σύστημα αποτελείται από τρία βασικά αρχιτεκτονικά επίπεδα. Ένα επίπεδο διεπαφής, μέσω του οποίου ο φοιτητής αλληλεπιδρά με το σύστημα διαβάζοντας θεωρία, γράφοντας κώδικα, λαμβάνοντας υποδείξεις και ανατροφοδότηση. Ένα επίπεδο ενορχήστρωσης, το οποίο φιλοξενεί τον πυρήνα του συστήματος, το πολυπρακτορικό γράφημα κατάστασης (LangGraph), καθώς και συμπληρωματική λογική που ρυθμίζει παραμέτρους της αλληλεπίδρασης χωρίς να περνά μέσα από αυτό το γράφημα, και ένα επίπεδο αποθήκευσης, το οποίο διατηρεί το ιστορικό συνομιλίας και το προφίλ κάθε φοιτητή, ώστε το σύστημα να θυμάται την πρόοδο του ανάμεσα σε διαφορετικές συνεδρίες.



Σχήμα 3.1. Αρχιτεκτονικά επίπεδα του συστήματος

Ο πυρήνας του συστήματος υλοποιείται ως γράφημα κατάστασης (state machine), όπου κάθε πράκτορας αποτελεί έναν κόμβο γραφήματος. Οι τρεις πράκτορες μοιράζονται μια κοινή, δομημένη κατάσταση (state), η οποία περιλαμβάνει πληροφορίες τεσσάρων ειδών. Αρχικά έχουμε το περιεχόμενο της αλληλεπίδρασης, όπως το ιστορικό συνομιλίας, ο κώδικας του φοιτητή, η τρέχουσα άσκηση και τα κριτήρια επιτυχίας της. Δεύτερον, υπάρχει το προφίλ και το επίπεδο εμπειρίας του φοιτητή. Τρίτον, οι μετρικές προόδου, ο αριθμός προσπαθειών, ο χρόνος που αφιερώθηκε, ο αριθμός υποδείξεων που δόθηκαν και ενδείξεις για την προσαρμογή της δυσκολίας. Τέλος, έχουμε τα ευρήματα και τις αποφάσεις κάθε πράκτορα, όπως αναφορές σφαλμάτων (Debugger) ή αποφάσεις προόδου (Assessor). Κάθε πράκτορας διαβάζει από αυτή την κοινή κατάσταση ότι χρειάζεται και την ενημερώνει με τα δικά του αποτελέσματα, χωρίς να απαιτείται απευθείας επικοινωνία μεταξύ τους.

Η ροή ανάμεσα στους πράκτορες (Σχήμα 3.2) καθορίζεται από ρητή, ντετερμινιστική λογική δρομολόγησης. Στη βασική λειτουργία (λειτουργία Μαθημάτων), σε μια τυπική αλληλεπίδραση, με την πρώτη σύνδεση του φοιτητή ο Mentoring Agent τον καλωσορίζει και ρωτά σχετικά με το επίπεδο εμπειρίας του, ενώ σε κάθε επόμενη σύνδεση παρουσιάζει αντ' αυτού μια σύντομη περίληψη της μέχρι τώρα προόδου του. Στη συνέχεια εκθέτει τη θεωρία και την άσκηση της τρέχουσας ενότητας. Όταν ο φοιτητής υποβάλει κώδικα, η ροή οδηγείται στον Debugging Agent για ανάλυση και έπειτα στον Assessment Agent, ο οποίος αποφασίζει αν ο φοιτητής προχωράει σε επόμενη ενότητα (advance), επαναλαμβάνει άλλη άσκηση στην ίδια (repeat), ή χρειάζεται υποστήριξη για την τρέχουσα άσκηση (support). Έπειτα η ροή επιστρέφει στον Mentoring Agent, ο οποίος αναλαμβάνει να επικοινωνήσει το αποτέλεσμα στον φοιτητή και να υλοποιήσει την κίνηση που του σύστησε ο Assessment Agent. Στη λειτουργία της Εξάσκησης, η ροή ακολουθεί ακριβώς την ίδια διαδρομή σε κάθε υποβολή κώδικα (Debugging Agent → Assessment Agent → Mentoring Agent), με μόνη διαφορά στον τρόπο με τον οποίο ο Mentor ερμηνεύει την κρίση του Assessor, περιορίζοντας το σε απλή επιβεβαίωση ή υπόδειξη, χωρίς να επηρεάζεται η πρόοδος του φοιτητή στο κομμάτι των μαθημάτων. Από την άλλη, στη λειτουργία της ελεύθερης Αξιολόγησης Κώδικα, ο Assessment Agent παρακάμπτεται εντελώς, και η ροή επιστρέφει απευθείας στον Mentoring Agent.



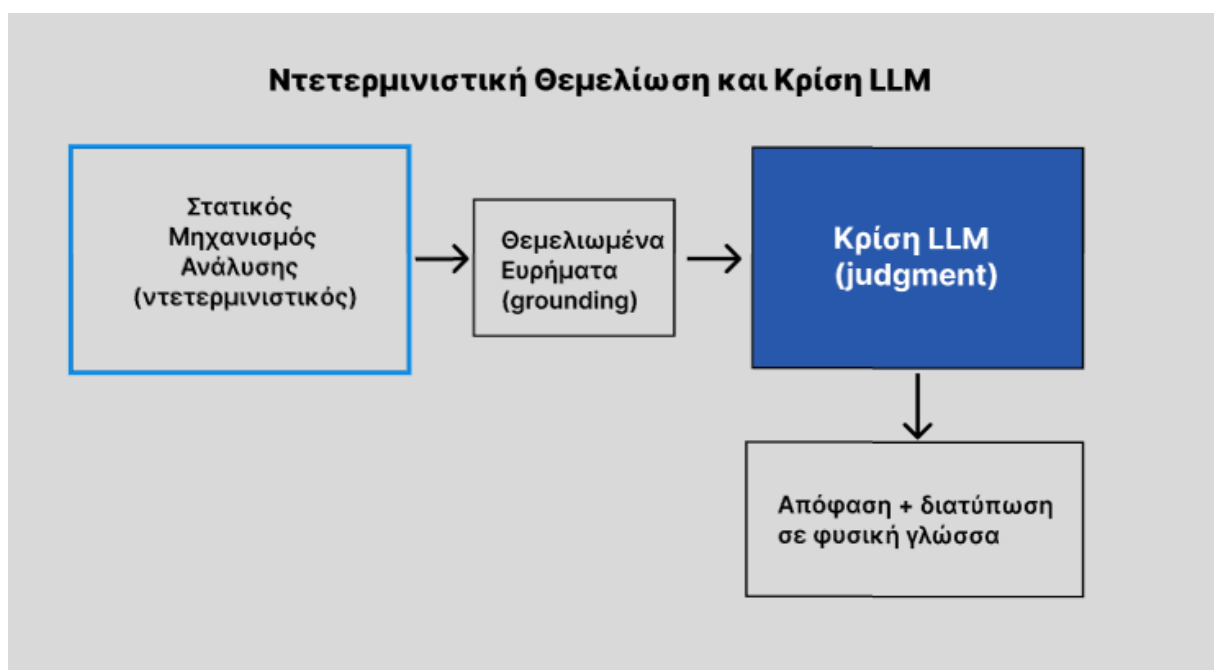
Σχήμα 3.2. Ροή δρομολόγησης ανά λειτουργία

Το εκπαιδευτικό περιεχόμενο στη λειτουργία των Μαθημάτων οργανώνεται σε πέντε ενότητες, η καθεμία αφιερωμένη σε ένα θεμελιώδες κομμάτι της Python. Συγκεκριμένα, οι ενότητες αυτές είναι: Μεταβλητές και Τύποι Δεδομένων, Δομές Ελέγχου, Λίστες, Επαναλήψεις, και Συναρτήσεις. Κάθε ενότητα διαθέτει δικό της θεωρητικό υλικό και δικά της πρότυπα ασκήσεων, σε δύο επίπεδα δυσκολίας, εύκολο και δύσκολο. Το επίπεδο δυσκολίας που βλέπει ο φοιτητής σε μια δεδομένη στιγμή δεν είναι σταθερό, προσαρμόζεται δυναμικά ανάλογα με την δήλωση του στην αρχική σύνδεση, αλλά και την επίδοσή του. Αυτή η δομή (θεωρία και άσκηση ανά ενότητα, σε δύο επίπεδα δυσκολίας η καθεμία), είναι θεμελιώδης για την κατανόηση του πώς λειτουργεί το σύστημα.

Συνολικά, η αρχιτεκτονική που υλοποιείται αντιστοιχεί στην κεντριοποιημένη οργάνωση πολυπρακτορικών συστημάτων που περιγράφηκε στην ενότητα 2.3 [8], όπου ένας μηχανισμός συντονισμού καθορίζει τη σειρά ενεργοποίησης των πρακτόρων, αντί αυτοί να αποφασίζουν αυτόνομα πότε θα δράσει ο καθένας. Παράλληλα, η κοινή και δομημένη κατάσταση (state) που μοιράζονται οι πράκτορες λειτουργεί ως ένα ανεξάρτητο επίπεδο αποθήκευσης γνώσης [10], επιτρέποντας την απρόσκοπτη ανταλλαγή δεδομένων μεταξύ τους χωρίς να απαιτείται άμεση ή άναρχη επικοινωνία.

3.3 Ντετερμινιστική Θεμελίωση και Κρίση LLM

Όπως παρουσιάστηκε στην ενότητα 2.6, ένα LLM δεν διαθέτει αποδοτικούς τρόπους να επαληθεύσει από μόνο του την ορθότητα των απαντήσεων που παράγει, για αυτό και χρειάζεται κάποιον εξωτερικό έλεγχο για να είναι αξιόπιστο. Στην παρούσα εργασία υιοθετείται το εξής μοτίβο σχεδίασης (Σχήμα 3.3) για να λυθεί αυτό το πρόβλημα: πρώτα, ένας στατικός μηχανισμός αναλύει τα δεδομένα και βγάζει συγκεκριμένα, ντετερμινιστικά επαληθευμένα ευρήματα (grounding). Τα δεδομένα αυτά περιλαμβάνουν πράγματα όπως το ποιες δομές χρησιμοποιεί ο κώδικας ενός φοιτητή, ή αν ταιριάζουν οι λύσεις των ασκήσεων με όσα ζητούνται στην εκφώνηση. Τέτοια ευρήματα συντελούν την δομημένη γνώση στην οποία αναφερθήκαμε ήδη ως τρόπο περιορισμού της αναξιοπιστίας των LLM. Μετά, το LLM χρησιμοποιεί μόνο αυτά τα ευρήματα για να καταλήξει στην τελική απόφαση και να τη διατυπώσει σε φυσική γλώσσα προς τον φοιτητή (judgment).



Σχήμα 3.3. Ντετερμινιστική θεμελίωση και κρίση LLM

Η ντετερμινιστική θεμελίωση δεν αφαιρεί την αυτονομία του LLM, αλλά αλλάζει την ποιότητα της πληροφορίας πάνω στην οποία στηρίζει τις αποφάσεις του. Το συμπέρασμα παραμένει δικό του, κανείς δεν του υπαγορεύει τι να κρίνει. Αυτό που εξασφαλίζεται είναι ότι κρίνει πάνω σε επαληθευμένα δεδομένα και όχι σε δικές του εικασίες για την κατάσταση του κώδικα. Ο συνδυασμός αυτός αντικατοπτρίζει μία από τις τέσσερις ιδιότητες των πρακτόρων που αναφέρθηκαν, την αντιδραστικότητα, δηλαδή την ικανότητα αντίληψης του περιβάλλοντος και ανταπόκρισης σε αυτό [4]. Τα ευρήματα είναι ακριβώς αυτή η αντίληψη, σε μια μορφή που δεν εξαρτάται από υποθέσεις.

Αυτή η τακτική εφαρμόζεται με συνέπεια και στους τρεις πράκτορες του συστήματος. Στον Debugging Agent για τον εντοπισμό σφαλμάτων, στον Assessment Agent για την αξιολόγηση της προόδου, και στον Mentoring Agent ακόμα και για την απάντηση θεωρητικών ερωτήσεων. Η συγκεκριμένη εφαρμογή της σε καθέναν εξηγείται αναλυτικότερα στις ενότητες 3.4 έως 3.6.

Το ίδιο μοτίβο βρίσκει επιβεβαίωση και εκτός της δικής μας εργασίας [13], [14]. Σε τεχνικά πεδία, όπου η ορθότητα δεν είναι θέμα πειστικότητας αλλά συγκεκριμένων περιορισμών, φαίνεται ακόμα πιο καθαρά πόσο σημαντικό είναι. Σε σύστημα καθοδήγησης ηλεκτρονικών ισχύος για παράδειγμα, η προσθήκη ρητού επιπέδου μηχανικής επαλήθευσης πάνω από την απλή ανάκτηση υλικού (RAG) ανέβασε το ποσοστό επιτυχίας των λύσεων από 29-58% σε 98.5% [18]. Παρόμοια, σε προβλήματα μετάφρασης φυσικής γλώσσας σε τυπική λογική, ένας συμβολικός μηχανισμός κατάφερε να εντοπίζει τις παραισθήσεις του μοντέλου και να του τις στέλνει πίσω για διόρθωση [21].

Αυτή η λογική διάκρισης ανάμεσα σε ότι πρέπει να μείνει προβλέψιμο και σε ότι μπορεί να παραμείνει ευέλικτο, συνδέεται άμεσα με το bounded determinism [9]. Πρακτικά, οι έλεγχοι ορθότητας και τα δομικά δεδομένα παραμένουν αυστηρά αιτιοκρατικά και ελέγξιμα, ενώ η φυσική γλώσσα και η προσαρμοστική διατύπωση ανατίθενται στην ευελιξία του LLM. Έτσι το σύστημα αποφεύγει τόσο τους περιορισμούς ενός καθαρά συμβολικού συστήματος, όπως τα κλασικά ITS, όσο και τους κινδύνους ενός συστήματος που βασίζεται μόνο σε LLM. Στην πράξη, αυτό το μοτίβο αποτελεί την υλοποίηση του δεύτερου ερευνητικού στόχου που τέθηκε, την αντικατάσταση της ελεύθερης παραγωγής κειμένου του LLM με αυστηρή, δομημένη ανάλυση όπου αυτό είναι εφικτό.

3.4 Mentoring Agent

Ο Mentor είναι ο πράκτορας που βρίσκεται σε άμεση επαφή με τον φοιτητή σε όλη τη διάρκεια της αλληλεπίδρασης, παρουσιάζει τη θεωρία κάθε ενότητας, αναθέτει ασκήσεις, απαντά σε ερωτήσεις και μεταφράζει τα ευρήματα του Debugger και του Assessor σε ανθρώπινα κατανοητή καθοδήγηση. Ενώ οι δύο άλλοι πράκτορες παράγουν δεδομένα για τον κώδικα του φοιτητή, ο Mentor είναι αυτός που τα μετατρέπει σε παιδαγωγικό λόγο, και επομένως ο πράκτορας στον οποίο η ισορροπία ανάμεσα σε ντετερμινιστική θεμελίωση και κρίση του LLM γίνεται πιο εμφανής στην πράξη, σε τρεις διαφορετικές λειτουργίες του.

3.4.1 Παρουσίαση περιεχομένου και ανάθεση ασκήσεων

Η πρώτη λειτουργία αφορά την παρουσίαση περιεχομένου στον φοιτητή. Το περιεχόμενο αυτό περιλαμβάνει τόσο τις θεωρίες της εκάστοτε ενότητας, όσο και τις ασκήσεις τους. Όταν πρέπει να ανατεθεί μια νέα άσκηση, ο Mentor δεν ζητά από το LLM να τη συνθέσει από την αρχή, ούτε χρησιμοποιεί κάποια στατική εκφώνηση ανά ενότητα. Χρησιμοποιώντας τη συνάρτηση `generate_random_task` επιλέγει τυχαία ένα πρότυπο εκφώνησης (`task_templates`) ανάμεσα σε όσα αντιστοιχούν στο τρέχον μάθημα και επίπεδο δυσκολίας, αποφεύγοντας σκόπιμα να δώσει ξανά το ίδιο πρότυπο με την αμέσως προηγούμενη άσκηση. Το πρότυπο περιέχει σταθερά κενά (`placeholders`), τα οποία συμπληρώνονται με τυχαία επιλεγμένες τιμές από ένα προκαθορισμένο σύνολο πιθανών τιμών (`possible_values`), ειδικό για κάθε μάθημα. Τα κριτήρια επιτυχίας της άσκησης περνούν από την ίδια ακριβώς αντικατάσταση τιμών, ώστε να παραμένουν συνεπή με τη συγκεκριμένη εκδοχή που παρήχθη (Παράρτημα Α, Σχήμα Α.2). Η ίδια λογική χρησιμοποιείται και στην απάντηση σε θεωρητικές ερωτήσεις. Όταν ο φοιτητής θέτει μια ερώτηση πάνω στο περιεχόμενο ενός μαθήματος, ο Mentor δεν αφήνει το LLM να απαντήσει εντελώς ελεύθερα. Φιλτράρει πρώτα ντετερμινιστικά ποιο μέρος της ύλης έχει ήδη διδαχθεί, με βάση το `"current_lesson_id"` του φοιτητή, και δίνει στο LLM μόνο αυτό το υποσύνολο ως βάση γνώσης, μαζί με ρητή οδηγία να μην αποκαλύψει τίποτα από επόμενα μαθήματα. Το LLM δεν αποφασίζει τι επιτρέπεται να πει, αποφασίζει μόνο πώς να το διατυπώσει καλύτερα μέσα σε αυτό το προκαθορισμένο περιθώριο.

Πριν από όλα αυτά, το σύστημα χρειάζεται ένα σημείο εκκίνησης. Με την πρώτη του σύνδεση, ο Mentor ρωτά τον φοιτητή αν έχει ξαναγράψει κώδικα ή αν πρόκειται για την πρώτη του επαφή με προγραμματισμό. Η απάντηση δίνεται σε ελεύθερη φυσική γλώσσα και ταξινομείται από το LLM σε αρχάριο ή έμπειρο επίπεδο. Όταν η απάντηση δεν είναι ξεκάθαρη, το σύστημα δεν ρισκάρει, ξεκινά τον φοιτητή από το βασικό επίπεδο, εξηγώντας του παράλληλα ότι αν αποδειχθεί πως ήδη τα γνωρίζει, το επίπεδο του θα ανέβει γρήγορα.

Αυτή η αρχική δήλωση δεν παραμένει απλή ετικέτα στο προφίλ του φοιτητή. Καθορίζει το επίπεδο δυσκολίας, εύκολο ή δύσκολο, με το οποίο ξεκινά, το οποίο με τη σειρά του επηρεάζει δύο ξεχωριστά πράγματα. Πρώτον, ποια πρότυπα ασκήσεων επιλέγει η `generate_random_task`, όπως περιγράφηκε παραπάνω. Δεύτερον, το ύφος με το οποίο ο Mentor παρουσιάζει τη θεωρία της κάθε ενότητας. Σε φοιτητή με χαμηλότερο επίπεδο δίνονται περισσότερα παραδείγματα και απλούστερη διατύπωση, ενώ σε φοιτητή με υψηλότερο επίπεδο δίνεται πιο σύντομη εξήγηση με τεχνική ορολογία. Η ίδια αρχική δήλωση διαμορφώνει δηλαδή τόσο τι ζητείται από τον φοιτητή όσο και το πώς του παρουσιάζεται η ύλη.

Η επιλογή να μη μείνει το σύστημα ουδέτερο ως προς το επίπεδο του φοιτητή από την πρώτη κιόλας αλληλεπίδραση δεν είναι απλά τεχνικής φύσης. Τόσο μια άσκηση πολύ εύκολη για κάποιον με ήδη υπόβαθρο όσο και μια άσκηση πολύ απαιτητική για κάποιον χωρίς καθόλου εμπειρία συνδέονται με πλήξη, η οποία προηγείται της αποδέσμευσης του μαθητή από τη διαδικασία. Η ίδια έρευνα, που αναφέρεται αναλυτικότερα στην επόμενη υποενότητα, δείχνει ότι η πλήξη σε Έξυπνα Συστήματα Διδασκαλίας εξαρτάται περισσότερο από την ίδια την άσκηση παρά από τον μαθητή που την επιχειρεί [42], γεγονός που καθιστά τη ρύθμιση της δυσκολίας σχεδιαστική ευθύνη του συστήματος. Πρόκειται δηλαδή για την ίδια λογική προσαρμογής στην οποία στηρίζεται και η κλιμάκωση των υποδείξεων (3.4.2), εφαρμοσμένη όμως στο σημείο εκκίνησης αντί μέσα στην πορεία μιας άσκησης. Και στις δύο κατευθύνσεις το αρχικό επίπεδο παραμένει σημείο εκκίνησης, όχι μόνιμος χαρακτηρισμός, όπως δείχνει η προσαρμογή που ακολουθεί.

Αξίζει να σημειωθεί πως το επίπεδο δυσκολίας που δέχεται η `generate_random_task` δεν είναι σταθερό, ούτε αποφασίζεται από το LLM. Προκύπτει από μια ξεχωριστή, εξίσου ντετερμινιστική λογική, η οποία παρακολουθεί την πραγματική επίδοση του φοιτητή στον χρόνο και τον δοκιμάζει προσωρινά σε αντίθετο επίπεδο πριν αλλάξει μόνιμα τον χαρακτηρισμό του, προς τα πάνω αν δείχνει έτοιμος για κάτι πιο απαιτητικό, προς τα κάτω αν δυσκολεύεται επαναλαμβανόμενα. Ο ακριβής μηχανισμός αυτής της δοκιμαστικής αλλαγής περιγράφεται στην ενότητα 4.8.2. Σε κάθε περίπτωση ο Mentor ενημερώνει σχετικά τον φοιτητή, έτσι ώστε η προσαρμογή να διατηρεί το επικοινωνιακό κομμάτι της εκπαίδευσης.

3.4.2 Παραγωγή και κλιμάκωση υποδείξεων

Η δεύτερη, και πιο σύνθετη, λειτουργία, εμφανίζεται στη δημιουργία υποδείξεων (hints) οι οποίες παράγονται όταν ο φοιτητής δυσκολεύεται σε μια άσκηση. Ο Mentor δεν λαμβάνει ποτέ τον ίδιο τον κώδικα του φοιτητή. Λαμβάνει όμως, από την ίδια στατική ανάλυση που τροφοδοτεί τους άλλους δύο πράκτορες, μια ρητή λίστα των δομών που υπάρχουν πραγματικά σε αυτόν και εκείνων που απουσιάζουν, μαζί με ρητή απαγόρευση να αναφερθεί σε δομή που δηλώνεται ως ανύπαρκτη. Ο λόγος αυτής της επιλογής προκύπτει από τη φύση της εργασίας του. Ο Mentor διατυπώνει την υπόδειξη σε φυσική γλώσσα, και ένα μοντέλο που καλείται να μιλήσει για κώδικα τον οποίο δεν έχει δει θα συμπληρώσει το κενό με εικασίες. Χωρίς αυτή τη λίστα, μια υπόδειξη μπορεί να αναφερθεί στη λογική ενός βρόχου σε πρόγραμμα που δεν περιέχει βρόχο. Η θεμελίωση εδώ δεν καθορίζει τι θα

αποφασιστεί, καθορίζει για ποια δομή επιτρέπεται να γίνει λόγος. Πρόκειται για την αρχή της ενότητας 3.3 εφαρμοσμένη στον τρίτο πράκτορα, σε ότι του αντιστοιχεί, δηλαδή όχι στην κρίση αλλά στη διατύπωση.

Πέρα από αυτή τη λίστα, ο Mentor έχει πρόσβαση μέσα από το κοινό state τόσο στο τεχνικό εύρημα του Debugger όσο και στην ολοκληρωμένη κρίση του Assessor, και χρησιμοποιεί όποιο από τα δύο είναι πραγματικά χρήσιμο σε κάθε περίπτωση όπου απαιτείται καθοδήγηση. Συγκεκριμένα, στην περίπτωση που ο κώδικας του φοιτητή δεν είναι καν συντακτικά έγκυρος, δεν υπάρχει τίποτα να ερμηνευτεί, οπότε το πιο ακριβές εύρημα είναι αυτό του Debugger. Ο Mentor λοιπόν το χρησιμοποιεί απευθείας, και αντί να καλέσει το LLM επιλέγει μια προδιατυπωμένη παιδαγωγική εξήγηση που αντιστοιχεί στο είδος του σφάλματος. Σε κάθε άλλη περίπτωση, που ο κώδικας είναι τεχνικά σωστός αλλά δεν ικανοποιεί τα κριτήρια της άσκησης, ο Mentor κάνει τη δική του κλήση LLM, μέσα από την οποία αξιοποιεί την κρίση του Assessor και αποφασίζει πώς θα επικοινωνηθούν κατάλληλα τα ευρήματα του. Ο φοιτητής, σε καμία περίπτωση δεν βλέπει φυσικά τις λέξεις «Debugger» ή «Assessor». Ο Mentor του μιλά σαν να βλέπει ο ίδιος τον κώδικα, διατηρώντας την εντύπωση ενός ενιαίου συνομιλητή παρότι η ανίχνευση, η αξιολόγηση και η διατύπωση προέρχονται από τρία διαφορετικά, διαδοχικά βήματα. Αν, τέλος, κάποια κλήση LLM στην αλυσίδα αποτύχει τεχνικά, το σύστημα δεν αφήνει τον φοιτητή χωρίς απάντηση. Υπάρχει ντετερμινιστικό εναλλακτικό κείμενο, έτοιμο να χρησιμοποιηθεί ως δίχτυ ασφαλείας.

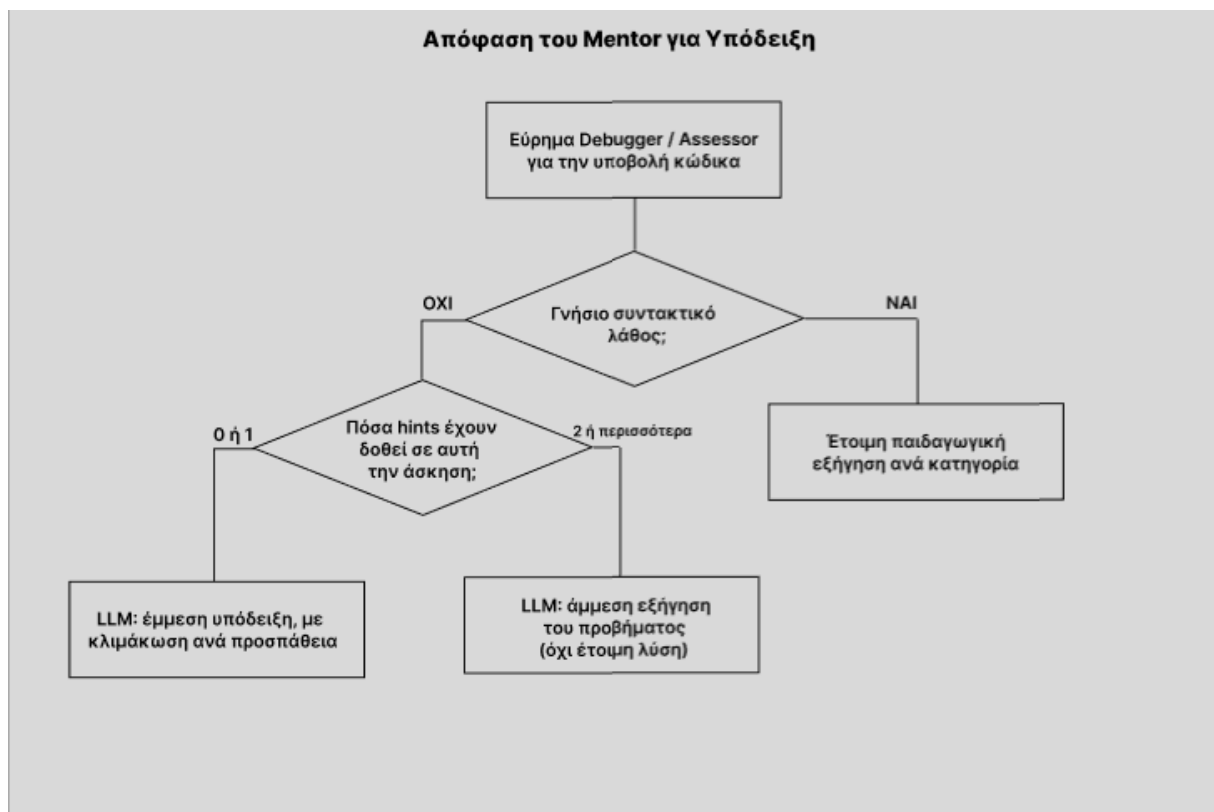
Ο τρόπος με τον οποίο διατυπώνεται η υπόδειξη του LLM δεν αφήνεται εντελώς ελεύθερος. Το prompt απαγορεύει ρητά στο LLM να δώσει τη λύση ή να γράψει κώδικα. Η υπόδειξη πρέπει να κατευθύνει τον φοιτητή προς τη σωστή σκέψη, χωρίς να του δώσει την απάντηση. Η επιλογή αυτή ακολουθεί τη λογική της Σωκρατικής μεθόδου διδασκαλίας, όπου ο δάσκαλος καθοδηγεί τον μαθητή μέσω ερωτήσεων και υπαινιγμών ώστε να φτάσει μόνος του στη λύση, αντί να του τη δώσει έτοιμη [31]. Στη βιβλιογραφία γύρω από LLMs που λειτουργούν ως Σωκρατικοί δάσκαλοι, μια υπόδειξη που αποκαλύπτει το σφάλμα πολύ νωρίς θεωρείται ρητά κακή πρακτική, ακριβώς επειδή στερεί από τον μαθητή την ευκαιρία να σκεφτεί μόνος του [31]. Ο Mentor εφαρμόζει την αρχή αυτή μέσα από συγκεκριμένο κανόνα στο prompt, όχι μέσα από κάποια γενική οδηγία σχετικά με τη Σωκρατική θεωρία, κάτι που θα άφηνε πολύ μεγαλύτερο περιθώριο στο LLM να αποφασίσει μόνο του πόσο άμεσο να είναι.

Η υποστήριξη όμως δεν μένει σταθερή όσο ο φοιτητής συνεχίζει να δυσκολεύεται στην ίδια άσκηση. Το σύστημα μετράει πόσα hints έχουν ήδη δοθεί (hint_count) και προσαρμόζει ανάλογα το επόμενο, ώστε να μη φαίνεται στον φοιτητή πως ξαναπαίρνει την ίδια, πανομοιότυπη υπόδειξη. Η λογική του να προσφέρεται περισσότερη υποστήριξη ακριβώς εκεί όπου ο φοιτητής τη χρειάζεται, αντιστοιχεί στην έννοια του scaffolding (πλαίσιο σταδιακής υποστήριξης) από τη θεωρία της Zone of Proximal Development (ζώνη επικείμενης ανάπτυξης) του Vygotsky: η υποστήριξη που παρέχεται σε έναν μαθητή δεν είναι σταθερή, αλλά προσαρμόζεται δυναμικά ανάλογα με το πόσο κοντά βρίσκεται στη λύση, αυξάνεται όταν χρειάζεται περισσότερη βοήθεια και μειώνεται όταν ο μαθητής δείχνει ότι μπορεί να προχωρήσει μόνος του [32]. Η ίδια λογική διαφοροποίησης εφαρμόζεται και σε επίπεδο μαθητή συνολικά: το σύστημα κρατά ιστορικό μέσου όρου hints ανά άσκηση για κάθε φοιτητή (avg_hints_per_task), και όσοι έχουν ιστορικό ανάγκης για περισσότερη καθοδήγηση λαμβάνουν εξαρχής πιο ξεκάθαρη κατεύθυνση, ενώ όσοι συνήθως αρκούνται σε μία υπόδειξη λαμβάνουν πιο γενικά, σύντομα hints.

Η σταδιακή απόσυρση της υποστήριξης καθώς ο μαθητής προοδεύει, γνωστή ως fading, έχει προταθεί στη βιβλιογραφία ως διαδικασία που θα έπρεπε να προσαρμόζεται στο προφίλ του κάθε μαθητή αντί

να εφαρμόζεται με ενιαίο ρυθμό για όλους [39]. Η διαφοροποίηση με βάση τον μέσο όρο υποδείξεων ανά άσκηση κινείται σε αυτή τη λογική.

Το όριο επιμονής στην έμμεση καθοδήγηση ακολουθεί και αυτό μια σταθερή λογική. Μετά από δύο συνεχόμενες ανεπιτυχείς έμμεσες υποδείξεις για την ίδια άσκηση, ο Mentor εγκαταλείπει επίτηδες τη Σωκρατική προσέγγιση και εξηγεί άμεσα ποιο είναι το πρόβλημα και γιατί συμβαίνει, χωρίς όμως να γράφει διορθωμένο κώδικα, αφήνοντας το φοιτητή να είναι αυτός που το διορθώνει. Η επιλογή αυτή δεν είναι αυθαίρετη. Πρόσφατη αξιολόγηση συστήματος διδασκαλίας βασισμένου σε LLM που ακολουθούσε αυστηρά Σωκρατική προσέγγιση, χωρίς καμία δυνατότητα αλλαγής στρατηγικής, εντόπισε ακριβώς αυτό το πρόβλημα. Όταν ο φοιτητής δεν διαθέτει το υπόβαθρο να προχωρήσει μέσα από ερωτήσεις, η επίμονη Σωκρατική καθοδήγηση δεν βοηθά αλλά οδηγεί σε αδιέξοδο διάλογο και απογοήτευση. Οι ερευνητές κατέληξαν ότι μια αποτελεσματική εκπαιδευτική εφαρμογή βασισμένη σε LLM χρειάζεται προσαρμοστική στρατηγική που μεταβαίνει σε πιο ρητή καθοδήγηση όταν ο μαθητής δεν προχωράει [33]. Το κατώφλι αυτό στον Mentor είναι μια απλή, ντετερμινιστική εκδοχή ακριβώς αυτής της αρχής. Δεν αφήνεται στο LLM να αποφασίσει μόνο του πότε να «παραιτηθεί» από τη Σωκρατική μέθοδο. Η αλλαγή στρατηγικής ενεργοποιείται αυτόματα, στη βάση μιας μετρήσιμης ένδειξης πραγματικής δυσκολίας (Σχήμα 3.4).



Σχήμα 3.4. Απόφαση του Mentor για υπόδειξη

Αν περάσει αρκετή ώρα αδράνειας πάνω στην άσκηση, ο Mentor δεν περιμένει να τον ρωτήσει κάτι ο φοιτητής, αλλά παίρνει την πρωτοβουλία να δώσει ενθάρρυνση ή, αν η αδράνεια συνεχιστεί, μια πρώτη γενική υπόδειξη, ακολουθώντας την ίδια λογική κλιμάκωσης με τα κανονικά hints. Τα χρονικά όρια δεν είναι κοινά για όλους. Ένας αρχάριος χρειάζεται περισσότερο χρόνο πριν η σιωπή του

σημάνει πραγματική δυσκολία, ενώ η επίμονη καθοδήγηση προς φοιτητή που δεν έχει ακόμη το απαραίτητο υπόβαθρο επιβάλλει υπερβολικό γνωστικό φόρτο και μπορεί να οδηγήσει σε απογοήτευση [33]. Η παρέμβαση όμως δεν μπορεί ούτε να καθυστερεί υπερβολικά, καθώς η μη αξιοποίηση των διαθέσιμων μηχανισμών βοήθειας αποτελεί τεκμηριωμένη μορφή μη παραγωγικής συμπεριφοράς που περιορίζει τα μαθησιακά οφέλη [40].

Η απόφαση του συστήματος να παρεμβαίνει αυτόνομα, χωρίς να περιμένει ρητό αίτημα από τον φοιτητή, βασίζεται σε ένα τεκμηριωμένο πρόβλημα των Έξυπνων Συστημάτων Διδασκαλίας. Η χρήση των διαθέσιμων μηχανισμών βοήθειας από τους μαθητές είναι συχνά μη παραγωγική, με δύο αντίθετες μορφές, την υπερβολική χρήση και την αποφυγή, όπου περιορίζουν τα μαθησιακά οφέλη [40]. Το σύστημα απαντά και στις δύο. Στην αποφυγή απαντά ο χρονιστής αδράνειας, ενώ στην υπερβολική χρήση απαντά η κλιμάκωση των υποδείξεων και η διαφοροποίηση με βάση το ιστορικό του φοιτητή.

Το γεγονός ότι ο Mentor δεν αποφασίζει, αλλά μεταφράζει μια απόφαση που έχει ήδη ληφθεί εμφανίζεται και όταν ο φοιτητής θέλει να προχωρήσει στην επόμενη ενότητα. Αν η τελευταία απόφαση του Assessor για την τρέχουσα άσκηση ήταν «repeat» ή «support» και ο φοιτητής ζητήσει παρ' όλα αυτά να προχωρήσει, ο Mentor δεν τον μπλοκάρει σιωπηλά ούτε αποφασίζει μόνος του αν είναι έτοιμος. Αντίθετα, εξηγεί ευγενικά γιατί η τρέχουσα ενότητα χρειάζεται ακόμα δουλειά και προτείνει εναλλακτικές, αντί να επιβάλει απλώς άρνηση. Η απόφαση προόδου παραμένει του Assessor, ο Mentor διαχειρίζεται μόνο πώς αυτή η απόφαση θα ανακοινωθεί στον φοιτητή.

3.4.3 Συμπληρωματικές λειτουργίες

Η υποδομή της δεύτερης λειτουργίας (ευρήματα Debugger και Assessor μεταφρασμένα σε hint μέσω LLM) χρησιμοποιείται και στο κομμάτι της Εξάσκησης, αλλά με διαφορετικό σκοπό. Όταν ο φοιτητής λύσει σωστά μια άσκηση στα πλαίσια της Εξάσκησης, ο Mentor δίνει απλώς μια σύντομη, ενθαρρυντική επιβεβαίωση, ενώ όταν κάνει λάθος ενεργοποιείται ακριβώς η ίδια διαδικασία hint με την κύρια ροή. Ο ίδιος μηχανισμός, δηλαδή, μπορεί να εξυπηρετήσει είτε αξιολόγηση με συνέπειες για την πρόοδο του φοιτητή είτε ελεύθερη εξάσκηση χωρίς συνέπειες, ανάλογα με το πλαίσιο μέσα στο οποίο καλείται.

Η δυνατότητα Εξάσκησης δεν είναι διαθέσιμη σε όλες τις ενότητες αδιακρίτως. Ένας φοιτητής μπορεί να ζητήσει άσκηση εξάσκησης μόνο σε ενότητες που έχει ήδη ολοκληρώσει, όχι στην τρέχουσα ή σε μελλοντικές. Ο περιορισμός αυτός δεν επιβάλλεται μόνο στο περιβάλλον χρήσης. Το ίδιο το endpoint που παράγει την άσκηση εξάσκησης ελέγχει ξανά, ανεξάρτητα, ποιες ενότητες έχει πράγματι ολοκληρώσει ο συγκεκριμένος φοιτητής στη βάση δεδομένων, και απορρίπτει οποιοδήποτε αίτημα για ενότητα που δεν πληροί αυτή την προϋπόθεση.

Η τρίτη λειτουργία του Mentor συναντάται στο τρίτο κουμπί, την Αξιολόγηση Κώδικα, όπου η ροή είναι διαφορετική από τα άλλα δύο. Εδώ ο φοιτητής ανεβάζει δικό του κώδικα εκτός άσκησης, δεν υπάρχει κριτήριο επιτυχίας να ελεγχθεί, και ο Assessor παρακάμπτεται εντελώς, αφήνοντας το Mentor να απαντά απευθείας με βάση το εύρημα του Debugger. Η απουσία βαθμολογούμενης άσκησης αλλάζει και τον τρόπο απάντησης. Αφού ο φοιτητής ζητά ρητά ανάλυση και όχι hint, η Σωκρατική απόκρυψη της λύσης δεν έχει πια λόγο ύπαρξης. Το prompt εδώ επιτρέπει ρητά στο LLM να είναι συγκεκριμένο για το πρόβλημα, σε αντίθεση με τα hint της ροής των ασκήσεων. Η λογική του γνήσιου συντακτικού σφάλματος παραμένει ίδια με πριν, όπως και η περίπτωση που ο κώδικας είναι τεχνικά ορθός, όπου ο Mentor απλώς το επιβεβαιώνει σύντομα. Και εδώ, αν η κλήση LLM αποτύχει τεχνικά, υπάρχει ντετερμινιστικό εναλλακτικό κείμενο.

Σε κάθε μία από αυτές τις λειτουργίες, ο Mentor δεν χρησιμοποιείται ούτε ως γενικός συνομιλητής που αποφασίζει ελεύθερα τι να πει, ούτε ως ανιχνευτής ή αξιολογητής σφαλμάτων. Η ανίχνευση γίνεται στον Debugger και η αξιολόγηση ως προς τα κριτήρια της συγκεκριμένης άσκησης και η απόφαση για την πρόοδο του φοιτητή στον Assessor. Ο Mentor παραλαμβάνει το τελικό αποτέλεσμα αυτής της αλυσίδας έτοιμο, και η δική του κρίση LLM αφορά αποκλειστικά πώς θα διατυπωθεί και θα κλιμακωθεί η παιδαγωγική απάντηση.

3.5 Debugging Agent

Ο Debugger είναι ο πρώτος πράκτορας που επεξεργάζεται τον κώδικα του φοιτητή σε κάθε ροή εξέτασης, πριν ο Assessor τον κρίνει ως προς τα κριτήρια της συγκεκριμένης άσκησης. Ο ρόλος του είναι στενά ορισμένος. Εντοπίζει τεχνικά προβλήματα, όπως ένα σφάλμα σύνταξης, μια δομή που λείπει, μια μεταβλητή που χρησιμοποιείται πριν οριστεί. Λαμβάνει την εκφώνηση και τα κριτήρια της άσκησης, αλλά μόνο για να ξέρει ποιες δομές αναμένονται να υπάρχουν στον κώδικα. Δεν τα χρησιμοποιεί ως πρότυπο βαθμολόγησης και δεν αποφαινεται αν η άσκηση λύθηκε. Η κρίση αυτή ανήκει αποκλειστικά στον Assessor. Ο Debugger απαντά σε ένα μόνο ερώτημα, αν ο κώδικας είναι τεχνικά υγιής.

Η ανάλυση ξεκινά με έναν έλεγχο για το αν ο κώδικας είναι έγκυρη Python. Το πρώτο πράγμα που κάνει ο Debugger δεν είναι να καταλάβει τι κάνει ο εκάστοτε κώδικας, αλλά να ελέγξει αν είναι γραμματικά σωστός, δηλαδή αν η Python μπορεί να τον “διαβάσει”, ανεξάρτητα από το αν επιτυγχάνει τελικά αυτό που ζητά η άσκηση. Αυτό το κάνει χρησιμοποιώντας τη συνάρτηση `ast.parse`, η οποία αποτελεί μέρος της ίδιας της Python. Κάθε φορά που η Python τρέχει οποιονδήποτε κώδικα, πρώτα τον μετατρέπει εσωτερικά σε AST (Abstract Syntax Tree). Πρόκειται για μια δενδρική αναπαράσταση της γραμματικής δομής του, όπου κάθε κόμβος αντιστοιχεί σε ένα κομμάτι του προγράμματος, όπως μια εντολή, μια συνθήκη, μια κλήση συνάρτησης. Αυτό το βήμα είναι απαραίτητο πριν αρχίσει να εκτελεί οτιδήποτε, γιατί πρέπει πρώτα να “καταλάβει” τη δομή του κειμένου. Η `ast.parse` δίνει πρόσβαση ακριβώς σε αυτό το ενδιάμεσο βήμα. Δέχεται τον κώδικα ως απλό κείμενο και επιστρέφει είτε το δέντρο, αν ο κώδικας είναι γραμματικά έγκυρος, είτε μια εξαίρεση με δικό της μήνυμα και αριθμό γραμμής, αν δεν είναι, χωρίς να προχωρήσει ποτέ στην εκτέλεση. Ο Debugger χρησιμοποιεί αυτή τη συνάρτηση πάνω στον κώδικα του φοιτητή.

Αν αυτός ο έλεγχος αποτύχει σημαίνει πως ο κώδικας έχει γνήσιο συντακτικό λάθος και δεν γίνεται να διαβαστεί. Δεν χρειάζεται καμία κρίση για το αν υπάρχει πρόβλημα ή πού ακριβώς βρίσκεται, η Python το εντοπίζει με σαφήνεια, χωρίς το περιθώριο ερμηνείας που θα υπήρχε σε ένα λογικό λάθος, όπου δεν είναι δεδομένο τι πραγματικά συμβαίνει. Η ακριβής διατύπωση του μηνύματος από τον parser, ωστόσο, δεν είναι πάντα εξίσου σαφής. Γι' αυτό ο Debugger, χωρίς να καλεί καθόλου LLM σε αυτή την περίπτωση, απλώς καταγράφει τι βρήκε.

Αυτή η καταγραφή γίνεται με δύο τρόπους. Ο πρώτος αφορά δύο πολύ συνηθισμένα λάθη αρχαρίων, για τα οποία η συνάρτηση `ast.parse` έχει προεπιλεγμένα μηνύματα λάθους, τα οποία μπορεί να αποβούν παραπλανητικά. Το ένα, είναι η χρήση σύνταξης `if/else` από άλλη γλώσσα προγραμματισμού αντί της σωστής δομής της Python. Σε αυτή την περίπτωση, ο parser επιστρέφει «expected '»», ανεξάρτητα από την πραγματική μορφή του λάθους. Το άλλο είναι η χρήση αριθμητικής τιμής στη θέση ονόματος παραμέτρου κατά τον ορισμό μιας συνάρτησης, όπου επιστρέφει «invalid syntax», χωρίς καμία ένδειξη του τι ακριβώς πάει στραβά. Εφόσον τα μηνύματα του parser δεν αρκούν από μόνα τους σε αυτές τις περιπτώσεις, ο Debugger δεν βασίζεται απλά σε αυτά, αλλά ψάχνει το ακριβές

μοτίβο απευθείας μέσα στο κείμενο του κώδικα του φοιτητή. Αν και δεν κατανοεί το νόημα του μοτίβου, αναγνωρίζει τη μορφή του, και το επισημαίνει ρητά ως γνωστό, συγκεκριμένο λάθος. Ο άλλος τρόπος σχετίζεται με οποιοδήποτε άλλο συντακτικό σφάλμα που δεν ταιριάζει σε αυτά τα δύο γνωστά μοτίβα. Εδώ ο Debugger δεν προσπαθεί να το ταξινομήσει περαιτέρω, απλώς μεταφέρει αυτούσιο το μήνυμα και τον αριθμό γραμμής που ήδη επέστρεψε ο parser της Python. Μόνο στην περίπτωση των δύο γνωστών λαθών γίνεται πραγματική αντικατάσταση, και αυτή γίνεται εξ ολοκλήρου μέσα στον Debugger, προκειμένου να μη φτάνει στο φοιτητή κάποια λανθασμένη εκτίμηση.

Αν όμως ο κώδικας είναι συντακτικά έγκυρος το ερώτημα αλλάζει εντελώς. Δεν υπάρχει πια ένα μονοσήμαντο σφάλμα να αναφερθεί, όπως στην προηγούμενη περίπτωση, αλλά ανοιχτά ερωτήματα. Μήπως λείπει κάποια δομή που ζητά η εκφώνηση της άσκησης; Μήπως κάτι στη λογική του κώδικα είναι λάθος, ακόμα κι αν τρέχει κανονικά;

Το πρώτο βήμα παραμένει ντετερμινιστικό, μόνο που τώρα υπάρχει δέντρο να εξεταστεί. Διατρέχοντάς το, ο Debugger καταγράφει αντικειμενικά, μετρήσιμα στοιχεία. Υπάρχει έστω μία δομή `if` στον κώδικα; Υπάρχει `for loop`; Ποιες μεταβλητές έχουν οριστεί πριν χρησιμοποιηθούν, και ποιες όχι; Χρησιμοποιείται η `append()` σε κάποια λίστα; Υπάρχει αριθμητική τιμή γραμμένη μέσα σε εισαγωγικά, σαν να ήταν κείμενο; Κάθε τέτοιος έλεγχος είναι μια απλή ναι ή όχι ερώτηση πάνω στη δομή, και δεν κρίνει τίποτα ακόμα.

Ο λόγος που η καταγραφή αυτή γίνεται πάνω στο δέντρο, και όχι διαβάζοντας τον κώδικα σαν κείμενο, είναι ότι ο προγραμματισμός δεν οργανώνεται γραμμικά όπως η φυσική γλώσσα. Ένα `for loop` μπορεί να περιέχει μέσα του ένα `if`, το οποίο με τη σειρά του περιέχει άλλο `for loop`. Το AST διατηρεί ακριβώς αυτές τις σχέσεις γονέα και παιδιού, δείχνοντας ποιο κομμάτι κώδικα βρίσκεται μέσα σε ποιο. Η πληροφορία αυτή χάνεται αν κοιτάζει κανείς μόνο τις λέξεις μια μια, στη σειρά που εμφανίζονται, και η απώλειά της έχει αναγνωριστεί ως αδυναμία σε εργαλεία που αντιμετωπίζουν τον πηγαίο κώδικα ως φυσική γλώσσα [25].

Η δομή από μόνη της, ωστόσο, δεν αρκεί πάντα. Σε εργαλεία εντοπισμού σφαλμάτων έχειδειχθεί ότι ο συνδυασμός της δενδρικής αναπαράστασης με πληροφορία για τη ροή του προγράμματος διακρίνει εμφανώς καλύτερα τον προβληματικό από τον υγιή κώδικα από ότι η δομή μόνη της [26]. Την ίδια λογική ακολουθεί ένας από τους ελέγχους του Debugger, ο οποίος εξετάζει αν ένας βρόχος `while` πρόκειται ποτέ να τερματίσει. Το ερώτημα αυτό δεν απαντιέται από την ύπαρξη ή την απουσία μιας δομής, χρειάζεται να συγκριθεί ποιες μεταβλητές ελέγχει η συνθήκη με ποιες μεταβάλλονται μέσα στο κείμενο (Παράρτημα Α, Σχήμα Α.3).

Αυτά τα ευρήματα αποκτούν νόημα μόνο σε σχέση με το τι ζητάει η συγκεκριμένη άσκηση. Αν η εκφώνηση αναφέρει, για παράδειγμα, τη λέξη “`if`”, και το AST δείχνει ότι δεν υπάρχει καμία τέτοια δομή στον κώδικα, αυτό σημειώνεται ως πιθανό εύρημα. Η διασταύρωση όμως είναι σκόπιμα χαλαρή. Δεν δηλώνει πως η άσκηση απέτυχε, απλώς ότι αυτό το κομμάτι αξίζει να ελεγχθεί περαιτέρω. Η οριστική κρίση για το αν η άσκηση ικανοποιείται πλήρως δεν είναι δουλειά του Debugger.

Το τι ακριβώς σημαίνουν, όμως, αυτά τα δομικά ευρήματα δεν αποφασίζεται μηχανικά. Ο λόγος είναι ότι τα ίδια τα ευρήματα, από μόνα τους, δεν αρκούν πάντα για μια σαφή απάντηση. Ο κώδικας μπορεί να έχει και δομή `if` και `for loop`, όπως ζητά η εκφώνηση, αλλά η λογική μέσα τους να είναι αντεστραμμένη, ή μπορεί να λείπει κάτι που το AST δεν εντοπίζει άμεσα ως ξεχωριστή κατηγορία. Χρειάζεται δηλαδή μια κρίση πάνω στα δεδομένα, όχι απλή καταμέτρησή τους.

Αυτή την κρίση την κάνει μια κλήση σε LLM, αλλά με πολύ συγκεκριμένους όρους. Ο Debugger δεν ρωτά απλώς “τι πρόβλημα έχει αυτός ο κώδικας;”, κάτι που θα άφηνε το LLM ελεύθερο να φανταστεί οτιδήποτε. Του δίνει τα ίδια τα δομικά ευρήματα που ήδη έχει συγκεντρώσει, τα χαρακτηρίζει ρητά μέσα στο prompt ως αντικειμενικά δεδομένα προς αξιολόγηση και του ζητά να διαλέξει ακριβώς μία κατηγορία προβλήματος από έναν κλειστό, προκαθορισμένο κατάλογο, ή να απαντήσει ότι δεν υπάρχει κανένα. Το ίδιο το system prompt του πράκτορα λέει ρητά πως πρέπει να κρίνει μόνος του αν υπάρχει πρόβλημα και ποιο είναι, αλλά αποκλειστικά βάσει των στοιχείων που του δίνονται, ποτέ βάσει υποθέσεων.

Αν το LLM είχε τη δυνατότητα να απαντήσει με ελεύθερο κείμενο, θα μπορούσε να περιγράψει το ίδιο ακριβώς σφάλμα με πολλούς διαφορετικούς τρόπους, κάνοντας αδύνατο για τον Mentor να ξέρει τι ακριβώς έγινε αντιληπτό ή να το συνδέσει αξιόπιστα με τη σωστή παιδαγωγική εξήγηση. Περιορίζοντας την απάντηση σε μία από λίγες, προκαθορισμένες κατηγορίες, το αποτέλεσμα γίνεται προβλέψιμο.

Το ίδιο πλαίσιο προβλέπει και τι γίνεται όταν ο κώδικας χρησιμοποιεί κάτι που ο Debugger δεν έχει σχεδιαστεί να αναγνωρίζει με σιγουριά, π.χ. try/except, list comprehensions, έννοιες πέρα από τις βασικές δομές που καλύπτουν οι κατηγορίες του καταλόγου. Δίνεται τότε ρητή οδηγία να προτιμηθεί η επιλογή “κανένα πρόβλημα”, αντί να υποτεθεί λάθος που δεν είναι σίγουρο ότι υπάρχει. Ένα ψευδές θετικό, δηλαδή μια υπόδειξη για πρόβλημα που δεν υπάρχει, θα οδηγούσε τον φοιτητή να ψάχνει κάτι ανύπαρκτο, χάνοντας χρόνο και εμπιστοσύνη στο σύστημα. Ένα ψευδές αρνητικό, δηλαδή η μη αναγνώριση ενός πραγματικού προβλήματος σε αυτό το σημείο, δεν είναι το τέλος της διαδικασίας. Ο Assessor, στο επόμενο βήμα, ελέγχει ούτως ή άλλως ανεξάρτητα αν ικανοποιούνται τα συγκεκριμένα κριτήρια της άσκησης, οπότε ένα πρόβλημα που δεν εντοπίστηκε εδώ έχει ακόμα ευκαιρία να εντοπιστεί εκεί. Το κόστος ενός λανθασμένου ισχυρισμού είναι μεγαλύτερο από το κόστος μιας παράλειψης που μπορεί να καλυφθεί αλλού, και αυτή η ασυμμετρία δικαιολογεί την προτίμηση προς το “κανένα πρόβλημα”.

Αυτή η λογική της ταξινόμησης σε κλειστό σύνολο κατηγοριών αντί για ελεύθερη παραγωγή εξήγησης δεν είναι πρωτότυπη επιλογή του συστήματός μας. Ακολουθεί ήδη τεκμηριωμένη κατεύθυνση στη βιβλιογραφία γύρω από αυτόματη ανατροφοδότηση για αρχάριους προγραμματιστές. Το εργαλείο TEGCER, για παράδειγμα, ταξινομεί σφάλματα μεταγλώττισης σε εκατοντάδες προκαθορισμένες κατηγορίες αντί να παράγει ελεύθερα τη διόρθωση, πετυχαίνοντας υψηλή ακρίβεια ταξινόμησης και βοηθώντας φοιτητές να διορθώσουν σφάλματα ταχύτερα απ' ό,τι με ανθρώπινους βοηθούς διδασκαλίας [30], μια ένδειξη ότι μια συνεπής, δομημένη ταξινόμηση σφαλμάτων μπορεί να είναι συγκρίσιμη σε αποτελεσματικότητα με την επί τούτο ανθρώπινη βοήθεια. Την ίδια γενική αρχή συναντάμε και σε εργαλεία εντοπισμού σφαλμάτων βασισμένα σε AST, έστω και σε διαφορετικό πλαίσιο. Λιγότερες, πιο ουσιαστικές κατηγορίες αντί για εξαντλητική ακατέργαστη λεπτομέρεια. Εκεί η περικοπή και ομαδοποίηση των κόμβων του δέντρου γίνεται για να μην επιβαρύνεται η εκπαίδευση ενός μοντέλου μηχανικής μάθησης με περιττή πληροφορία, όμως η λογική είναι η ίδια. Η επιλεκτική περικοπή αυξάνει, δεν μειώνει, την ακρίβεια της ανάλυσης [27]. Η ίδια δενδρική αναπαράσταση έχει αξιοποιηθεί και σε καθαρά εκπαιδευτικά πλαίσια, ως μέσο για να γίνει ορατή η δομή ενός προγράμματος σε αρχάριους προγραμματιστές [28].

Πέρα από το τρέχον αίτημα, η σταθερότητα αυτών των κατηγοριών έχει και ένα δεύτερο όφελος στο συγκεκριμένο σύστημα. Όταν μια απάντηση δεν είναι σωστή λόγω σημασιολογικού προβλήματος, η κατηγορία που επέλεξε ο Debugger δεν χρησιμοποιείται μόνο για την τρέχουσα υπόδειξη. Προστίθεται σε ένα ιστορικό συχνών λαθών του συγκεκριμένου φοιτητή, το οποίο τροφοδοτεί

μελλοντικές υποδείξεις του Mentor και σε άλλες ασκήσεις. Αυτό δεν θα ήταν εφικτό αν η περιγραφή του προβλήματος διατυπωνόταν κάθε φορά διαφορετικά. Εφόσον το ίδιο είδος λάθους αντιστοιχεί πάντα στην ίδια, σταθερή ετικέτα, μπορεί το σύστημα να “θυμάται” ότι ένας φοιτητής κάνει επανειλημμένα το ίδιο είδος λάθους σε διαφορετικές ασκήσεις.

Σε καμία περίπτωση ο Debugger δεν παράγει διορθωμένο κώδικα, ούτε καν εσωτερικά για δικό του έλεγχο. Αν και άλλες προσεγγίσεις στη βιβλιογραφία συγκρίνουν δομικά τον λανθασμένο κώδικα με μια σωστή λύση-αναφορά και παράγουν αυτόματα διορθωμένη εκδοχή του προγράμματος [29], εδώ ακολουθείται διαφορετική τακτική. Μια τέτοια σύγκριση σε αυτό το σύστημα θα απαιτούσε σημείο αναφοράς παραγόμενο από LLM, αφού οι ασκήσεις συντίθενται δυναμικά από πρότυπα με μεταβλητά ονόματα και τιμές και κάθε εκδοχή τους είναι διαφορετική. Η ορθότητα της κρίσης θα στηριζόταν τότε σε ένα δεύτερο, μη επαληθευμένο αποτέλεσμα LLM, ακριβώς αυτό που η ντετερμινιστική θεμελίωση υπάρχει για να αποκλείσει. Ο Debugger κρίνει επομένως τον κώδικα ως προς τη δομή του και ως προς την εκφώνηση, ποτέ ως προς μια λύση που θα έπρεπε πρώτα να εμπιστευτεί.

Όπως και στους άλλους δύο πράκτορες, υπάρχει δίκτυ ασφαλείας αν η κλήση LLM αποτύχει τεχνικά ή επιστρέψει κάτι που δεν μπορεί να ερμηνευτεί ως μία από τις έγκυρες κατηγορίες, επιστρέφοντας τον Debugger πίσω στα ντετερμινιστικά ευρήματα που είχε ήδη συγκεντρώσει. Αν δεν έχει εντοπιστεί τίποτα ντετερμινιστικά, η προεπιλογή είναι ότι δεν υπάρχει πρόβλημα.

Το σύνολο της διαδικασίας που περιγράφηκε εκτελείται αυτούσιο, ανεξάρτητα από το ποιο από τα τρία κουμπιά της εφαρμογής την πυροδοτήσει. Ο Debugger δεν ξέρει, και δεν χρειάζεται να ξέρει, αν πρόκειται για Μαθήματα, Εξάσκηση ή Αξιολόγηση Κώδικα. Στην Αξιολόγηση Κώδικα ειδικότερα, όπου δεν υπάρχουν κριτήρια επιτυχίας άσκησης, οι έλεγχοι που εξαρτώνται από αυτά (π.χ. αν λείπει μια δομή που θα ζητούσε μια εκφώνηση) απλώς δεν ενεργοποιούνται ποτέ. Αυτό δεν συμβαίνει επειδή ο Debugger κάνει κάτι διαφορετικό σε αυτή την περίπτωση, αλλά επειδή δεν υπάρχει η πληροφορία που θα τους ενεργοποιούσε. Οι έλεγχοι που δεν εξαρτώνται από κριτήρια άσκησης, όπως η χρήση μεταβλητής πριν οριστεί, το πιθανό ατέρμονο loop, ο λάθος αριθμός ορισμάτων σε κλήση συνάρτησης, παραμένουν ενεργοί κανονικά, ότι κι αν ανεβάσει ο φοιτητής. Το πού θα καταλήξει τελικά το εύρημα, στον Assessor ή απευθείας στον Mentor, αποφασίζεται αλλού.

3.6 Assessment Agent

Ο Assessor είναι ο πράκτορας που κρίνει αν ο κώδικας του φοιτητή ικανοποιεί τη συγκεκριμένη άσκηση που του ανατέθηκε. Ενώ ο Debugger αξιοποιεί την εκφώνηση και τα κριτήρια μόνο ως ένδειξη για το ποιες δομές αναμένονται, ο Assessor τα αντιμετωπίζει ως τον ίδιο τον κανόνα της κρίσης του. Δέχεται δεδομένο ότι ο κώδικας μπορεί να είναι τεχνικά έγκυρος και εξετάζει αν ικανοποιεί τα συγκεκριμένα, ρητά κριτήρια (success_criteria) της τρέχουσας άσκησης. Το αποτέλεσμα της κρίσης του δεν είναι απλώς σωστό ή λάθος, αλλά είναι ένα σύνολο τεσσάρων στοιχείων. Αν ο κώδικας είναι σωστός (is_correct), τι πρέπει να συμβεί στη συνέχεια (assessment_decision: advance αν ο φοιτητής προχωράει στην επόμενη ενότητα, repeat αν χρειάζεται να ξαναδοκιμάσει στην ίδια, support αν χρειάζεται πιο ενεργή καθοδήγηση στην τρέχουσα άσκηση), πόσο καλά φαίνεται να έχει κατανοήσει ο φοιτητής την ύλη (understanding_level), και μια σύντομη τεχνική περιγραφή (assessment_feedback) που δεν προορίζεται για τον φοιτητή, αλλά χρησιμοποιείται ως context από τον Mentor.

Η κρίση του Assessor περνά από τρία διαδοχικά επίπεδα ελέγχου πριν κληθεί το LLM. Το πρώτο επίπεδο βασίζεται στα success_criteria της άσκησης, μια σύντομη λίστα προτάσεων σε φυσική

γλώσσα (π.χ. "Χρήση δομής επανάληψης for", "Χρήση της append() σε λίστα"). Ο έλεγχος εδώ αφορά μόνο τη γενική παρουσία μιας δομής, όχι το αν χρησιμοποιείται σωστά ή με συγκεκριμένο περιεχόμενο. Κάθε τέτοιο κριτήριο αντιστοιχίζεται σε ένα δομικό εύρημα του κώδικα μέσω μιας συνάρτησης που αναγνωρίζει συγκεκριμένες λέξεις-κλειδιά (π.χ. η λέξη "for" ή "επανάληψη" αντιστοιχίζεται στην ύπαρξη κάποιας δομής for στο AST). Η αντιστοίχιση αυτή δεν περιορίζεται σε απλή αναζήτηση λέξεων μέσα στον πηγαίο κώδικα. Για κριτήρια που αναφέρονται σε print(), για παράδειγμα, δεν αρκεί η απλή παρουσία μιας κλήσης print() οπουδήποτε στο αρχείο, ελέγχεται αν αυτή η κλήση βρίσκεται πραγματικά σε σημείο του κώδικα που εκτελείται, δηλαδή αν δεν είναι θαμμένη μέσα σε μια συνάρτηση που ορίζεται αλλά δεν καλείται ποτέ. Παρόλα αυτά, ο έλεγχος παραμένει στο κομμάτι της δομής και όχι του περιεχομένου, δεν ενδιαφέρεται τι ακριβώς τυπώνεται, μόνο ότι υπάρχει έστω ένα προσβάσιμο print(). Η αντιστοίχιση είναι εξ ολοκλήρου ντετερμινιστική, δεν περνά από LLM, ούτε καν στην περίπτωση που κάποιο κριτήριο δεν αναγνωρίζεται από καμία λέξη-κλειδί, οπότε θεωρείται προεπιλεγμένα ικανοποιημένο. Το ποσοστό των κριτηρίων που αντιστοιχίζονται θετικά με αυτόν τον τρόπο αποτελεί ένα πρώτο, χαλαρό δείκτη ικανοποίησης της άσκησης, στο οποίο αναφερόμαστε παρακάτω ως το εσωτερικό ποσοστό ικανοποίησης κριτηρίων. Το ίδιο σύνολο δομικών ευρημάτων (has_if, has_for, has_append, has_list κλπ) υπολογίζεται με ανάλυση του AST του κώδικα, ανεξάρτητα από το αντίστοιχο σύνολο ευρημάτων που υπολογίζει ο Debugger για δικούς του σκοπούς. Οι δύο πράκτορες δεν μοιράζονται αυτή την ανάλυση, ο καθένας τη φτιάχνει ξανά από την αρχή, γιατί την χρησιμοποιούν για διαφορετικό σκοπό: ο Debugger για να αποφασίσει αν υπάρχει τεχνικό πρόβλημα, ενώ ο Assessor για να κρίνει αν ικανοποιούνται συγκεκριμένα κριτήρια.

Αυτό το πρώτο επίπεδο όμως θα περνούσε και μια λύση που χρησιμοποιεί εντελώς λάθος ονόματα ή τιμές, αρκεί να υπάρχει η σωστή γενική δομή. Αν η εκφώνηση ζητά για παράδειγμα μεταβλητή age με τιμή 25 και ο φοιτητής γράψει `x = 999`, το πρώτο επίπεδο θα δει απλώς ότι υπάρχει μια ανάθεση και θα το θεωρήσει ικανοποιητικό. Για να αποφευχθεί αυτό, το δεύτερο επίπεδο αντλεί πληροφορία από διαφορετικό σημείο, όχι από τα success criteria, αλλά την ίδια την εκφώνηση της άσκησης. Ο Assessor δεν τη διαβάζει μόνο για να τη δώσει ως context στο LLM, αλλά την αναλύει και ο ίδιος, για να εξαγάγει συγκεκριμένες, ελέγξιμες απαιτήσεις, όπως ονόματα μεταβλητών που πρέπει να υπάρχουν, αριθμητικές τιμές που πρέπει να εμφανίζονται, strings που πρέπει να τυπωθούν με print(). Έπειτα ελέγχει απευθείας στο δέντρο του κώδικα αν αυτές οι συγκεκριμένες απαιτήσεις ικανοποιούνται, ψάχνοντας ακριβώς το όνομα, την τιμή ή το string που ανέφερε η εκφώνηση, όχι απλώς αν υπάρχει κάποια δομή του ίδιου γενικού τύπου. Αυτή η αυστηρή αντιστοίχιση με την εκφώνηση, όπως την ονομάζει ο κώδικας, θα εντόπιζε στο παράδειγμα με το age ότι ούτε το όνομα ούτε η τιμή 25 εμφανίζονται πουθενά στις αναθέσεις του κώδικα, και θα σημείωνε αποτυχία ακριβώς εκεί που το πρώτο επίπεδο θα την προσπερνούσε. Στο ίδιο πλαίσιο εντάσσεται και ένας πιο εξειδικευμένος έλεγχος για μια συγκεκριμένη, συνηθισμένη περίπτωση λάθους. Όταν η εκφώνηση περιγράφει μια απλή if/else απόφαση (π.χ. "αν μεγαλύτερη από 50, τύπωσε High, αλλιώς Low"), ο Assessor συγκρίνει την κατεύθυνση της συνθήκης στον κώδικα με την κατεύθυνση που περιγράφει η εκφώνηση, ώστε να εντοπίσει αντεστραμμένη λογική ακόμα κι όταν και οι δύο τιμές τυπώνονται σωστά, απλώς η μία στη θέση της άλλης.

Το τρίτο επίπεδο συνδέεται άμεσα με τα δύο προηγούμενα, καθώς συνδυάζει τα ευρήματά τους με νέους, αυστηρότερους κανόνες ελέγχου. Το system prompt του Assessor δηλώνει ρητά μηδενική ανοχή σε σημασιολογικά λάθη τύπων δεδομένων, όπου το περιεχόμενο χωρίς τη σωστή μορφή θεωρείται εξαρχής λανθασμένο. Στην πράξη αυτό σημαίνει πως αν μια μεταβλητή με αναγνωρίσιμα αριθμητικό όνομα (πχ age, score) έχει οριστεί ως string ("25" αντί για 25), ή αντίστροφα μια

μεταβλητή συμβολοσειράς (πχ username, email) έχει τιμή που δεν αντιστοιχεί σε αυτήν, ο κώδικας απορρίπτεται. Ο έλεγχος αυτός δεν περιορίζεται μόνο σε μια λίστα γνωστών ονομάτων. Όταν η ίδια η εκφώνηση ή τα κριτήρια δηλώνουν ρητά ότι ζητείται αριθμητική τιμή, ο έλεγχος αυστηροποιείται και καμία αριθμητική τιμή μέσα σε εισαγωγικά δεν γίνεται δεκτή. Οι συγκεκριμένες λέξεις-κλειδιά που ενεργοποιούν αυτή τη λειτουργία, καθώς και οι υπόλοιποι επιμέρους έλεγχοι, περιγράφονται στην ενότητα 4.7. Οποιοδήποτε τέτοιο εύρημα, σε συνδυασμό με τεχνικό σφάλμα από τον Debugger ή αποτυχία στην αυστηρή αντιστοίχιση με την εκφώνηση, επιφέρει αυτό που ορίζεται ως *hard violation* (σοβαρή παράβαση). Η ενεργοποίησή της μηδενίζει αυτόματα το εσωτερικό ποσοστό ικανοποίησης κριτηρίων, όσα άλλα σημεία κι αν έχουν καλυφθεί. Το ποσοστό αυτό δεν αποτελεί βαθμό του φοιτητή, δεν αποθηκεύεται στη βάση δεδομένων και δεν εμφανίζεται ποτέ στον φοιτητή, υπάρχει μόνο ως ενδιάμεσο δεδομένο εισόδου προς το LLM κατά την τρέχουσα εκτέλεση.

Όλα αυτά τα ευρήματα δίνονται στο LLM μαζί με το ιστορικό του φοιτητή στην τρέχουσα άσκηση (προσπάθειες, hints, χρόνος). Το έργο του LLM είναι να τα συνδυάσει και να καταλήξει σε μία απόφαση πάνω στα τέσσερα προαναφερθέντα στοιχεία, σε αυστηρά καθορισμένη μορφή ώστε η απάντηση να μπορεί να αναλυθεί αυτόματα πίσω σε δομημένα πεδία. Το ίδιο το prompt του θέτει ρητό όριο στην ελευθερία του. Το *is_correct* πρέπει να είναι θετικό μόνο αν όλα τα ευρήματα το επιβεβαιώνουν, με ρητή αναφορά σε μηδενική ανοχή, ακόμα κι αν το ποσοστό ικανοποίησης κριτηρίων που περιγράφηκε πριν είναι υψηλό.

Η απόφαση του LLM, ωστόσο, δεν γίνεται αυτόματα δεκτή. Αφού αναλυθεί η απάντηση, ελέγχεται πρώτα για εσωτερική συνέπεια. Αν το LLM δηλώσει *advance* ενώ ο κώδικας κρίθηκε λάθος, ή *support* ενώ κρίθηκε σωστός, η απάντηση απορρίπτεται εξ ολοκλήρου ως ασυνεπής, και ο Assessor πέφτει στο ντετερμινιστικό εναλλακτικό μονοπάτι. Ελέγχεται επίσης αν η αιτιολόγηση της απόρριψης στέκει. Αν το LLM απορρίψει τον κώδικα επικαλούμενο δομή που λείπει, ενώ το δέντρο τη δείχνει παρούσα, ο ισχυρισμός είναι ψευδής και η απάντηση δεν γίνεται δεκτή. Πρόκειται για τον ίδιο μηχανισμό επαλήθευσης που χρησιμοποιεί ο Debugger για τον έλεγχο τύπων δεδομένων, μεταφερόμενο στον δεύτερο πράκτορα. Αν όμως η απάντηση είναι εσωτερικά συνεπής, εφαρμόζονται ακόμα δύο ντετερμινιστικοί έλεγχοι πάνω της, ανεξάρτητα από το τι αποφάσισε το LLM. Αν ο κώδικας κρίθηκε λάθος και η απόφαση είναι *repeat*, αλλά ο φοιτητής έχει ήδη πολλές προσπάθειες, αρκετά hints, ή έχει περάσει πολύ χρόνο σε αυτή την άσκηση, η απόφαση αναβαθμίζεται σε *support*. Αντίστροφα, αν ο κώδικας κρίθηκε σωστός και η απόφαση είναι *advance*, αλλά χρειάστηκε δυσανάλογα μεγάλος αριθμός προσπαθειών ή hints για να φτάσει εκεί, η απόφαση υποβαθμίζεται σε *repeat*, παρότι ο κώδικας είναι τεχνικά σωστός. Το ίδιο μοτίβο επικύρωσης εφαρμόζεται και στο *understanding_level*. Αν η τιμή που επιστρέφει το μοντέλο δεν ανήκει στις προκαθορισμένες έγκυρες κατηγορίες, δεν απορρίπτεται το σύνολο της απάντησης, αλλά αντικαθίσταται μόνο η συγκεκριμένη τιμή από μια ντετερμινιστική συνάρτηση. Η συνάρτηση αυτή υπολογίζει το επίπεδο κατανόησης συνδυάζοντας το ποσοστό επιτυχίας, το πλήθος των προσπαθειών, τον αριθμό των υποδείξεων και την τελική ορθότητα του κώδικα. Το να καταλήξει κανείς στη σωστή απάντηση δεν είναι από μόνο του αρκετό για να προχωρήσει, αν χρειάστηκε υπερβολική προσπάθεια για να φτάσει εκεί. Η λογική αυτή αντιστοιχεί στην έννοια της *mastery learning* [1], όπου η προαγωγή σε επόμενο στάδιο προϋποθέτει επίδειξη επαρκούς, ανεξάρτητης επάρκειας πάνω στην τρέχουσα ύλη, όχι απλώς τελικά ορθή απάντηση ανεξαρτήτως του πόσες φορές ή με πόση βοήθεια χρειάστηκε να φτάσει κανείς εκεί.

Αν η κλήση στο LLM αποτύχει τεχνικά, ή η απάντηση δεν μπορεί να αναλυθεί καθόλου στην αναμενόμενη μορφή, ο Assessor δεν επινοεί μια νέα λογική εναλλακτικά. Χρησιμοποιεί ακριβώς τα ίδια αντικειμενικά ευρήματα και τα ίδια όρια προσπαθειών, hints και χρόνου που θα χρησιμοποιούσε ούτως ή άλλως για να ελέγξει τη συνέπεια μιας απάντησης του LLM, απλώς τα εφαρμόζει απευθείας.

Ο κώδικας θεωρείται σωστός μόνο αν δεν υπάρχει καμία *hard violation* και όλα τα κριτήρια ικανοποιούνται. Η απόφαση προκύπτει από τα ίδια όρια δυσκολίας που περιγράφηκαν πριν. Το εναλλακτικό μονοπάτι δεν είναι επομένως μια ξεχωριστή, απλούστερη λογική, είναι η ίδια η λογική επικύρωσης που χρησιμοποιείται ούτως ή άλλως, μόνο που εδώ γίνεται η μοναδική πηγή της απόφασης αντί για έλεγχο πάνω σε μια απόφαση του LLM. Υπάρχει ακόμα ένα δεύτερο, πιο εξωτερικό δίκτυο ασφαλείας, ξεχωριστό από αυτό. Αν αποτύχει κάτι στην ίδια τη συλλογή των αντικειμενικών ευρημάτων, πριν καν φτάσουμε στο σημείο να χρειαστεί απόφαση, δεν υπάρχουν δεδομένα πάνω στα οποία να στηριχθεί οποιαδήποτε από τις παραπάνω λογικές. Σε αυτή την ακραία περίπτωση επιστρέφεται μια απλή, καθολική απάντηση (ο κώδικας θεωρείται λάθος, η ενότητα επαναλαμβάνεται, το επίπεδο κατανόησης θεωρείται σε εξέλιξη), ώστε ο φοιτητής να λάβει έστω μια απάντηση αντί να μείνει χωρίς καμία.

Ο ρόλος του *assessment_feedback* αποκαλύπτει κάτι πιο γενικό για το πώς λειτουργεί ολόκληρη αυτή η κρίση. Στην εκπαίδευση υπάρχουν δύο βασικά είδη αξιολόγησης. Το *summative assessment* (συνολική αξιολόγηση) δίνει έναν τελικό βαθμό στο τέλος μιας διαδικασίας, σαν πιστοποίηση αυτού που έμαθε κάποιος. Το *formative assessment* (διαμορφωτική αξιολόγηση) είναι διαφορετικό. Δεν δίνει τελικό βαθμό. Στην βιβλιογραφία περιγράφεται ως το σύνολο των δραστηριοτήτων που παρέχουν πληροφορία προς τον εκπαιδευτικό ή τον μαθητή, πληροφορία η οποία στη συνέχεια χρησιμοποιείται για να αλλάξει τη διδασκαλία ή τη μάθηση [34]. Ο *Assessor*, μέσα από τις αποφάσεις για *advance*, *repeat* ή *support* λειτουργεί με τον δεύτερο τρόπο, τροφοδοτώντας με *context* τον *Mentor* για να υλοποιήσει την επόμενη παιδαγωγική ενέργεια.

3.7 Open Learner Model & Αυτορρύθμιση

Μέχρι εδώ, η αρχιτεκτονική που περιγράψαμε αφορούσε το πώς αποφασίζεται κάθε μεμονωμένη αλληλεπίδραση. Ο *Debugger* εντοπίζει τεχνικά προβλήματα σε μια συγκεκριμένη υποβολή κώδικα. Ο *Assessor* κρίνει αν αυτή η συγκεκριμένη υποβολή ικανοποιεί τη δεδομένη άσκηση. Ο *Mentor* μεταφράζει τα ευρήματά τους σε λόγο προς τον φοιτητή. Και οι τρεις όμως ξεχνούν ότι έκριναν μόλις τελειώσει η κάθε αλληλεπίδραση, κανένας τους δεν κρατά μια συνολική εικόνα του φοιτητή στον χρόνο, ποιες ενότητες έμαθε καλά, πού δυσκόλεψε, πώς εξελίσσεται ανάμεσα σε συνεδρίες. Αυτή η συνολική εικόνα υπάρχει, αλλά ζει έξω από τους τρεις πράκτορες, σε μια δομή δεδομένων που ονομάζεται *mastery_profile*. Στην κλασική αρχιτεκτονική ενός Έξυπνου Συστήματος Διδασκαλίας, ένα τέτοιο μοντέλο του μαθητή δεν είναι προαιρετικό πρόσθετο, είναι ένα από τα θεμελιώδη αρχιτεκτονικά κομμάτια που πρέπει να ορίζει το σύστημα, εξίσου σημαντικό με το πώς διδάσκει ή πώς αξιολογεί [13]. Το *mastery_profile* είναι η υλοποίηση ακριβώς αυτού του κομματιού, με μια επιπλέον ιδιότητα: αντί να παραμένει κρυφό, εσωτερικό δεδομένο, επιστρέφεται απευθείας στον φοιτητή, ακατέργαστο, χωρίς μεσολάβηση κάποιου LLM. Ένα τέτοιο ανοιχτό μοντέλο μαθητή, γνωστό στη βιβλιογραφία ως *Open Learner Model (OLM)*, έχει συγκεκριμένο σκοπό: να βοηθήσει τον μαθητή να παρακολουθεί την πρόοδό του, να αναστοχάζεται πάνω σε αυτήν, και να ρυθμίζει ο ίδιος τον ρυθμό της μάθησής του [35]. Η ενότητα αυτή εξετάζει τρία σχετιζόμενα κομμάτια. Πρώτα, πώς υπολογίζεται και τι ακριβώς δείχνει το *mastery_profile* ανά ενότητα. Μετά, πώς ένας ξεχωριστός μετρητής συνεχόμενων σωστών απαντήσεων στην Εξάσκηση δίνει στον φοιτητή άμεση εικόνα της τρέχουσας πορείας του. Και τέλος, πώς όλα αυτά συγκλίνουν σε ένα συνολικό μήνυμα όταν ολοκληρωθεί το πρόγραμμα.

Ας δούμε ένα συγκεκριμένο παράδειγμα για να γίνει αυτό απτό. Ένας φοιτητής έχει ολοκληρώσει τις δύο πρώτες ενότητες και βρίσκεται τώρα στην τρίτη (Λίστες). Στη δεύτερη ενότητα (Δομές Ελέγχου) χρειάστηκε τέσσερις προσπάθειες και τρία *hints* πριν το πετύχει. Το *mastery_profile* του θα δείξει:

100% στις Μεταβλητές, με καμία ένδειξη δυσκολίας. 100% στις Δομές Ελέγχου, αλλά ME ένδειξη δυσκολίας ενεργή, παρότι η ενότητα ολοκληρώθηκε επιτυχώς. Κάποιο ποσοστό ανάμεσα σε 0% και 95% στις Λίστες, ανάλογα με το αν και πόσο έχει ήδη προσπαθήσει εκεί. 0% στις Επαναλήψεις και τις Συναρτήσεις, γιατί δεν τις έχει φτάσει ακόμα. Το ίδιο το ποσοστό της τρέχουσας ενότητας, όπως και η ένδειξη δυσκολίας, δεν προκύπτουν από κάποιο νέο υπολογισμό ειδικά για το OLM. Χρησιμοποιούν ξανά ότι έχουν ήδη αποφασίσει ο Assessor και το ιστορικό της συνομιλίας, με σκοπό να το κάνουν ορατό, όχι να το υπολογίσουν από την αρχή.

Η μοντελοποίηση της γνώσης ενός μαθητή στον χρόνο αποτελεί αυτοτελές ερευνητικό πεδίο, γνωστό ως knowledge tracing (ιχνηλάτηση γνώσης), με προσεγγίσεις που εκτείνονται από πολύπλοκα στατιστικά μοντέλα έως και τεχνητή νοημοσύνη [45]. Το mastery_profile της παρούσας εργασίας δεν ακολουθεί αυτή την περίπλοκη προσέγγιση. Αντίθετα, χρησιμοποιεί μία απλή αριθμητική τιμή για κάθε ενότητα, επειδή η πληροφορία αυτή απευθύνεται απευθείας στον ίδιο τον φοιτητή. Χρησιμοποιώντας μετρήσεις που βασίζονται στην πιο πρόσφατη αξιολόγηση του Assessor, το ποσοστό προόδου μεταφράζεται σε αυξανόμενες αριθμητικές τιμές ανάλογα με το επίπεδο κατανόησης, από τη χαμηλότερη για την ανάγκη υποστήριξης έως την υψηλότερη για την ισχυρή κατανόηση. Οι ακριβείς τιμές δίνονται στην ενότητα 4.9.

Η ένδειξη δυσκολίας που είδαμε στο παραπάνω παράδειγμα δεν προέρχεται από κάποιο ξεχωριστό πεδίο της βάσης δεδομένων. Ανακατασκευάζεται εκ των υστέρων, διαβάζοντας το ίδιο το αποθηκευμένο ιστορικό συνομιλίας, μετρώντας πόσες προσπάθειες και πόσα hints συγκεντρώθηκαν σε μια ενότητα πριν ο φοιτητής προαχθεί από αυτήν. Τα όρια που χρησιμοποιούνται, τρεις προσπάθειες ή δύο hints, δεν είναι νέος, αυθαίρετος κανόνας ειδικά για το OLM. Είναι ακριβώς τα ίδια νούμερα με αυτά που χρησιμοποιεί ο Assessor για να αναβαθμίσει μια απόφαση από repeat σε support.

Η ένδειξη αυτή δεν είναι μόνιμη. Αν ο φοιτητής επιστρέψει αργότερα και εξασκηθεί στη σχετική ενότητα μέσω της λειτουργίας Εξάσκηση, πετυχαίνοντας πέντε συνεχόμενες σωστές απαντήσεις, η ένδειξη αφαιρείται αυτόματα. Το όριο για την κατάργηση της ένδειξης είναι σκόπιμα πιο απαιτητικό σε σχέση με το όριο των τριών προσπαθειών ή δύο υποδείξεων που την ενεργοποίησε αρχικά. Με αυτή τη σχεδιαστική επιλογή εξασφαλίζεται ότι η ανάκαμψη του φοιτητή είναι πραγματική και όχι αποτέλεσμα τυχαίας επιτυχίας. Η διαδικασία αυτή ενισχύει την αυτορρύθμιση στη μάθηση. Το σύστημα δεν επιβάλλει υποχρεωτικές επαναλήψεις, αλλά αφήνει στον ίδιο τον φοιτητή την πρωτοβουλία να αξιολογήσει την εικόνα του, να επιλέξει επιπλέον εξάσκηση και να διαπιστώσει άμεσα τη βελτίωση στο προφίλ του.

Αυτή η λειτουργία αντιστοιχεί στην έννοια της αυτορρυθμιζόμενης μάθησης (self-regulated learning), όπου ο μαθητής παρακολουθεί την πορεία του, καταλαβαίνει πού δυσκολεύεται και επιλέγει μόνος του να διορθώσει τα κενά του. Η βιβλιογραφία επισημαίνει ότι στους μαθητές δεν αρκεί απλώς να βλέπουν δεδομένα, αλλά χρειάζονται εξατομικευμένη υποστήριξη που τους καθοδηγεί στο τι να κάνουν με αυτά [37]. Το mastery_profile κινείται σε αυτή την κατεύθυνση, καθώς η ένδειξη δυσκολίας δεν είναι απλή πληροφορία, αλλά οδηγεί σε μια συγκεκριμένη διαθέσιμη ενέργεια. Παράλληλα, η οπτική αναπαράσταση της προόδου προτείνεται στη βιβλιογραφία ως εργαλείο που βοηθά τον μαθητή να παρακολουθεί την κατάστασή του και να προσαρμόζει τις στρατηγικές του σε διαδικτυακά περιβάλλοντα [38]. Η δυνατότητα του φοιτητή όχι μόνο να παρατηρεί αλλά και να διαμορφώνει την εικόνα της προόδου του μέσα από τις επιλογές του ενισχύει ουσιαστικά τη συμμετοχή του στην εκπαιδευτική διαδικασία [46].

Το mastery_profile περιλαμβάνει επίσης, ανά ενότητα, το ποσοστό όλων των υπόλοιπων χρηστών του συστήματος που έχουν φτάσει ή προσπεράσει την ίδια ενότητα, μια απλή στατιστική στη βάση δεδομένων. Χωρίς αυτό, ο φοιτητής θα έβλεπε μόνο τον δικό του αριθμό απομονωμένο, χωρίς κανένα σημείο αναφοράς για το αν αυτό που βλέπει είναι φυσιολογικό ή όχι. Η σύγκριση με άλλους μαθητές αποτελεί το κεντρικό χαρακτηριστικό της κοινωνικής επέκτασης των OLM, όπου η προβολή της επίδοσης του μαθητή δίπλα στον μέσο όρο της ομάδας συνδέεται με αυξημένο κίνητρο και εμπλοκή [36].

Ξεχωριστό από το mastery_profile, αλλά συμπληρωματικό, είναι ένας απλός μετρητής συνεχόμενων σωστών απαντήσεων στη λειτουργία Εξάσκησης. Μηδενίζεται στην πρώτη λάθος απάντηση και ανεβαίνει σε κάθε σωστή, ανεξάρτητα από ποια ενότητα εξασκείται ο φοιτητής κάθε φορά. Ο μετρητής είναι ορατός στον φοιτητή σε πραγματικό χρόνο όσο εξασκείται, και χρησιμοποιείται επίσης για την αυτόματη προσαρμογή της δυσκολίας μέσα στην Εξάσκηση, ώστε η άσκηση να παραμένει ανάλογη με το πόσο άνετα τα πηγαίνει ο φοιτητής εκείνη τη στιγμή.

Αξίζει να ξεχωρίσουμε αυτόν τον συνολικό μετρητή από το ανά-ενότητα κατώφλι των πέντε σωστών απαντήσεων που είδαμε πριν, γιατί εξυπηρετούν διαφορετικό σκοπό ο καθένας. Ο συνολικός μετρητής πρέπει να αντικατοπτρίζει πώς πάει ο φοιτητής γενικά, αυτή τη στιγμή, στην Εξάσκηση, γι' αυτό μηδενίζεται στο πρώτο λάθος όπου κι αν συμβεί. Αν χρησιμοποιούνταν ο ίδιος μετρητής και για την άρση της ένδειξης δυσκολίας μιας συγκεκριμένης ενότητας, ένα λάθος σε εντελώς άσχετη ενότητα θα ακύρωνε την πρόοδο ανάκαμψης που έχτιζε ο φοιτητής αλλού, κάτι παραπλανητικό. Γι' αυτό υπάρχει και ένας δεύτερος, ξεχωριστός μετρητής, κρατημένος ανά ενότητα, ο οποίος παρακολουθεί αποκλειστικά την ανάκαμψη σε αυτή τη συγκεκριμένη ενότητα, χωρίς να επηρεάζεται από το τι συμβαίνει στις υπόλοιπες.

Τέλος, όταν ο φοιτητής ολοκληρώσει όλο το πρόγραμμα μαθημάτων, το σύστημα συνθέτει ένα συγκεντρωτικό μήνυμα το οποίο συνοψίζει το σύνολο των ασκήσεων που λύθηκαν, τον μέσο χρόνο ανά άσκηση, το τελικό επίπεδο εμπειρίας, και τις τρεις συχνότερες κατηγορίες λαθών που έχουν καταγραφεί για τον συγκεκριμένο φοιτητή σε όλη τη διάρκεια του προγράμματος. Αν κάποια ενότητα έχει παραμείνει σημειωμένη ως δύσκολη μέχρι το τέλος, το μήνυμα την αναφέρει ονομαστικά και προτείνει στοχευμένη επιπλέον εξάσκηση σε αυτή. Το mastery_profile, δηλαδή, δεν σταματά απλά να λειτουργεί όταν τελειώσουν τα μαθήματα, κλείνει τον κύκλο του δείχνοντας στον φοιτητή, ως τελευταία ενέργεια, ακριβώς πού αξίζει να επιστρέψει.

3.8 Επίλογος

Το κεφάλαιο αυτό περιέγραψε την αρχιτεκτονική του προτεινόμενου συστήματος. Ξεκίνησε από τη γενική οργάνωση των τριών πρακτόρων, το επίπεδο ενορχήστρωσης που καθορίζει πότε ενεργοποιείται ο καθένας, και τη δομή του προγράμματος σπουδών (3.2). Συνέχισε με το κεντρικό μοτίβο σχεδίασης πάνω στο οποίο στηρίζεται όλο το σύστημα, τον συνδυασμό ντετερμινιστικής θεμελίωσης και κρίσης LLM (3.3), και ανέλυσε πώς αυτό εφαρμόζεται στον Mentor (3.4), στον Debugger (3.5) και στον Assessor (3.6). Έδειξε πώς το σύστημα κρατά και εκθέτει στον φοιτητή μια συνολική εικόνα της προόδου του (3.7). Η παρούσα ενότητα κλείνει το κεφάλαιο συνδέοντας την αρχιτεκτονική με τα θεμελιώδη κομμάτια ενός Έξυπνου Συστήματος Διδασκαλίας.

Μέσα από αυτή τη διαδρομή, το ίδιο μοτίβο επαναλαμβάνεται σε κάθε επίπεδο, όχι μόνο μέσα σε κάθε πράκτορα αλλά και στη σχέση ανάμεσά τους. Η δρομολόγηση είναι πλήρως προβλέψιμη, ενώ το τι θα αποφασίσει ο καθένας μέσα στον δικό του κόμβο παραμένει ευέλικτο. Είναι ακριβώς σε αυτό το

δεύτερο σημείο, στην κρίση μέσα στο κάθε βήμα και όχι στη χρονική στιγμή ενεργοποίησής του, που εντοπίζεται η αυτονομία του κάθε πράκτορα όπως ορίστηκε στην ενότητα 2.2. Ο Debugger συγκεντρώνει αντικειμενικά ευρήματα πριν κρίνει τι πρόβλημα υπάρχει. Ο Assessor περνά τον κώδικα από τρία επίπεδα ντετερμινιστικού ελέγχου πριν καν κληθεί το μοντέλο, και έπειτα επικυρώνει την απάντησή του με τα ίδια αντικειμενικά κριτήρια. Ο Mentor δεν αποφασίζει ποτέ αν κάτι είναι σωστό, και ακόμα και όταν διατυπώνει, λαμβάνει τη λίστα των δομών που υπάρχουν πραγματικά στον κώδικα ώστε να μη μιλήσει για κάτι ανύπαρκτο. Το μοτίβο δηλαδή δεν εφαρμόζεται μόνο εκεί όπου κρίνεται η ορθότητα, αλλά και εκεί όπου διατυπώνεται το αποτέλεσμα.

Η αρχιτεκτονική αυτή αντιστοιχεί σε κάτι πιο συγκεκριμένο από ένα γενικό σχήμα, δηλαδή στα τέσσερα θεμελιώδη κομμάτια που έχει κατά την κλασική βιβλιογραφία ένα Έξυπνο Σύστημα Διδασκαλίας [13]. Το επίπεδο διεπαφής (3.2) αντιστοιχεί στη διεπαφή χρήστη. Ο συνδυασμός της δρομολόγησης (3.2) και του Mentor (3.4) αντιστοιχεί στο tutoring module, δηλαδή σε αυτό που αποφασίζει πώς και πότε παραδίδεται η διδασκαλία. Το mastery_profile (3.7) αντιστοιχεί στο student model.

Το τέταρτο κομμάτι, το expert ή domain module, είναι το σημείο όπου το σύστημα αποκλίνει σκόπιμα από το κλασικό μοντέλο. Δεν υπάρχει ανεξάρτητος μηχανισμός επίλυσης που να παράγει τη σωστή λύση μιας άσκησης για σύγκριση, για τον λόγο που εξηγήθηκε στην ενότητα 3.5. Η γνώση του πεδίου υπάρχει μόνο ως θεωρία και ως κριτήρια επιτυχίας σε φυσική γλώσσα, ποτέ ως έτοιμος κώδικας αναφοράς, ακριβώς για να μην εξαρτάται η ορθότητα μιας απάντησης από ένα δεύτερο, μη επαληθευμένο αποτέλεσμα LLM. Η απόκλιση αυτή είναι συνεπής με τον δεύτερο ερευνητικό στόχο, καθώς κάθε σημείο όπου το σύστημα θα μπορούσε να στηριχθεί σε παραγόμενο και ανεπιβεβαίωτο περιεχόμενο αντικαθίσταται από ελέγξιμη ανάλυση.

Το επόμενο κεφάλαιο μεταφέρει αυτή την αρχιτεκτονική περιγραφή στο επίπεδο της υλοποίησης, εξετάζοντας τις τεχνολογικές επιλογές, τη μορφή των prompts και τις μηχανικές προκλήσεις που προέκυψαν κατά την ανάπτυξη.

Κεφάλαιο 4ο: Υλοποίηση

4.1 Εισαγωγή

Η υλοποίηση του συστήματος, του οποίου η αρχιτεκτονική παρουσιάστηκε στο Κεφάλαιο 3, περιγράφεται αναλυτικά στο παρόν κεφάλαιο. Εξετάζονται οι τεχνολογίες που χρησιμοποιήθηκαν, ο τρόπος αποθήκευσης των δεδομένων, η μορφή των prompts των τριών πρακτόρων, και οι τεχνικές αποφάσεις που καθόρισαν την ανάπτυξη του συστήματος.

Το σύστημα υλοποιήθηκε ως διαδικτυακή εφαρμογή δύο επιπέδων. Το backend, γραμμένο σε Python, φιλοξενεί τους τρεις πράκτορες, τη λογική ενορχήστρωσης, την επικοινωνία με τα Μεγάλα Γλωσσικά Μοντέλα και τη διαχείριση της βάσης δεδομένων. Το frontend, γραμμένο σε JavaScript, τρέχει στον φυλλομετρητή (browser) του φοιτητή και αναλαμβάνει την παρουσίαση και την αλληλεπίδραση με τον χρήστη. Η επικοινωνία ανάμεσα στα δύο γίνεται μέσω αιτημάτων HTTP.

Η ενότητα 4.2 παρουσιάζει την τεχνολογική στοίβα και τεκμηριώνει την επιλογή κάθε εργαλείου. Η ενότητα 4.3 περιγράφει τον σχεδιασμό της βάσης δεδομένων, μαζί με έναν συμπληρωματικό τρόπο διατήρησης κατάστασης που δεν στηρίζεται σε πεδία πινάκων. Η ενότητα 4.4 εξετάζει τον σχεδιασμό των prompts. Οι ενότητες 4.5 έως 4.7 αναλύουν την υλοποίηση καθενός από τους τρεις πράκτορες. Η ενότητα 4.8 παρουσιάζει το επίπεδο ενορχήστρωσης, το οποίο περιβάλλει τους τρεις πράκτορες και

διαχειρίζεται όσα δεν αποφασίζονται μέσα τους. Η ενότητα 4.9 περιγράφει την υλοποίηση του Open Learner Model. Η ενότητα 4.10 αφορά το frontend και τη διεπαφή χρήστη. Η ενότητα 4.11 συζητά τις δυσκολίες που προέκυψαν κατά την ανάπτυξη και πώς αντιμετωπίστηκαν. Η ενότητα 4.12 συνοψίζει το κεφάλαιο.

4.2 Τεχνολογική Στοιίβα

Η τεχνολογική στοιίβα οργανώνεται σε τέσσερα μέρη: το πλαίσιο ανάπτυξης του backend, τις βιβλιοθήκες που διαχειρίζονται τους πράκτορες και τις κλήσεις προς τα Μεγάλα Γλωσσικά Μοντέλα, τα εργαλεία αποθήκευσης δεδομένων, και τις βιβλιοθήκες του frontend.

Το backend βασίζεται στο FastAPI, ένα framework της Python για διαδικτυακά API. Κάθε ενέργεια του φοιτητή πυροδοτεί τουλάχιστον μία κλήση σε εξωτερικό μοντέλο και μία ή περισσότερες προσπελάσεις στη βάση, ενέργειες που διαρκούν αρκετά δευτερόλεπτα και στις οποίες ο server περιμένει απάντηση. Το FastAPI υποστηρίζει ασύγχρονη (async) εκτέλεση, ώστε να μπορεί να εξυπηρετεί άλλα αιτήματα όσο περιμένει. Τρέχει μέσω του unicorn, που ακολουθεί το ίδιο μοντέλο, και επικυρώνει κάθε εισερχόμενο αίτημα με το Pydantic, ώστε ένα κακοσχηματισμένο αίτημα να απορρίπτεται πριν φτάσει σε οποιαδήποτε λογική πράκτορα. Η επικοινωνία με το frontend περνά από προσαρμοσμένο CORS middleware. Επειδή τα δύο επίπεδα φιλοξενούνται σε διαφορετικές διευθύνσεις και η εφαρμογή δεν διαχειρίζεται ευαίσθητα δεδομένα πέρα από τα στοιχεία σύνδεσης, οι επιτρεπόμενες προελεύσεις δεν περιορίστηκαν, επιλογή που σε περιβάλλον πραγματικής χρήσης θα χρειαζόταν αυστηροποίηση.

Ο πυρήνας του πολυπρακτορικού συστήματος υλοποιείται με το LangGraph, μια βιβλιοθήκη φτιαγμένη ακριβώς για εφαρμογές σε μορφή γραφήματος κατάστασης. Επιτρέπει να δηλωθούν ρητά οι κόμβοι του γραφήματος, οι συνθήκες που οδηγούν από τον έναν στον άλλο, και η κοινή κατάσταση που μοιράζονται, δηλαδή η κεντρικοποιημένη, προβλέψιμη δρομολόγηση που περιγράφηκε στο Κεφάλαιο 3. Οι ίδιες οι κλήσεις προς τα μοντέλα γίνονται μέσω του langchain-core. Ως πάροχος χρησιμοποιείται το Grog, μέσω της langchain-grog, με κριτήριο την ταχύτητα, αφού το σύστημα θέτει ανώτατο όριο εξήντα δευτερολέπτων σε κάθε πλήρη κύκλο του γραφήματος και ένας κύκλος μπορεί να περιλαμβάνει περισσότερες από μία διαδοχικές κλήσεις.

Το μοντέλο που εξυπηρετεί όλες τις κλήσεις είναι το openai/gpt-oss-120b, δηλωμένο σε ένα και μόνο σημείο των ρυθμίσεων, με όλο το υπόλοιπο σύστημα, μαζί με το σύστημα αναφοράς της αξιολόγησης, να αναφέρεται σε αυτή τη δήλωση. Η τιμή μπορεί να παρακαμφθεί με μεταβλητή περιβάλλοντος, ώστε μια απόσυρση μοντέλου από τον πάροχο να απαιτεί αλλαγή σε μία μόνο γραμμή, όχι σε κάθε αρχείο του συστήματος. Ο λόγος που αυτό αποδείχθηκε χρήσιμο στην πράξη, και όχι μόνο σε θεωρία, περιγράφεται στην ενότητα 4.11, στην πρόκληση της εξάρτησης από εξωτερικό πάροχο. Το κλειδί πρόσβασης στο API διαβάζεται επίσης από μεταβλητή περιβάλλοντος, μέσω του python-dotenv.

Όλες οι κλήσεις απευθύνονται στο ίδιο μοντέλο. Αυτό που αλλάζει ανά περίπτωση είναι η θερμοκρασία (temperature), η παράμετρος που ελέγχει πόσο προβλέψιμη ή πόσο μεταβλητή είναι η έξοδος. Όσες κλήσεις επιστρέφουν μία κατηγορία από κλειστό σύνολο τρέχουν με θερμοκρασία μηδέν, αφού δύο ίδιες εισοδοί πρέπει να δίνουν το ίδιο αποτέλεσμα. Εδώ ανήκουν η επιλογή κατηγορίας του Debugger και οι εσωτερικές ταξινομήσεις του Mentor, καμία από τις οποίες δεν φτάνει ποτέ ως κείμενο στον φοιτητή. Όσες κλήσεις συνθέτουν πολλά στοιχεία μαζί ή παράγουν κείμενο που θα διαβάσει άνθρωπος τρέχουν με θερμοκρασία 0.1, αρκετά χαμηλή ώστε να μην εισάγεται ουσιαστική ασυνέπεια, αλλά όχι μηδενική, γιατί μια απολύτως ντετερμινιστική ρύθμιση θα οδηγούσε σε πανομοιότυπες διατυπώσεις σε κάθε παρόμοια περίπτωση. Εδώ ανήκουν η κρίση του

Assessor και η παραγωγή απαντήσεων από τον Mentor. Η ρύθμιση της θερμοκρασίας αντικατοπτρίζει επομένως την ίδια διάκριση που διατρέχει όλη την αρχιτεκτονική, ανάμεσα σε ότι πρέπει να είναι επαναλήψιμο και σε ότι απαιτεί κρίση.

Η ντετερμινιστική ανάλυση του κώδικα του φοιτητή, αυτή που κάνουν ο Debugger και ο Assessor, δεν στηρίζεται σε καμία εξωτερική βιβλιοθήκη στατικής ανάλυσης. Χρησιμοποιεί το άρθρωμα `ast` της ίδιας της Python, το οποίο δίνει πρόσβαση στη δενδρική αναπαράσταση που κατασκευάζει ο ίδιος ο διερμηνέας πριν από κάθε εκτέλεση. Το σύστημα βλέπει έτσι τον κώδικα με ακριβώς τους ίδιους κανόνες με τους οποίους θα τον έτρεχε η Python, χωρίς κάποιο ενδιάμεσο εργαλείο που θα μπορούσε να αποκλίνει από αυτούς ή να χρειάζεται δική του συντήρηση σε κάθε νέα έκδοση της γλώσσας.

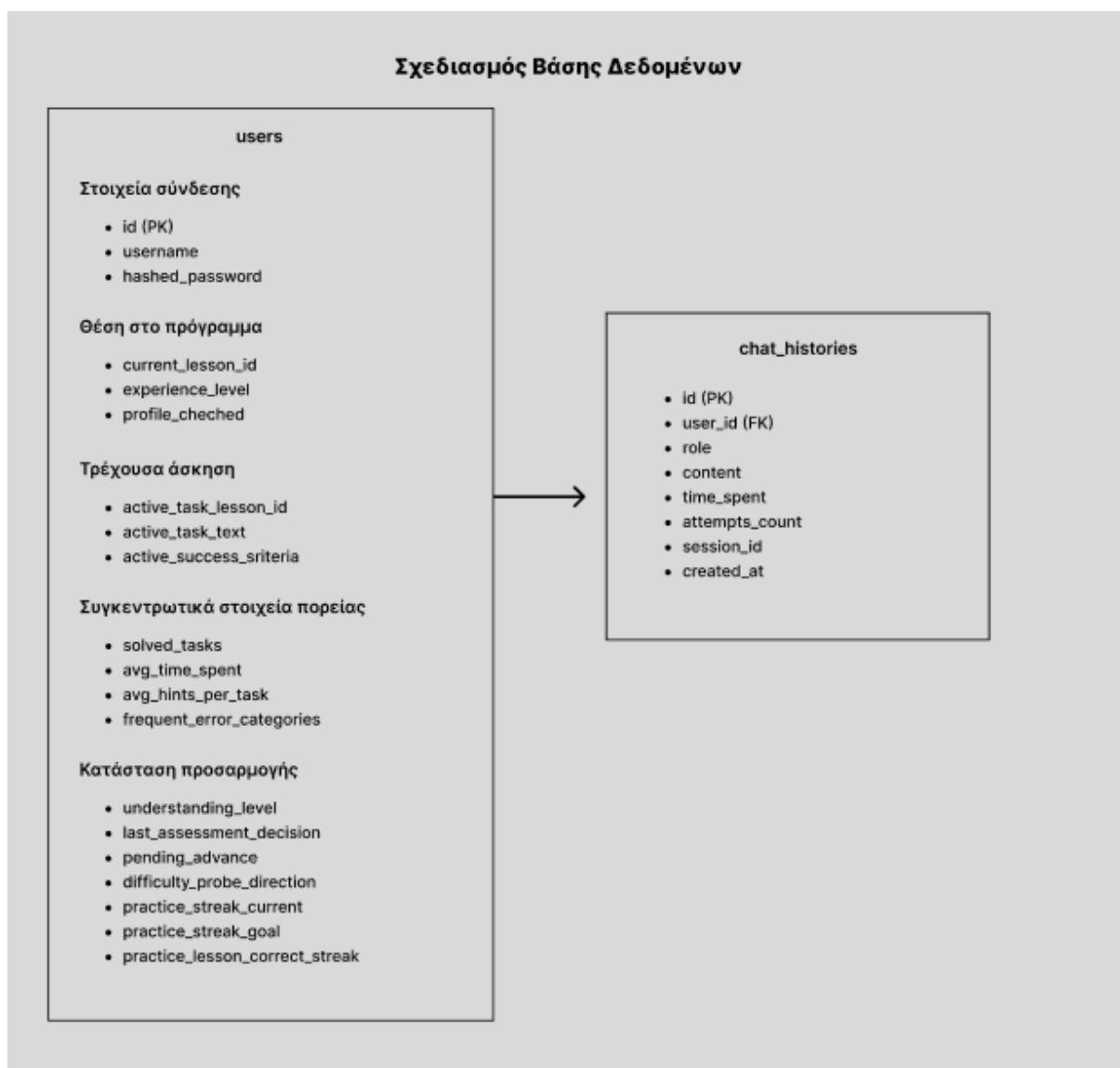
Η πρόσβαση στη βάση δεδομένων γίνεται μέσω του SQLAlchemy, σε ασύγχρονη λειτουργία. Στο περιβάλλον ανάπτυξης χρησιμοποιείται SQLite, μέσω του οδηγού `aiosqlite`, που δεν χρειάζεται δικό του server. Στο περιβάλλον παραγωγής, όπου η εφαρμογή φιλοξενείται στο Render με βάση δεδομένων Neon, χρησιμοποιείται PostgreSQL μέσω `asyncpg`, αφού το SQLite δεν είναι φτιαγμένο για ταυτόχρονη πρόσβαση πολλών χρηστών. Η μετάβαση ανάμεσα στα δύο δεν απαιτεί καμία αλλαγή στον κώδικα της εφαρμογής, η διεύθυνση σύνδεσης διαβάζεται από μεταβλητή περιβάλλοντος και προσαρμόζεται αυτόματα. Δεν χρησιμοποιείται ξεχωριστό εργαλείο διαχείρισης μεταβολών σχήματος, οι αλλαγές στα πεδία εφαρμόζονται με εντολές που ελέγχουν πρώτα αν μια στήλη υπάρχει ήδη, ώστε να μπορούν να τρέχουν σε κάθε εκκίνηση χωρίς σφάλμα. Η πιστοποίηση χρηστών γίνεται με κρυπτογράφηση κωδικών μέσω `bcrypt`, με τη βιβλιοθήκη `passlib`.

Το frontend αναπτύχθηκε σε React, με το Vite ως εργαλείο ανάπτυξης. Είναι μονοσέλιδη εφαρμογή, χωρίς βιβλιοθήκη δρομολόγησης και χωρίς ξεχωριστή βιβλιοθήκη διαχείρισης κατάστασης, αφού η μόνιμη κατάσταση ζει εξ ολοκλήρου στο backend και το frontend κρατά μόνο ότι αφορά την τρέχουσα οθόνη. Ο επεξεργαστής κώδικα μέσα στην εφαρμογή είναι το Monaco Editor, το ίδιο εργαλείο που χρησιμοποιεί το Visual Studio Code, με ενσωματωμένο χρωματισμό σύνταξης για Python. Η επικοινωνία με το backend γίνεται μέσω `axios`, η απόδοση μορφοποιημένου κειμένου στις απαντήσεις των πρακτόρων μέσω `react-markdown`, και τα εικονίδια της διεπαφής προέρχονται από τη βιβλιοθήκη `lucide-react`.

Το σύνολο αυτών των τεχνολογιών καλύπτει τρεις ανάγκες: την εξυπηρέτηση αιτημάτων με μεγάλους χρόνους αναμονής, την οργάνωση και εκτέλεση των τριών πρακτόρων, και τη διατήρηση της προόδου κάθε φοιτητή ανάμεσα σε διαφορετικές συνεδρίες. Οι επόμενες ενότητες δείχνουν πώς αξιοποιείται καθεμία στην υλοποίηση των επιμέρους μηχανισμών.

4.3 Σχεδιασμός Βάσης Δεδομένων

Η βάση δεδομένων έχει μόνο δύο πίνακες (Σχήμα 4.1), έναν για τους χρήστες και έναν για το ιστορικό των συνομιλιών. Ο αριθμός αυτός αρκεί επειδή μεγάλο μέρος της πληροφορίας του συστήματος δεν χρειάζεται να αποθηκευτεί εκεί. Το εκπαιδευτικό περιεχόμενο, δηλαδή η θεωρία, τα πρότυπα εκφωνήσεων και τα κριτήρια επιτυχίας κάθε ενότητας, βρίσκεται σε ένα αρχείο JSON, το οποίο διαβάζεται μία φορά στην εκκίνηση της εφαρμογής και δεν αλλάζει ποτέ. Είναι ίδιο για όλους τους φοιτητές, οπότε δεν έχει νόημα να μπει σε βάση δεδομένων. Στη βάση μένει μόνο ότι διαφέρει από φοιτητή σε φοιτητή.



Σχήμα 4.1. Σχεδιασμός βάσης δεδομένων

Ο πρώτος πίνακας κρατά τα στοιχεία κάθε φοιτητή. Τα πεδία του χωρίζονται εννοιολογικά σε πέντε ομάδες. Η πρώτη ομάδα είναι τα στοιχεία σύνδεσης, το όνομα χρήστη και ο κρυπτογραφημένος κωδικός, με το όνομα χρήστη μοναδικό και ευρετηριασμένο ώστε η αναζήτησή του να είναι γρήγορη. Η δεύτερη δείχνει πού βρίσκεται ο φοιτητής στο διδακτικό πρόγραμμα, δηλαδή σε ποια ενότητα είναι, ποιο είναι το επίπεδο εμπειρίας του, και αν έχει ήδη κατηγοριοποιηθεί σε επίπεδο εμπειρίας. Η τρίτη

κρατά την τρέχουσα άσκηση, το κείμενο της εκφώνησης, τα κριτήριά της, και την ενότητα στην οποία ανήκει. Η τέταρτη έχει συγκεντρωτικά στοιχεία της πορείας του, πόσες ασκήσεις έχει λύσει, πόσο χρόνο κάνει κατά μέσο όρο, πόσες υποδείξεις χρειάζεται συνήθως, ποια λάθη κάνει συχνότερα. Η πέμπτη κρατά την κατάσταση των μηχανισμών προσαρμογής: τελευταίο επίπεδο κατανόησης, τελευταία απόφαση του Assessor, ένδειξη εκκρεμούς προαγωγής, κατεύθυνση δοκιμαστικής αλλαγής δυσκολίας, μετρητές της Εξάσκησης.

Τρία πεδία δεν κρατούν μία απλή τιμή αλλά μια μικρή δομή σε μορφή JSON, γραμμένη μέσα σε πεδίο κειμένου: η λίστα κριτηρίων επιτυχίας, η λίστα συχνότερων λαθών, και οι μετρητές σωστών απαντήσεων ανά ενότητα στην λειτουργία της Εξάσκησης. Και τα τρία είναι μικρά κείμενα, ανήκουν σε έναν μόνο φοιτητή, και διαβάζονται πάντα μαζί με τα υπόλοιπα στοιχεία του. Επειδή στην ροή εκτέλεσης δεν απαιτείται κάποια αναζήτηση με βάση το περιεχόμενό τους, ξεχωριστοί πίνακες γι' αυτά θα πρόσθεταν πολυπλοκότητα χωρίς κανένα όφελος.

Ένα ξεχωριστό ζήτημα αφορά τους μέσους όρους. Ο μέσος χρόνος ανά άσκηση και ο μέσος αριθμός υποδείξεων δεν υπολογίζονται από την αρχή κάθε φορά που χρειάζονται. Αποθηκεύονται έτοιμοι και ενημερώνονται σταδιακά: κάθε φορά που ο φοιτητής λύνει σωστά μια άσκηση, ο νέος μέσος όρος προκύπτει από τον προηγούμενο και τη νέα μέτρηση, χωρίς να χρειάζεται να διαβαστεί ξανά ολόκληρο το ιστορικό, κάτι που θα γινόταν όλο και πιο αργό όσο αυτό μεγαλώνει. Η τεχνική αυτή λέγεται αποκανονικοποίηση (denormalization). Σημαίνει ότι η ίδια πληροφορία υπάρχει δύο φορές, μία σε ακατέργαστη μορφή και μία ήδη υπολογισμένη, με αντάλλαγμα την ταχύτητα.

Ο δεύτερος πίνακας κρατά το ιστορικό των συνομιλιών. Κάθε γραμμή του είναι ένα μήνυμα, είτε του φοιτητή είτε του συστήματος, συνδεδεμένο με τον φοιτητή στον οποίο ανήκει μέσω ξένου κλειδιού (foreign key), με πληροφορίες όπως το πότε γράφτηκε, πόσος χρόνος αφιερώθηκε, και πόσες προσπάθειες έγιναν, όπου αυτά έχουν νόημα. Κάθε φορά που ο φοιτητής συνδέεται, το σύστημα παράγει έναν αριθμό από τη χρονική στιγμή της σύνδεσης, και όλα τα μηνύματα εκείνης της φοράς παίρνουν αυτόν τον αριθμό. Μια συνεδρία δεν αποθηκεύεται δηλαδή ως ξεχωριστή οντότητα, είναι απλώς όσα μηνύματα μοιράζονται τον ίδιο αριθμό. Ο πίνακας έχει ευρετήρια (indexes) στον φοιτητή και στη συνεδρία, ώστε η αναζήτηση να μη χρειάζεται να διατρέξει όλες τις γραμμές.

Δεν καταγράφεται όμως ότι κάνει ο φοιτητής. Στο ιστορικό μπαίνουν μόνο τα μηνύματα της κύριας ροής διδασκαλίας, το καλωσόρισμα κάθε σύνδεσης και ο διάλογος γύρω από θεωρία και ασκήσεις. Οι υποβολές στην Εξάσκηση και στην Αξιολόγηση Κώδικα δεν αποθηκεύονται, γιατί δεν επηρεάζουν την πρόοδο στα μαθήματα και δεν χρειάζεται να ανακτηθούν αργότερα. Από την Εξάσκηση κρατιέται μόνο το αποτέλεσμα, δηλαδή οι μετρητές σωστών απαντήσεων, όχι τα ίδια τα μηνύματα.

Εκτός από τα πεδία των δύο πινάκων, υπάρχει κι ένας δεύτερος τρόπος με τον οποίο το σύστημα θυμάται πράγματα. Κάθε μήνυμα που γράφει ο Mentor κλείνει με ειδικά σύμβολα μέσα σε αγκύλες, τα οποία λένε τι ακριβώς έγινε σε εκείνο το μήνυμα. Το '[HINT]' σημαίνει ότι δόθηκε υπόδειξη. Το '[AWAITING_QUESTIONS]' ότι το σύστημα περιμένει τυχόν απορίες πριν δώσει άσκηση. Τα '[ASSESSMENT:ADVANCE]', '[ASSESSMENT:REPEAT]' και '[ASSESSMENT:SUPPORT]' δείχνουν ποια ήταν η απόφαση του Assessor. Το '[THEORY_DIFFICULTY:easy:2]' ότι παρουσιάστηκε η θεωρία μιας ενότητας, εδώ της δεύτερης, σε εύκολο επίπεδο. Τα σύμβολα αυτά αποθηκεύονται μαζί με το υπόλοιπο κείμενο του μηνύματος, στο ίδιο πεδίο.

Το ζήτημα αυτό, η διατήρηση και η ανάκτηση της κατάστασης μιας συνομιλίας ανάμεσα σε διαδοχικές εκφράσεις, είναι γνωστό στη βιβλιογραφία των συνομιλιακών πρακτόρων (chatbots) ως διαχείριση διαλόγου (dialog management), με προσεγγίσεις που πηγάζουν από ρητά ορισμένες δομές

κατάστασης έως μοντέλα που τη συνάγουν από δεδομένα [44]. Η προσέγγιση εδώ ανήκει στην πρώτη κατηγορία, με τη διαφορά ότι η κατάσταση δεν κρατιέται σε ξεχωριστή δομή αλλά μέσα στο ίδιο το ιστορικό.

Ο λόγος είναι απλός. Πολλά από όσα πρέπει να θυμάται το σύστημα δεν αφορούν τον φοιτητή γενικά, αφορούν τη συγκεκριμένη στιγμή της συνομιλίας (πχ πόσες υποδείξεις έχουν δοθεί σε αυτή την άσκηση, αν έχει ήδη παρουσιαστεί η θεωρία, αν εκκρεμεί απάντηση σε ερώτηση). Πρόκειται για δεδομένα που αλλάζουν συνεχώς. Για να μουν σε πεδία, θα χρειαζόνταν είτε πολλές επιπλέον στήλες που θα ενημερώνονταν διαρκώς, είτε ένας ολόκληρος νέος πίνακας. Αντί γι' αυτό, το σύστημα τα ξαναβρίσκει όποτε τα χρειάζεται, διαβάζοντας το ιστορικό προς τα πίσω και ψάχνοντας τα σύμβολα. Το ιστορικό υπάρχει ήδη αποθηκευμένο, οπότε χρησιμεύει ταυτόχρονα και ως καταγραφή του τι έχει ήδη συμβεί.

Ο σχεδιασμός στηρίζεται σε μια απλή διάκριση, η οποία ακολουθεί το εξής σκεπτικό. Ότι περιγράφει τον φοιτητή συνολικά και πρέπει να είναι αμέσως διαθέσιμο μπαίνει σε πεδίο πίνακα, όπως το `avg_hints_per_task`. Ο λόγος δεν είναι απλά η άμεση διαθεσιμότητα καθαυτή, είναι το εύρος της σάρωσης που θα απαιτούνταν χωρίς αυτό. Το `avg_hints_per_task` αφορά όλη τη διάρκεια του προγράμματος, άρα θα σήμαινε σάρωση ολόκληρου του ιστορικού κάθε φορά, όσο αυτό μεγαλώνει. Αντίθετα, ότι περιγράφει την πορεία μιας συγκεκριμένης συνομιλίας, όπως πόσες υποδείξεις έχουν δοθεί στην εκάστοτε την άσκηση, έχει από μόνο του ένα φυσικό, στενό όριο σάρωσης, μέχρι το προηγούμενο σύμβολο μετάβασης, όχι ολόκληρο το ιστορικό. Εκεί το κόστος της επανάληψης παραμένει μικρό, γι' αυτό δεν χρειάζεται δικό του πεδίο.

4.4 Σχεδιασμός Prompts

Το σύστημα καλεί Μεγάλα Γλωσσικά Μοντέλα σε δώδεκα διαφορετικά σημεία. Τα prompts τους δεν γράφτηκαν το καθένα ανεξάρτητα, αλλά ακολουθούν κοινές αρχές, οι οποίες πηγάζουν από την ίδια ανάγκη: το μοντέλο να λαμβάνει συγκεκριμένα δεδομένα και σαφές πλαίσιο και να επιστρέφει απάντηση σε μορφή που το υπόλοιπο σύστημα μπορεί να αξιοποιήσει αυτόματα. Οι αρχές αυτές δεν επινοήθηκαν εδώ. Ο σχεδιασμός prompts αποτελεί πλέον αντικείμενο συστηματικής μελέτης, με τεχνικές που περιλαμβάνουν, μεταξύ άλλων, τον ρητό καθορισμό του πλαισίου και της επιθυμητής μορφής της εξόδου, καθώς και την προσαρμογή των οδηγιών στις απαιτήσεις της εκάστοτε εργασίας [43].

Στους δύο πράκτορες που κρίνουν κώδικα, το prompt έχει τέσσερα μέρη σε σταθερή σειρά. Πρώτα ο ρόλος, τι είναι ο πράκτορας και με βάση τι κρίνει. Μετά τα δεδομένα της συγκεκριμένης περίπτωσης, δηλαδή εκφώνηση, κριτήρια, κώδικας του φοιτητή. Μετά τα ευρήματα των ντετερμινιστικών ελέγχων. Τέλος οι οδηγίες για την ακριβή μορφή της απάντησης.

Κάτι που επαναλαμβάνεται και στους δύο είναι ο τρόπος που παρουσιάζονται τα ευρήματα. Δεν δίνονται ως απλή πληροφορία, χαρακτηρίζονται ρητά μέσα στο ίδιο το κείμενο ως δεδομένα. Στον Debugger τα δομικά στοιχεία φέρουν την ένδειξη ότι προέρχονται από στατική ανάλυση (Πράρτημα Α, Σχήμα Α.1), στον Assessor τα αντικειμενικά ευρήματα φέρουν την ένδειξη ότι προέρχονται από ντετερμινιστικούς ελέγχους. Ο σκοπός είναι η κρίση του μοντέλου να στηρίζεται σε αυτά, όχι σε δικές του εκτιμήσεις για την κατάσταση του κώδικα.

Η μορφή της απάντησης καθορίζεται με την ίδια ακρίβεια. Ζητείται από το μοντέλο να απαντήσει σε συγκεκριμένες γραμμές με σταθερές ετικέτες, χωρίς κανένα άλλο κείμενο πριν ή μετά. Ο Debugger επιστρέφει κατάσταση, κατηγορία, εξήγηση. Ο Assessor επιστρέφει αν ο κώδικας είναι σωστός, την απόφαση προόδου, το επίπεδο κατανόησης, το κείμενο ανατροφοδότησης. Οι απαντήσεις αναλύονται

με κανονικές εκφράσεις (regular expressions) που ψάχνουν ακριβώς αυτές τις ετικέτες, ώστε τα πεδία να περάσουν αυτόματα στην κατάσταση του συστήματος. Αν η αναζήτηση δεν βρει τη μορφή αυτή, χρησιμοποιείται το ντετερμινιστικό εναλλακτικό μονοπάτι. Ο περιορισμός της εξόδου σε αυστηρά δομημένη μορφή είναι τεκμηριωμένη τεχνική για την αξιοποίηση αποτελεσμάτων LLM από αυτοματοποιημένα συστήματα [24].

Ο Mentor αντιμετωπίζει διαφορετικό πρόβλημα. Η έξοδος του δεν είναι ετικέτα, είναι κείμενο που θα διαβάσει άνθρωπος, οπότε δεν μπορεί να περιοριστεί σε κλειστή μορφή. Γι' αυτό υπάρχει μια κοινή συνάρτηση παραγωγής απάντησης, η οποία δέχεται πέντε στοιχεία: την περιγραφή της τρέχουσας κατάστασης, μια ενδεικτική φράση ως παράδειγμα ύφους και όχι κείμενο προς αντιγραφή, τον επιθυμητό τόνο, τι πρέπει να αποφευχθεί σε αυτή τη συγκεκριμένη περίπτωση, και ένδειξη συντομίας όταν η απάντηση πρέπει να είναι μία ή δύο προτάσεις. Ο προγραμματιστής περιγράφει τι συμβαίνει, το μοντέλο διατυπώνει την απάντηση. Μια παραλλαγή της ίδιας συνάρτησης παράγει, με μία μόνο κλήση, δύο φράσεις μαζί, μια εισαγωγική πριν από το περιεχόμενο (π.χ. πριν τη θεωρία) και μια καταληκτική μετά από αυτό (π.χ. ερώτηση αν υπάρχουν απορίες), εξοικονομώντας μία επιπλέον κλήση σε σχέση με το να παραχθούν χωριστά.

Η περιγραφή της κατάστασης δεν είναι σταθερή. Μέσα στον κόμβο του Mentor υπάρχει μια εκτενής αλυσίδα αμοιβαία αποκλειόμενων συνθηκών, η οποία εξετάζει τι ακριβώς συμβαίνει, αν ο φοιτητής μόλις συνδέθηκε, αν ζητά άσκηση, αν έκανε θεωρητική ερώτηση, αν υπέβαλε λάθος κώδικα, αν ζητά να προχωρήσει, και ούτω καθεξής. Κάθε κλάδος παράγει τη δική του περιγραφή, και το μοντέλο λαμβάνει μόνο εκείνη που αντιστοιχεί στην τρέχουσα στιγμή.

Στην περίπτωση των υποδείξεων, το prompt συντίθεται από περισσότερα κομμάτια, τα οποία προστίθενται μόνο όταν ισχύουν. Αν ο φοιτητής έχει καταγεγραμμένα συχνά λάθη, προστίθεται μια γραμμή με τα τρία συχνότερα, σε αναγνώσιμη μορφή. Αν έχει ήδη δοθεί μία υπόδειξη, προστίθεται οδηγία η επόμενη να είναι λίγο πιο συγκεκριμένη. Αν έχουν δοθεί δύο ή περισσότερες, προστίθεται οδηγία μετάβασης σε άμεση εξήγηση. Αν ο φοιτητής χρειάζεται κατά μέσο όρο πολλές υποδείξεις, προστίθεται οδηγία για πιο αναλυτική κατεύθυνση, ενώ αν συνήθως αρκείται σε μία, προστίθεται οδηγία να παραμείνει γενική. Το τελικό prompt είναι έτσι διαφορετικό για κάθε φοιτητή και κάθε στιγμή, με τις επιλογές να προκύπτουν από τα αποθηκευμένα δεδομένα, όχι από το ίδιο το μοντέλο.

Το prompt που απαντά σε θεωρητικές ερωτήσεις έχει τη δική του ιδιαιτερότητα. Το μοντέλο λαμβάνει ως βάση γνώσης μόνο την ύλη που έχει ήδη διδαχθεί, μαζί με τους τίτλους των επόμενων εννοιών και οδηγία, αν η ερώτηση αφορά κάτι από αυτές, να απαντήσει ότι θα καλυφθεί αργότερα. Περιέχει επίσης διευκρινίσεις πάνω σε συγκεκριμένα σημεία της ίδιας της γλώσσας, και οδηγία να απαντηθεί ακριβώς αυτό που ρωτήθηκε και όχι το γενικό θέμα, με το πολύ ένα σύντομο παράδειγμα, και η καταληκτική ερώτηση επιβεβαίωσης να εναλλάσσεται ανάμεσα σε διατυπώσεις ώστε να μην επαναλαμβάνεται πάντα η ίδια.

Στη λειτουργία της Αξιολόγησης Κώδικα, ο ίδιος πράκτορας λαμβάνει σχεδόν αντίθετη οδηγία. Ενώ στις υποδείξεις των ασκήσεων ζητείται ρητά να μην αποκαλυφθεί το πρόβλημα, εδώ δηλώνεται ότι η απάντηση μπορεί να είναι συγκεκριμένη, αφού ο φοιτητής ζήτησε ανάλυση και όχι υπόδειξη. Χρησιμοποιούνται μάλιστα δύο διαφορετικά prompts, ένα για κώδικα που φαίνεται σωστός και ένα για κώδικα με πρόβλημα. Ο παιδαγωγικός περιορισμός δεν είναι δηλαδή ιδιότητα του πράκτορα, είναι ιδιότητα του πλαισίου μέσα στο οποίο καλείται.

Ένα ξεχωριστό είδος prompt χρησιμοποιείται όταν ο Mentor δεν παράγει κείμενο αλλά αναγνωρίζει τι ζητά ο φοιτητής. Το μοντέλο επιστρέφει μία μόνο λέξη-κλειδί: αν ζητά άσκηση, αν κάνει ερώτηση

θεωρίας, αν θέλει να προχωρήσει σε επόμενη ενότητα, αν ζητά βοήθεια με τον κώδικά του, ή τίποτα από αυτά. Ο κατάλογος των κατηγοριών δεν είναι σταθερός. Αν ο φοιτητής βρίσκεται στη φάση της θεωρίας, οι διαθέσιμες κατηγορίες είναι διαφορετικές από όσες δίνονται όταν έχει ήδη ξεκινήσει άσκηση, και το prompt δηλώνει σε ποια φάση βρίσκεται, ώστε το μοντέλο να διαλέγει μόνο ανάμεσα σε ότι μπορεί πράγματι να ισχύει εκείνη τη στιγμή.

Κάθε κατηγορία συνοδεύεται από παραδείγματα φράσεων και διευκρινίσεις για περιπτώσεις που μπερδεύονται εύκολα: ότι το «κατάλαβα, αλλά» ακολουθούμενο από ερώτηση είναι ερώτηση θεωρίας και όχι αίτημα νέας άσκησης, ή ότι ένα «όχι» στην ερώτηση αν υπάρχουν απορίες σημαίνει ετοιμότητα για άσκηση. Δηλώνεται επίσης ότι οι απαντήσεις πρέπει να αναγνωρίζονται ακόμα κι όταν είναι γραμμένες με λατινικούς χαρακτήρες ή με ορθογραφικά λάθη, κάτι συνηθισμένο σε ελληνόφωνους χρήστες. Η λέξη που επιστρέφεται αντιστοιχίζεται στη συνέχεια με το κλειστό σύνολο έγκυρων κατηγοριών, και αν δεν ανήκει σε αυτό χρησιμοποιείται η ουδέτερη κατηγορία. Αν η κλήση δεν ολοκληρωθεί, η ταξινόμηση γίνεται με αναζήτηση λέξεων-κλειδιών στο μήνυμα.

Σε δύο από αυτές τις ταξινομήσεις, εκείνη που εξετάζει αν ο φοιτητής δέχεται να προχωρήσει στην επόμενη ενότητα και εκείνη που κρίνει το αρχικό του επίπεδο εμπειρίας, το prompt περιλαμβάνει έναν κατάλογο παραδειγμάτων, συγκεκριμένα μηνύματα μαζί με τη σωστή κατηγορία για το καθένα. Στην ταξινόμηση εμπειρίας τα παραδείγματα ξεπερνούν τα δεκαπέντε και καλύπτουν και οριακές περιπτώσεις, όπως ένα γυμνό «ναι» που δεν διευκρινίζει αν ο φοιτητής έχει εμπειρία ή όχι, και γι' αυτό κατατάσσεται σε ξεχωριστή, ασαφή κατηγορία. Η τεχνική λέγεται few-shot prompting, δίνονται στο μοντέλο λίγα έτοιμα παραδείγματα μέσα στο ίδιο το prompt αντί να επανεκπαιδευτεί για τη συγκεκριμένη εργασία [17].

Ο σχεδιασμός ενός prompt αφορά και τα δεδομένα που θα του δοθούν. Χαρακτηριστικό παράδειγμα είναι η περίληψη που βλέπει ο φοιτητής όταν επιστρέφει σε νέα συνεδρία. Το ιστορικό δεν δίνεται ολόκληρο. Επιλέγονται τα δώδεκα τελευταία μηνύματα, αφαιρούνται τα εσωτερικά σύμβολα, εξαιρούνται οι υποβολές κώδικα, και κάθε μήνυμα περικόπτεται σε συγκεκριμένο μήκος, ώστε το μοντέλο να λαμβάνει μια σύντομη, καθαρή εκδοχή της συνομιλίας. Το prompt αυτό έχει και τις πιο αναλυτικές οδηγίες τεκμηρίωσης σε όλο το σύστημα: να στηριχθεί μόνο σε ότι φαίνεται καθαρά στη συνομιλία, να αναφέρει μια άσκηση ως ολοκληρωμένη μόνο εφόσον υπάρχει ρητή επιβεβαίωση επιτυχίας μέσα στα μηνύματα, να μην παραθέσει αυτούσια λόγια του φοιτητή, και να μην αναφέρει ότι το σύστημα διαβάξει ιστορικό, ώστε η περίληψη να διαβάζεται σαν ανάμνηση της συζήτησης και όχι σαν αναφορά δεδομένων.

Ο βαθμός στον οποίο απαιτείται συγκεκριμένη μορφή διαφέρει ανάλογα με τη χρήση. Στον Debugger και στον Assessor η δομή είναι απαραίτητη για να περάσουν τα πεδία στην κατάσταση του συστήματος. Στη συνάρτηση εισαγωγικής και καταληκτικής φράσης, η δομή εξυπηρετεί μόνο τον διαχωρισμό των δύο. Αν αυτός δεν γίνει, όλο το κείμενο χρησιμοποιείται ως εισαγωγή και το κλείσιμο παραλείπεται, μια εισαγωγική φράση παραμένει αξιοποιήσιμη ακόμα και χωρίς τον διαχωρισμό.

Ορισμένες απαιτήσεις επηρεάζουν άμεσα την εμπειρία του φοιτητή, γι' αυτό υποστηρίζονται και από αντίστοιχο έλεγχο στον κώδικα. Το prompt ζητά μία ή δύο προτάσεις όταν χρειάζεται σύντομία, και υπάρχει συνάρτηση που κρατά τις δύο πρώτες προτάσεις όποτε χρειαστεί. Ζητά αποκλειστικά ελληνικά, και υπάρχει φίλτρο που αφαιρεί χαρακτήρες από άλλα αλφάβητα. Αφαιρούνται επίσης τυχόν ίχνη εσωτερικού συλλογισμού, αν βρεθούν μέσα στην έξοδο.

Ως προς τη μορφή τους, τα prompts συντίθενται σχεδόν όλα ως ένα ενιαίο κείμενο. Μόνο η τελευταία, γενική διαδρομή του Mentor, εκείνη που καλύπτει ό,τι δεν ταιριάζει σε κανέναν ειδικό κλάδο,

χρησιμοποιεί τη δομή δύο μηνυμάτων της βιβλιοθήκης, ένα μήνυμα συστήματος με τους γενικούς παιδαγωγικούς κανόνες και ένα ξεχωριστό μήνυμα χρήστη.

Συνολικά, ο σχεδιασμός των prompts ακολουθεί την ίδια διάκριση που διατρέχει όλο το σύστημα. Το μοντέλο αναλαμβάνει ότι απαιτεί ερμηνεία και διατύπωση σε φυσική γλώσσα, ο κώδικας γύρω του ετοιμάζει τα δεδομένα που θα λάβει και αξιοποιεί τη μορφή όσων επιστρέφει.

4.5 Λογική Καθοδήγησης (Mentoring Agent)

Ο Mentor είναι το εκτενέστερο κομμάτι του συστήματος, περίπου τετραπλάσιο σε μέγεθος από τον Debugger ή τον Assessor. Αυτό οφείλεται σε μια δομική διαφορά στον ρόλο του. Ο Debugger και ο Assessor ενεργοποιούνται πάντα στην ίδια κατάσταση, όταν έχει υποβληθεί κώδικας, και παράγουν πάντα το ίδιο είδος αποτελέσματος. Ο Mentor ενεργοποιείται σε κάθε μήνυμα του φοιτητή, ότι κι αν περιέχει αυτό, και πρέπει να αποκριθεί κατάλληλα σε καθεμία από τις πολλές πιθανές περιστάσεις.

Είναι επίσης ο πράκτορας που κάνει τις περισσότερες κλήσεις μοντέλου, δέκα από τις δώδεκα συνολικά, και ο μόνος του οποίου η έξοδος φτάνει αυτούσια στον φοιτητή. Κάθε εξήγηση θεωρίας, κάθε υπόδειξη πέρα από το γνήσιο συντακτικό σφάλμα, κάθε επιβεβαίωση επιτυχίας και κάθε περίληψη προηγούμενης συνεδρίας διατυπώνεται από το μοντέλο. Ο ντετερμινιστικός κώδικας που περιγράφεται στη συνέχεια δεν αντικαθιστά αυτές τις κλήσεις. Απαντά στα δύο ερωτήματα που προηγούνται καθεμιάς τους: τι ζητά ο φοιτητής αυτή τη στιγμή, και τι έχει ήδη συμβεί μέχρι τώρα στη συνομιλία, ώστε η κλήση που θα γίνει να είναι η σωστή και να συνοδεύεται από ακριβή πληροφορία.

Αναφορικά με το πρώτο ερώτημα, πριν κληθεί το μοντέλο μια σειρά απλών ελέγχων προσπαθεί να καταλάβει την πρόθεση του φοιτητή από τη μορφή του μηνύματος. Ένα κενό μήνυμα κατατάσσεται στην ουδέτερη κατηγορία. Ένα μήνυμα με μπλοκ κώδικα ή με την τυποποιημένη φράση υποβολής αναγνωρίζεται αμέσως ως αίτημα βοήθειας με τον κώδικα. Ένα μήνυμα με λέξεις όπως θεωρία, θύμισέ μου ή ξαναπές αναγνωρίζεται ως ερώτηση θεωρίας. Αν ο φοιτητής βρίσκεται ήδη σε άσκηση και δηλώνει ρητά ότι δεν ξέρει πώς να ξεκινήσει, κατατάσσεται απευθείας ως αίτημα βοήθειας. Σε καμία από αυτές τις περιπτώσεις δεν παράγεται η τελική απάντηση προς τον φοιτητή, μόνο μια κατηγορία πρόθεσης, η οποία στη συνέχεια καθορίζει ποιο prompt θα χρησιμοποιηθεί για να γραφτεί η πραγματική απάντηση, με τον ίδιο τρόπο που τα ευρήματα του Debugger καθορίζουν, χωρίς να αντικαθιστούν, την κρίση του Assessor.

Οι έλεγχοι αυτοί καλύπτουν μόνο ότι είναι μονοσήμαντο. Κάθε μήνυμα του οποίου η πρόθεση δεν προκύπτει με βεβαιότητα από τη μορφή του προωθείται στο μοντέλο, όπου καταλήγουν οι απρόβλεπτες διατυπώσεις, οι φράσεις που συνδυάζουν δύο προθέσεις μαζί, και ότι κανένας σταθερός κανόνας δεν θα αναγνώριζε. Ο καταμερισμός δεν είναι τυχαίος: το σταθερό κομμάτι αναλαμβάνει ότι είναι βέβαιο, το μοντέλο ότι απαιτεί ερμηνεία.

Πριν από την κλήση του μοντέλου γίνεται και ένας έλεγχος για ακατάληπτο μήνυμα, δηλαδή τυχαίο πάτημα πλήκτρων που δεν σημαίνει τίποτα. Ένα μήνυμα θεωρείται ακατάληπτο αν είναι ενός μόνο χαρακτήρα, αν δεν περιέχει κανένα φωνήεν, αν αποτελείται από το ίδιο γράμμα επαναλαμβανόμενο τρεις ή περισσότερες φορές, ή αν περιέχει τέσσερα και πλέον συνεχόμενα σύμφωνα. Χωρίς αυτόν τον έλεγχο, ένα τυχαίο πάτημα θα μπορούσε να ταξινομηθεί ως ερώτηση και να προκαλέσει μια εκτενή, εντελώς άσχετη εξήγηση.

Το δεύτερο ερώτημα συντελεί το επόμενο μεγάλο κομμάτι της υλοποίησης, οπου αφορά την ανασύσταση της κατάστασης της συνομιλίας. Ο Mentor διαθέτει μια ομάδα μικρών συναρτήσεων,

καθεμία από τις οποίες απαντά σε ένα συγκεκριμένο ερώτημα διαβάζοντας το ιστορικό προς τα πίσω και ψάχνοντας τα σύμβολα που περιγράφηκαν στην ενότητα 4.3. Ελέγχουν αν έχει ήδη παρουσιαστεί το οπτικό βοήθημα της τρέχουσας ενότητας, αν έχει ήδη δοθεί η τρέχουσα άσκηση, αν έχει παρουσιαστεί η θεωρία της νέας ενότητας μετά την τελευταία προαγωγή, αν η προηγούμενη άσκηση έκλεισε με απόφαση επανάληψης, ποια ήταν η τελευταία απόφαση του Assessor. Οι απαντήσεις αυτές δεν καθορίζουν τι θα πει ο Mentor, καθορίζουν ποια περιγραφή κατάστασης θα λάβει το μοντέλο, ώστε να μην ξαναπεί κάτι που έχει ήδη ειπωθεί.

Καμία από αυτές τις συναρτήσεις δεν σαρώνει ολόκληρο το ιστορικό. Σταματούν μόλις συναντήσουν ένα σύμβολο που σηματοδοτεί όριο. Η συνάρτηση που ελέγχει αν έχει ήδη δοθεί η τρέχουσα άσκηση, για παράδειγμα, διατρέχει τα μηνύματα προς τα πίσω και σταματά με αρνητική απάντηση μόλις συναντήσει σύμβολο προαγωγής ή επανάληψης, αφού από εκείνο το σημείο και πριν τα μηνύματα αφορούν προηγούμενη άσκηση. Το ερώτημα δεν είναι ποτέ αν κάτι συνέβη κάποια στιγμή, είναι αν συνέβη μετά την τελευταία μετάβαση.

Οι υποδείξεις που παρέχει ο Mentor δεν παράγονται με τον ίδιο τρόπο σε όλες τις περιπτώσεις. Όταν το εύρημα αφορά σημασιολογικό πρόβλημα ή μη ικανοποίηση κριτηρίων, η υπόδειξη συντίθεται από το μοντέλο, με κλιμάκωση μέσα στο ίδιο το prompt ανάλογα με το πλήθος των υποδείξεων που έχουν ήδη δοθεί. Αυτή είναι και η συνηθέστερη περίπτωση, αφού τα περισσότερα λάθη σε συντακτικά έγκυρο κώδικα είναι σημασιολογικά. Όταν όμως ο κώδικας έχει γνήσιο συντακτικό λάθος, το μοντέλο παρακάμπτεται, με την ίδια, ήδη τεκμηριωμένη εξαίρεση της ενότητας 3.4.2: όταν δεν υπάρχει καμία ερμηνευτική αμφισημία προς επίλυση, δεν υπάρχει και κρίση να ανατεθεί σε μοντέλο. Αυτό δεν είναι λιγότερη αυτονομία, είναι η ίδια αυτονομία σε λειτουργία. Η ενότητα 3.8 εντόπισε την αυτονομία ενός πράκτορα ως την κρίση για το τι θα γίνει μέσα στο δικό του βήμα, όχι ως την υποχρεωτική χρήση μοντέλου σε κάθε βήμα ανεξαρτήτως. Ο Mentor εδώ ασκεί ακριβώς αυτή την κρίση, αναγνωρίζει ότι δεν υπάρχει τίποτα να ερμηνευτεί, και αποφασίζει μόνος του να μην καλέσει το μοντέλο, όχι επειδή του λείπει η δυνατότητα, αλλά επειδή η κατάσταση δεν τη χρειάζεται. Ο Mentor διαλέγει τότε μια προδιατυπωμένη εξήγηση από ένα σύνολο έτοιμων κειμένων, αντιστοιχίζοντας το τεχνικό μήνυμα του parser σε παιδαγωγική διατύπωση. Η χρήση σύνταξης if/else από άλλη γλώσσα οδηγεί σε εξήγηση για την ύπαρξη ειδικής λέξης-κλειδιού στην Python. Ένα μήνυμα για εσοχή οδηγεί σε εξήγηση για την ευαισθησία της γλώσσας στα κενά μπροστά από κάθε γραμμή. Η χρήση αριθμητικής τιμής ως παραμέτρου σε ορισμό συνάρτησης οδηγεί σε εξήγηση της διαφοράς ανάμεσα σε ορισμό και κλήση, με ολοκληρωμένο παράδειγμα σωστής μορφής. Ένα γενικό συντακτικό σφάλμα οδηγεί σε προτροπή ελέγχου του σημείου, με τον αριθμό γραμμής, όταν υπάρχει, ενσωματωμένο στην εξήγηση.

Επειδή το κείμενο αυτό είναι σταθερό ανά κατηγορία, ένας φοιτητής που επαναλαμβάνει το ίδιο λάθος θα λάμβανε κατά λέξη την ίδια απάντηση. Γι' αυτό οι έτοιμες εξηγήσεις δίνονται μέσα από ένα περίβλημα που προσαρμόζει τη διατύπωση ανάλογα με το πλήθος των προηγούμενων υποδείξεων, προσθέτοντας μια φράση που αναγνωρίζει την επιμονή του προβλήματος και, στην τρίτη επανάληψη, μια συγκεκριμένη πρόταση δράσης, να γράψει ο φοιτητής μόνο τη γραμμή που λείπει και να εκτελέσει ξανά. Η κλιμάκωση εδώ ξεκινά από διαφορετική αφετηρία σε σχέση με τη διαδρομή του μοντέλου: εκεί οδηγεί σταδιακά από την έμμεση προς την άμεση εξήγηση, ενώ εδώ η εξήγηση είναι ήδη άμεση από την πρώτη φορά, οπότε αυτό που προστίθεται είναι συγκεκριμένη καθοδήγηση για την επόμενη κίνηση.

Κάθε prompt του Mentor που πρόκειται να αναφερθεί στον κώδικα συνοδεύεται από τα δομικά ευρήματα της στατικής ανάλυσης, σε πέντε σημεία της υλοποίησης. Συγκεκριμένα, στη συνάρτηση

που παράγει την ίδια την υπόδειξη όταν ο κώδικας έχει πρόβλημα, στις δύο συναρτήσεις που παράγουν μια σύντομη εισαγωγική φράση πριν παρουσιαστεί η υπόδειξη, ανάλογα με το αν πρόκειται για πρώτη ή επόμενη προσπάθεια, και στις δύο συναρτήσεις που παράγουν το μήνυμα επιβεβαίωσης όταν η λύση είναι σωστή, ανάλογα με το αν χρειάστηκαν πολλές προσπάθειες ή όχι.

Δύο μικρότεροι μηχανισμοί φροντίζουν ώστε η εσωτερική ορολογία του συστήματος να μη φτάνει ποτέ στον φοιτητή. Ο πρώτος είναι ένας πίνακας αντιστοίχισης που μετατρέπει τους εσωτερικούς κωδικούς των κατηγοριών σφαλμάτων σε αναγνώσιμη ελληνική περιγραφή, όπως «χρήση μεταβλητής πριν οριστεί» ή «ξεχασμένο print», και χρησιμοποιείται όταν τα συχνότερα λάθη του φοιτητή ενσωματώνονται σε prompt. Ο δεύτερος αφαιρεί από το κείμενο ανατροφοδότησης του Assessor τυχόν εσωτερικές ετικέτες σε αγκύλες, πριν αυτό δοθεί στο μοντέλο ως πληροφορία για τη σύνθεση της υπόδειξης.

Η δομή του κόμβου του Mentor ξεκινά με δύο πρώιμες εξόδους, μία για καθεμία από τις δύο συμπληρωματικές λειτουργίες του συστήματος, οι οποίες δεν χρειάζεται να περάσουν από την κύρια αλυσίδα συνθηκών, αφού αυτή διαχειρίζεται καταστάσεις που αφορούν αποκλειστικά την πορεία του φοιτητή στα μαθήματα. Αν η κλήση προέρχεται από την Αξιολόγηση Κώδικα, ο κόμβος παράγει την απάντηση και επιστρέφει αμέσως. Αν προέρχεται από την Εξάσκηση, παράγει σύντομη επιβεβαίωση ή υπόδειξη και επίσης επιστρέφει αμέσως. Και στις δύο περιπτώσεις η απάντηση παράγεται από το μοντέλο, με διαφορετικό prompt σε κάθε μία.

Συνολικά, η υλοποίηση του Mentor εξυπηρετεί έναν σκοπό: να φτάνει στο μοντέλο ακριβώς η πληροφορία που αντιστοιχεί στη συγκεκριμένη στιγμή. Η ταξινόμηση της πρόθεσης και η ανασύσταση όσων έχουν ήδη συμβεί δεν περιορίζουν το τι μπορεί να πει ο Mentor. Καθορίζουν σε ποιο ερώτημα καλείται να απαντήσει. Η ίδια η απάντηση, σε κάθε περίπτωση εκτός από το γνήσιο συντακτικό σφάλμα, διατυπώνεται από το μοντέλο και είναι διαφορετική για κάθε φοιτητή, ανάλογα με το ιστορικό, το επίπεδο κατανόησης και την πορεία του στη συγκεκριμένη άσκηση.

4.6 Λογική Εντοπισμού Σφαλμάτων (Debugging Agent)

Ο Debugger απαντά σε δύο βασικά ερωτήματα: αν ο κώδικας που υπέβαλε ο φοιτητής έχει τεχνικό πρόβλημα, και ποιο είναι αυτό. Η απάντηση προκύπτει σε δύο στάδια με σαφή διαχωρισμό. Πρώτα η στατική ανάλυση διαπιστώνει τι υπάρχει στον κώδικα, με βεβαιότητα και χωρίς ερμηνεία. Έπειτα το μοντέλο κρίνει αν αυτό που υπάρχει αντανακλά το ζητούμενο της άσκησης.

Το πρώτο στάδιο στηρίζεται στο άρθρωμα ast, του οποίου ο ρόλος περιγράφηκε στην ενότητα 3.5. Για να διατρέξει το δέντρο που αυτό παράγει, ο Debugger χρησιμοποιεί έναν έτοιμο μηχανισμό της βιβλιοθήκης, το μοτίβο του επισκέπτη (visitor pattern). Αντί να γραφτεί κώδικας που τρέχει μόνος του σε όλους τους κόμβους, δηλώνεται απλώς ποια είδη κόμβων σχετίζονται και τι πρέπει να καταγραφεί σε καθένα. Τη διαδρομή την αναλαμβάνει η βιβλιοθήκη.

Με ένα μόνο πέρασμα από το δέντρο συγκεντρώνονται όλα τα στοιχεία που θα χρειαστούν αργότερα. Καταγράφεται αν υπάρχει δομή if, αν υπάρχει βρόχος for, αν ορίζεται συνάρτηση, αν υπάρχει λίστα, αν γίνεται πρόσβαση με δείκτη, αν καλείται η μέθοδος προσθήκης στοιχείου. Κρατούνται επίσης δύο σύνολα ονομάτων, αυτά στα οποία δίνεται τιμή και αυτά από τα οποία διαβάζεται τιμή. Όσα διαβάζονται χωρίς να τους έχει δοθεί ποτέ τιμή είναι πιθανά λάθη, αφού πρώτα αφαιρεθούν όσα ανήκουν στην ίδια τη γλώσσα, όπως η len ή η range, ώστε να μη θεωρηθούν άγνωστες μεταβλητές.

Καταγράφονται επίσης πιο ειδικά στοιχεία, το καθένα φτιαγμένο για συγκεκριμένο, συχνό λάθος αρχαρίου. Μια αριθμητική τιμή γραμμένη μέσα σε εισαγωγικά (π.χ. `age = "25"` αντί για `age = 25`) αφήνει πίσω της το όνομα της μεταβλητής. Το ίδιο συμβαίνει όταν το όνομα `print` χρησιμοποιείται ως μεταβλητή (π.χ. `print = 5`), κάτι που καθιστά τη συνάρτηση `print()` μη διαθέσιμη από εκείνο το σημείο και μετά. Ξεχωριστά σημειώνεται η κλήση μιας μεθόδου μήκους πάνω σε λίστα (π.χ. `λίστα.length()`), μορφή που δεν υπάρχει καν στην Python (η σωστή είναι `len(λίστα)`) αλλά συγχέεται εύκολα με αυτή. Για τις λίστες κρατιούνται τρία ακόμα στοιχεία, το πλήθος των στοιχείων της καθεμίας, αν όλα είναι κείμενο, και αν όλα είναι κενό κείμενο. Για τις συναρτήσεις, τέλος, μετριέται πόσες παραμέτρους δηλώνει κάθε ορισμός, ώστε αργότερα να συγκριθεί με τον αριθμό ορισμάτων σε κάθε κλήση.

Όλα αυτά από μόνα τους δεν είναι ακόμα ευρήματα. Γίνονται ευρήματα στο επόμενο βήμα, όπου συγκρίνονται με το τι ζητά η άσκηση. Για τη σύγκριση αυτή, η εκφώνηση και τα κριτήρια ενώνονται σε ένα κείμενο και μετατρέπονται σε πεζά, ώστε μια απαίτηση να αναγνωρίζεται είτε γράφεται στα κριτήρια είτε μόνο μέσα στην εκφώνηση. Αν το κείμενο αναφέρει δομή `if` και δεν βρέθηκε καμία, καταγράφεται εύρημα. Το ίδιο ισχύει για τον βρόχο `for`, τον ορισμό συνάρτησης, τη λίστα, και την προσθήκη στοιχείου.

Άλλοι έλεγχοι εντοπίζουν λάθος ενώ η ζητούμενη δομή υπάρχει, απλά χρησιμοποιείται λανθασμένα. Αν η εκφώνηση ζητά το πρώτο στοιχείο μιας λίστας και ο κώδικας δείχνει με δείκτη σε κάτι άλλο εκτός από το μηδέν, ο δείκτης σημειώνεται ως λάθος. Υπάρχει βρόχος φτιαγμένος για άθροισμα αλλά καμία τιμή δεν αυξάνεται μέσα του; Τότε λείπει ο αθροιστής. Δύο ακόμα έλεγχοι αφορούν όχι τη δομή μιας λίστας αλλά το περιεχόμενό της. Αν η εκφώνηση ζητά αριθμούς και η λίστα περιέχει μόνο κείμενο, το πρόβλημα είναι ο τύπος των στοιχείων. Αν όμως το πλήθος είναι το σωστό αλλά όλα τα στοιχεία είναι κενά, το πρόβλημα δεν είναι πια το πλήθος, είναι η απουσία πραγματικού περιεχομένου, μια διάκριση που κρατιέται ξεχωριστή ώστε η υπόδειξη να στοχεύει στο σωστό σημείο.

Ένας από τους ελέγχους κοιτάζει τη ροή της εκτέλεσης και όχι απλώς την ύπαρξη δομών. Για κάθε βρόχο `while`, μαζεύονται τα ονόματα που εμφανίζονται στη συνθήκη του και τα ονόματα που αλλάζουν τιμή μέσα στο σώμα του. Αν κανένα όνομα δεν είναι κοινό στα δύο σύνολα και δεν υπάρχει εντολή διακοπής, ο βρόχος δεν πρόκειται να σταματήσει ποτέ. Ο έλεγχος αυτός πιάνει ένα πρόβλημα που δεν φαίνεται ούτε ως συντακτικό λάθος ούτε ως δομή που λείπει, και το οποίο θα εμφανιζόταν μόνο αν έτρεχε ο κώδικας.

Δύο έλεγχοι ξεχωρίζουν επειδή διακρίνουν διαφορετικές αιτίες που δίνουν το ίδιο ορατό αποτέλεσμα. Όταν η εκφώνηση ζητά εμφάνιση αποτελέσματος και δεν βρίσκεται καμία εκτύπωση, το σύστημα δεν καταγράφει απλώς ότι λείπει η εντολή. Κοιτάζει πρώτα αν η συνάρτηση εκτύπωσης έχει χρησιμοποιηθεί ως μεταβλητή, και έπειτα αν ορίζεται συνάρτηση που δεν καλείται ποτέ, οπότε το πρόγραμμα δεν βγάζει τίποτα επειδή το σώμα της δεν εκτελείται. Μόνο αν δεν ισχύει κανένα από τα δύο καταγράφεται η απλή απουσία εκτύπωσης. Ξεχωριστά, αν η εκτύπωση παίρνει ως όρισμα το όνομα μιας συνάρτησης χωρίς παρενθέσεις, αναγνωρίζεται ότι εμφανίζεται η ίδια η συνάρτηση αντί για το αποτέλεσμά της. Τρεις διαφορετικές αιτίες για το ίδιο κενό αποτέλεσμα οδηγούν έτσι σε τρεις διαφορετικές κατηγορίες, και επομένως σε τρεις διαφορετικές εξηγήσεις προς τον φοιτητή.

Οι έλεγχοι αυτοί δεν αποκλείουν ο ένας τον άλλο. Σε έναν κώδικα μπορεί να ενεργοποιηθούν ταυτόχρονα περισσότεροι από έναν, για παράδειγμα να λείπει μια δομή που ζητά η εκφώνηση και παράλληλα να χρησιμοποιείται μεταβλητή που δεν έχει οριστεί. Κανένας τους όμως δεν αποτελεί από μόνος του συμπέρασμα. Είναι υποψήφια ευρήματα, που δηλώνουν ότι κάτι αξίζει να εξεταστεί. Ποιο από αυτά είναι το πραγματικό πρόβλημα, ή αν τελικά κανένα δεν είναι, το κρίνει το μοντέλο. Η

δυνατότητα να απαντήσει ότι δεν υπάρχει πρόβλημα, παρότι έχουν ενεργοποιηθεί έλεγχοι, είναι μέρος αυτής της κρίσης και όχι εξαίρεση σε αυτήν.

Στο τέλος αυτής της διαδικασίας υπάρχουν δύο σύνολα: οι διατυπωμένες παρατηρήσεις, καθεμία με την κατηγορία της, και ο πίνακας με τα ακατέργαστα στοιχεία, οι απαντήσεις ναι ή όχι και τα ονόματα που βρέθηκαν. Και τα δύο δίνονται στο μοντέλο, το πρώτο ως διατυπωμένες ενδείξεις και το δεύτερο ως ακατέργαστα δεδομένα, ώστε να μπορεί να κρίνει και πέρα από όσα έχουν ήδη διατυπωθεί.

Τα ευρήματα αυτά δεν εξαντλούν όσα εξετάζονται. Το prompt ζητά επιπλέον από το μοντέλο να ελέγξει σημεία που η στατική ανάλυση δεν μπορεί να καλύψει με βεβαιότητα, όλα τους ζητήματα νοήματος και όχι δομής. Ταιριάζουν οι τιμές που τυπώνονται με αυτές που ζητά η εκφώνηση, κεφαλαία και ορθογραφία συμπεριλαμβανομένων; Είναι αντεστραμμένη η λογική ανάμεσα σε if, elif και else, με το ένα αποτέλεσμα να εμφανίζεται εκεί που ζητείται το άλλο; Ελέγχει η συνθήκη τη σωστή τιμή, με τον σωστό τελεστή σύγκρισης; Ορίζεται και καλείται η συνάρτηση με τρόπο που αντιστοιχεί πράγματι σε αυτό που περιγράφει η άσκηση; Καμία από αυτές τις ερωτήσεις δεν απαντιέται από την ύπαρξη ή την απουσία μιας δομής. Απαιτούν σύγκριση ανάμεσα στο νόημα της εκφώνησης και στο νόημα του κώδικα, και εκεί ακριβώς βρίσκεται η συνεισφορά του μοντέλου.

Ο κατάλογος των κατηγοριών έχει είκοσι δύο εγγραφές, καθεμία με σύντομη περιγραφή. Οι ίδιες ονομασίες χρησιμοποιούνται και αλλού στο σύστημα, από τον Mentor για να διαλέξει την κατάλληλη παιδαγωγική εξήγηση, και από το επίπεδο ενορχήστρωσης για να καταγράψει τα συχνότερα λάθη κάθε φοιτητή. Επειδή τα τρία αυτά σημεία πρέπει να λένε το ίδιο πράγμα, ο κώδικας φέρει σχόλιο που το υπενθυμίζει.

Μετά την απάντηση του μοντέλου γίνονται δύο έλεγχοι στην κατηγορία που επέλεξε. Ο πρώτος βεβαιώνει ότι υπάρχει στον κατάλογο, αλλιώς την αλλάζει στη γενική κατηγορία λογικού προβλήματος. Ο δεύτερος είναι πιο στοχευμένος. Αν η κατηγορία αφορά αριθμητική τιμή γραμμένη σε εισαγωγικά, αλλά η ανάλυση δεν βρήκε καμία τέτοια μεταβλητή, η κατηγορία αλλάζει επίσης σε γενική. Είναι η μόνη κατηγορία του καταλόγου με απολύτως μετρήσιμο αντίκρισμα στον κώδικα, οπότε είναι και η μόνη που μπορεί να διασταυρωθεί με αυτόν τον τρόπο. Οι υπόλοιπες αφορούν κρίσεις που η στατική ανάλυση δεν μπορεί να επιβεβαιώσει, και γίνονται δεκτές όπως δόθηκαν.

Ο κόμβος του Debugger ακολουθεί τη σειρά που περιγράφηκε στο Κεφάλαιο 3. Πρώτα επιχειρείται η ανάλυση του κώδικα. Αν αποτύχει, ελέγχονται με κανονικές εκφράσεις τα δύο γνωστά μοτίβα, η σύνταξη ελέγχου από άλλη γλώσσα και η αριθμητική τιμή στη θέση παραμέτρου, και επιστρέφεται η αντίστοιχη ένδειξη, αλλιώς το μήνυμα του parser μαζί με τον αριθμό γραμμής. Αν η ανάλυση πετύχει, μαζεύονται τα ευρήματα, καλείται το μοντέλο, και το αποτέλεσμα παίρνει μία από δύο μορφές, είτε ότι δεν εντοπίστηκε πρόβλημα είτε ότι εντοπίστηκε, μαζί με την κατηγορία και την εξήγηση.

Η αναφορά αυτή ξεκινά πάντα με μια σταθερή ένδειξη που δηλώνει το είδος της. Οι ενδείξεις δεν προορίζονται για ανάγνωση από άνθρωπο, επιτρέπουν στα επόμενα στάδια να καταλάβουν αμέσως, χωρίς να ερμηνεύσουν κείμενο, αν πρόκειται για συντακτικό λάθος, τεχνικό πρόβλημα, ή καθαρό αποτέλεσμα. Ο Assessor τις χρησιμοποιεί για να αναγνωρίσει σοβαρή παράβαση, ο Mentor για να διαλέξει ανάμεσα σε έτοιμη και σε παραγόμενη υπόδειξη, το επίπεδο ενορχήστρωσης για να απομονώσει την κατηγορία και να την προσθέσει στο ιστορικό λαθών του φοιτητή.

Ο καταμερισμός στον Debugger είναι επομένως σαφής και στα δύο στάδια. Η στατική ανάλυση απαντά με βεβαιότητα σε ερωτήματα που έχουν αντικειμενική απάντηση, τι υπάρχει και τι λείπει στον κώδικα. Το μοντέλο απαντά στο ερώτημα που δεν έχει τέτοια απάντηση, αν ο συγκεκριμένος κώδικας

κάνει αυτό που περιγράφει η συγκεκριμένη εκφώνηση. Καμία από τις δύο πλευρές δεν θα επαρκούσε μόνη της.

4.7 Λογική Κριτηρίων (Assessment Agent)

Ο Assessor παράγει τέσσερα στοιχεία για κάθε υποβολή: αν ο κώδικας είναι σωστός, τι πρέπει να συμβεί στη συνέχεια, πόσο καλά φαίνεται να έχει κατανοήσει ο φοιτητής, και ένα σύντομο κείμενο ανατροφοδότησης που θα αξιοποιήσει ο Mentor. Κανένα από τα τέσσερα δεν προκύπτει από μεμονωμένο έλεγχο, οι ντετερμινιστικοί μηχανισμοί που ακολουθούν διαπιστώνουν τι ισχύει για τον συγκεκριμένο κώδικα, ενώ η συνολική κρίση για το τι σημαίνουν όλα αυτά μαζί, για τον συγκεκριμένο φοιτητή σε αυτό το σημείο της πορείας του, ανήκει στο μοντέλο. Τα επίπεδα ελέγχου που ακολουθούν αφορούν τα δεδομένα εισόδου προς το μοντέλο, τη βάση πάνω στην οποία στηρίζεται αυτή η μία κρίση του.

Το πρώτο επίπεδο ελέγχου αφορά τι ζητούν τα κριτήρια. Κάθε κριτήριο επιτυχίας αντιστοιχίζεται σε ένα δομικό εύρημα. Πριν από αυτό, τα κριτήρια κανονικοποιούνται, ώστε να αντιμετωπίζονται με τον ίδιο τρόπο είτε δίνονται ως λίστα είτε ως ενιαίο κείμενο, και αν λείπουν εντελώς χρησιμοποιείται ένα γενικό κριτήριο λειτουργικότητας. Η αντιστοίχιση γίνεται με αναζήτηση λέξεων-κλειδιών μέσα στο κείμενο κάθε κριτηρίου: αν αναφέρει βρόχο ή επανάληψη, ελέγχεται η ύπαρξη βρόχου, αν αναφέρει συνάρτηση ή ορισμό, ελέγχεται η ύπαρξη ορισμού. Επειδή ένα τέτοιο κριτήριο συχνά απαιτεί ταυτόχρονα και επιστροφή τιμής ή κλήση, ο έλεγχος είναι συζευκτικός, αν αναφέρει return ελέγχεται και αυτό, όπως επίσης και αν αναφέρει κλήση ελέγχεται και αυτή (Παράρτημα Α, Σχήμα Α.4). Ένα κριτήριο που αναφέρεται στη συντακτική ορθότητα θεωρείται πάντα ικανοποιημένο, αφού αυτή έχει ήδη ελεγχθεί νωρίτερα στη ροή. Ένα κριτήριο που αναφέρεται σε `elif` ή σε λογικό τελεστή θεωρείται ικανοποιημένο είτε ο φοιτητής χρησιμοποίησε πράγματι τη λέξη-κλειδί `elif` είτε πέτυχε το ίδιο αποτέλεσμα με εμφωλευμένο `if` και λογική πράξη (`and/or`), αναγνωρίζοντας ότι πρόκειται για δομικά ισοδύναμες λύσεις του ίδιου προβλήματος.

Δύο έλεγχοι αυτού του επιπέδου είναι εξαιρετικά προσεκτικοί. Ο πρώτος αφορά την εκτύπωση: δεν αρκεί να υπάρχει κλήση εκτύπωσης κάπου στο αρχείο, μια ειδική διάσχιση του δέντρου κρατά μετρητή βάθους που αυξάνεται όταν η διάσχιση μπαίνει μέσα σε συνάρτηση που δεν καλείται ποτέ, και μετρούν μόνο οι κλήσεις σε βάθος μηδέν, δηλαδή σε σημείο που πράγματι εκτελείται. Ο δεύτερος αφορά το πλήθος στοιχείων μιας λίστας, όταν το κριτήριο το προσδιορίζει με λέξεις όπως ένα, δύο ή τρία.

Το δεύτερο επίπεδο αφορά τι ζητά η εκφώνηση. Εδώ οι απαιτήσεις αντλούνται απευθείας από την ίδια την εκφώνηση, με κανονικές εκφράσεις. Αναζητούνται ονόματα που ακολουθούν λέξεις όπως όνομα, μεταβλητή, συνάρτηση ή λίστα, ονόματα που παρατίθενται μετά από άνω κάτω τελεία, αριθμητικές τιμές που ακολουθούν τη λέξη τιμή, και ονόματα με ρητή δήλωση τύπου, ως `string` ή ως αριθμό. Αν δεν βρεθεί ρητή δήλωση τύπου, το σύστημα κοιτάζει το γενικότερο κείμενο, αν η εκφώνηση μιλά για κείμενο ή για αριθμούς.

Για να συγκριθούν οι απαιτήσεις αυτές με τον κώδικα, το σύστημα δεν αρκείται σε αναζήτηση κειμένου. Διατρέχει το δέντρο και υπολογίζει την τιμή που δίνεται σε κάθε όνομα, καλύπτοντας σταθερές τιμές, πρόσημα, και αριθμητικές πράξεις μεταξύ σταθερών. Αν λοιπόν η εκφώνηση ζητά μια τιμή που προκύπτει από παράσταση, και ο φοιτητής γράψει την ίδια παράσταση αντί για το αποτέλεσμά της, η σύγκριση πετυχαίνει, το σύστημα έχει ήδη υπολογίσει ότι οι δύο τιμές ταυτίζονται. Μια απαιτούμενη αριθμητική τιμή δεν χρειάζεται μάλιστα να βρίσκεται σε ανάθεση, μπορεί να

δίνεται ως όρισμα σε κλήση, σε σύγκριση ή σε επιστροφή, μετράει κάθε αριθμητική σταθερά του προγράμματος, όπου κι αν εμφανίζεται.

Στο ίδιο επίπεδο ανήκει και ο έλεγχος αντεστραμμένης λογικής, ο οποίος αξίζει να περιγραφεί βήμα-βήμα, αφού συνδυάζει ανάλυση κειμένου και ανάλυση δέντρου. Παίρνει το παράδειγμα που ήδη χρησιμοποιήθηκε στο 3.6, «αν μεγαλύτερη από 50, τύπωσε High, αλλιώς Low», και το κάνει συγκεκριμένο. Ενεργοποιείται μόνο αν η εκφώνηση περιέχει τη λέξη «αλλιώς» και μια κατεύθυνση σύγκρισης, εδώ «μεγαλύτερη». Χωρίζει το κείμενο στα δύο στο σημείο του «αλλιώς», και από κάθε μέρος απομονώνει το κείμενο μέσα σε εισαγωγικά, δηλαδή τι πρέπει να τυπωθεί σε κάθε περίπτωση: «High» από το πρώτο κομμάτι, «Low» από το δεύτερο. Έπειτα εντοπίζει στον κώδικα τη δομή if και βρίσκει τι τυπώνεται σε κάθε κλάδο της. Επειδή η εκφώνηση λέει «μεγαλύτερη», το High αναμένεται στον κλάδο if. Αν στον κώδικα είναι αντίστροφα, δηλαδή το Low στο if και το High στο else, ενώ η κατεύθυνση της σύγκρισης είναι η ίδια, η λογική σημειώνεται ως αντεστραμμένη.

Το τρίτο επίπεδο αφορά τη συνέπεια τύπων, και στηρίζεται σε δύο καταλόγους ονομάτων. Ο πρώτος περιέχει τριάντα ένα ονόματα που συνήθως αντιστοιχούν σε αριθμητικές τιμές, όπως age, score, total ή price. Ο δεύτερος περιέχει δεκαέξι ονόματα που συνήθως αντιστοιχούν σε κείμενο, όπως username, email ή city. Η βασική μορφή του ελέγχου συγκρίνει το όνομα της μεταβλητής με τους δύο καταλόγους. Αν το όνομα ανήκει στον πρώτο αλλά η τιμή του είναι γραμμένη μέσα σε εισαγωγικά, ή αν ανήκει στον δεύτερο αλλά η τιμή δεν μοιάζει με κείμενο, καταγράφεται ασυμφωνία τύπου. Ο έλεγχος γίνεται αυστηρότερος όταν το ίδιο το κείμενο των κριτηρίων αναφέρει λέξεις σχετικές με αριθμούς, όπως αριθμητική τιμή, int, float, δεκαδικός ή χωρίς εισαγωγικά. Τότε οποιαδήποτε μεταβλητή έχει πάρει αριθμό μέσα σε εισαγωγικά θεωρείται παράβαση, ανεξάρτητα από το όνομά της. Ένας τρίτος έλεγχος, ενεργός μόνο στην ενότητα των Λιστών, εντοπίζει λίστες των οποίων τα στοιχεία είναι κενά κείμενα. Τα ευρήματα αυτού του επιπέδου, μαζί με τεχνική αναφορά σφάλματος από τον Debugger ή αποτυχία στην αυστηρή αντιστοίχιση, ενεργοποιούν σοβαρή παράβαση και μηδενίζουν το εσωτερικό ποσοστό, το οποίο υπολογίζεται ως το πλήθος των κριτηρίων που αντιστοιχίστηκαν θετικά προς το συνολικό τους πλήθος.

Ο Assessor εκτελείται σε κάθε υποβολή, ακόμα και όταν ο κώδικας δεν είναι συντακτικά έγκυρος και ο Debugger έχει ήδη εντοπίσει σφάλμα. Σε αυτή την περίπτωση το δέντρο δεν μπορεί να κατασκευαστεί, οπότε κάθε δομικό εύρημα καταγράφεται ως απόν, καμία ανάθεση και καμία εκτύπωση δεν εντοπίζεται. Αυτό δεν αφήνει την κρίση αναποφάσιστη. Η ίδια η αναφορά σφάλματος του Debugger λειτουργεί ως σοβαρή παράβαση και καθορίζει μόνη της την έκβαση, ο κώδικας κρίνεται λάθος ανεξάρτητα από το τι θα έδειχναν τα υπόλοιπα κριτήρια. Η ανάλυση δεν χρειάζεται δηλαδή έγκυρο κώδικα για να ξεκινήσει, απλώς δεν βρίσκει τίποτα θετικό να καταγράψει όταν ο κώδικας δεν είναι έγκυρος.

Όλα τα παραπάνω συγκροτούν την εικόνα που δίνεται στο μοντέλο, μαζί με το ιστορικό του φοιτητή στη συγκεκριμένη άσκηση, πόσες προσπάθειες έχει κάνει, πόσες υποδείξεις έχει λάβει, πόση ώρα έχει αφιερώσει. Το ερώτημα που καλείται να απαντήσει είναι τι απόφαση πρέπει να ληφθεί συνδυάζοντας όλα αυτά μαζί. Ένας κώδικας μπορεί να ικανοποιεί όλα τα κριτήρια χωρίς να λύνει πραγματικά την άσκηση, ή να αποτυγχάνει σε ένα κριτήριο για λόγο που δεν επηρεάζει το ζητούμενο. Δύο φοιτητές με το ίδιο ακριβώς ποσοστό μπορεί να βρίσκονται σε εντελώς διαφορετική κατάσταση, αν ο ένας έφτασε εκεί με μία προσπάθεια και ο άλλος με πέντε. Καμία από αυτές τις διακρίσεις δεν μπορεί να προκύψει από έναν μεμονωμένο έλεγχο, χρειάζεται να ζυγιστούν όλες μαζί για να παραχθούν τα τέσσερα στοιχεία της απόφασης. Το κείμενο ανατροφοδότησης ακολουθεί την ίδια λογική. Σε αντίθεση με τον Debugger, που διαλέγει ανάμεσα σε είκοσι δύο προκαθορισμένες κατηγορίες, εδώ δεν υπάρχει

έτοιμος κατάλογος από τον οποίο να διαλέξει. Το κείμενο γράφεται εξαρχής, ειδικά για τον συνδυασμό ευρημάτων και ιστορικού της συγκεκριμένης υποβολής, και είναι αυτό που θα καθορίσει τελικά τι θα πει ο Mentor στον φοιτητή.

Το τέταρτο και τελευταίο στάδιο αφορά το πώς αποφασίζεται advance, repeat ή support. Η απάντηση που επιστρέφεται περνά από επικύρωση, η οποία δεν αντικαθιστά την κρίση του μοντέλου, ελέγχει αν αυτή είναι συνεπής με τα ίδια αντικειμενικά δεδομένα πάνω στα οποία στηρίχθηκε. Στην ίδια λογική εντάσσεται και ο έλεγχος των ισχυρισμών απουσίας. Το κείμενο ανατροφοδότησης σαρώνεται για διατυπώσεις που δηλώνουν ότι λείπει μια δομή, και αν το δέντρο δείχνει την ίδια δομή παρούσα, η απάντηση απορρίπτεται. Ο έλεγχος περιορίζεται σε διατυπώσεις για την ύπαρξη της δομής και όχι για το τι κάνει, ώστε να μην ακυρώνονται σημασιολογικές κρίσεις που η στατική ανάλυση δεν μπορεί να επιβεβαιώσει.

Τα κατώφλια που χρησιμοποιούνται είναι συγκεκριμένα. Η μετάβαση από επανάληψη σε υποστήριξη ενεργοποιείται όταν οι προσπάθειες έχουν φτάσει τις τρεις, ή οι υποδείξεις τις δύο, ή ο χρόνος τα εκατόν ογδόντα δευτερόλεπτα. Η μετάβαση από προαγωγή σε επανάληψη ενεργοποιείται όταν χρειάστηκαν τέσσερις προσπάθειες ή τρεις υποδείξεις. Ένα συγκεκριμένο παράδειγμα δείχνει τη λογική καλύτερα από τον κατάλογο κατωφλίων. Το δείγμα D3 της αξιολόγησης του Κεφαλαίου 5 έχει σωστό κώδικα, αλλά ο φοιτητής έφτασε εκεί με τέσσερις προσπάθειες και δύο υποδείξεις. Το μοντέλο, βλέποντας σωστό κώδικα, τείνει να προτείνει advance. Το κατώφλι όμως έχει ήδη ξεπεραστεί, οπότε η επικύρωση υποβαθμίζει την απόφαση σε repeat, χωρίς να αλλάξει την ετυμηγορία για τον κώδικα. Ο φοιτητής μαθαίνει ότι ο κώδικάς του είναι σωστός, αλλά προτείνεται μία ακόμα άσκηση πριν προχωρήσει, γιατί η ευκολία με την οποία έφτασε εκεί δεν επιβεβαιώνει ακόμα ανεξάρτητη επάρκεια.

Το επίπεδο κατανόησης αντιμετωπίζεται ξεχωριστά. Αν η τιμή που επέστρεψε το μοντέλο ανήκει στις τέσσερις έγκυρες κατηγορίες, γίνεται δεκτή. Αν όχι, αντικαθίσταται μόνο αυτό το πεδίο από ντετερμινιστικό υπολογισμό, ο οποίος αποδίδει την υψηλότερη κατηγορία όταν ο κώδικας είναι σωστός με ποσοστό τουλάχιστον ενενήντα, το πολύ δύο προσπάθειες και καμία υπόδειξη, την επόμενη όταν είναι σωστός με ποσοστό τουλάχιστον ογδόντα, τη χαμηλότερη όταν οι προσπάθειες έχουν φτάσει τις τέσσερις ή οι υποδείξεις τις τρεις, και την ενδιάμεση σε κάθε άλλη περίπτωση.

Το εναλλακτικό μονοπάτι χρησιμοποιεί τα ίδια ευρήματα και τα ίδια κατώφλια. Αυτό που προσθέτει είναι μια σταθερή σειρά προτεραιότητας για τη σύνταξη του κειμένου ανατροφοδότησης, ξεκινώντας από την τεχνική αναφορά του Debugger, συνεχίζοντας με τις ασυμφωνίες τύπων, έπειτα τις αποτυχίες της αυστηρής αντιστοίχισης, και καταλήγοντας στα κριτήρια που δεν ικανοποιήθηκαν. Το ίδιο εξωτερικό δίκτυ ασφαλείας που περιγράφηκε στην ενότητα 3.6 περιβάλλει ολόκληρη τη διαδικασία.

Ο Assessor διαφέρει από τους άλλους δύο πράκτορες μόνο ως προς την έκταση των ελέγχων του. Είναι περισσότεροι και πιο εξειδικευμένοι, επειδή η αξιολόγηση απαιτεί σύγκριση με συγκεκριμένα ονόματα, τιμές και κείμενα. Το ερώτημα όμως που μένει στο τέλος παραμένει ένα, και δεν έχει αντικειμενική απάντηση: τι σημαίνει το σύνολο αυτών των στοιχείων για την πορεία ενός συγκεκριμένου φοιτητή. Οι έλεγχοι εξασφαλίζουν ότι η κρίση αυτή στηρίζεται σε ακριβή δεδομένα, η επικύρωση που ακολουθεί ότι παραμένει συνεπής με αυτά.

4.8 Επίπεδο Ενορχήστρωσης

Το γράφημα των τριών πρακτόρων αποφασίζει τι θα ειπωθεί στον φοιτητή. Δεν αποφασίζει όμως ποια άσκηση θα του ανατεθεί, σε ποια ενότητα βρίσκεται, πόσες προσπάθειες έχει κάνει, ούτε τι θα παραμείνει αποθηκευμένο όταν κλείσει τη σελίδα. Στην πραγματικότητα, το γράφημα δεν θυμάται τίποτα από τη μία κλήση στην άλλη, ξέρει μόνο ότι του δοθεί την κάθε φορά. Κάποιος πρέπει να ετοιμάσει αυτά τα δεδομένα πριν το καλέσει, και κάποιος πρέπει να κρατήσει το αποτέλεσμα μετά, ώστε να υπάρχει έτοιμο την επόμενη φορά. Αυτό το ρόλο έχει το επίπεδο ενορχήστρωσης, ο κώδικας που περιβάλλει το γράφημα και τρέχει πριν και μετά από κάθε κλήση του. Η παιδαγωγική σημασία όσων περιγράφονται εδώ έχει ήδη εξεταστεί στο Κεφάλαιο 3, η παρούσα ενότητα δείχνει πού ζει αυτή η λογική και πώς διατηρείται η κατάστασή της.

Το γράφημα καλείται από τρία διαφορετικά σημεία, ένα για κάθε λειτουργία. Η διαφορά τους δεν βρίσκεται στο γράφημα, που είναι το ίδιο και στις τρεις, αλλά στο πόση κατάσταση συγκροτεί το καθένα πριν το καλέσει. Η κύρια ροή διδασκαλίας συγκροτεί είκοσι επτά πεδία, η Αξιολόγηση Κώδικα έξι, η Εξάσκηση πέντε. Το επίπεδο ενορχήστρωσης είναι αυτό που εξειδικεύει το γράφημα σε κάθε περίπτωση, χωρίς το ίδιο το γράφημα να χρειάζεται τρεις διαφορετικές εκδοχές.

4.8.1 Ο κύκλος της κύριας ροής

Κάθε μήνυμα του φοιτητή περνά από την ίδια ακολουθία βημάτων, πριν φτάσει καν στο γράφημα. Πρώτα ανακτάται ο φοιτητής και όλο το ιστορικό του από τη βάση. Αν είναι η πρώτη φορά που συνδέεται, ταξινομείται το επίπεδο εμπειρίας του. Ανασυντίθενται οι μετρητές της τρέχουσας άσκησης. Επιλέγεται δυσκολία και άσκηση. Όλα αυτά μαζεύονται σε ένα πακέτο, και μόνο τότε καλείται το γράφημα. Στο τέλος, αποθηκεύεται ότι προέκυψε και επιστρέφεται η απάντηση.

Δύο από αυτά τα βήματα καλούν μοντέλο, αλλά έξω από το γράφημα, πριν καν αρχίσει η δουλειά των τριών πρακτόρων. Το πρώτο είναι η ταξινόμηση του αρχικού επιπέδου εμπειρίας, μόνο στην πρώτη σύνδεση: η απάντηση του φοιτητή στην εισαγωγική ερώτηση κατατάσσεται σε αρχάριο, προχωρημένο, ή ασαφές. Στην ασαφή περίπτωση το σύστημα δεν επιμένει, τον βάζει σιωπηρά ως αρχάριο και σημειώνει ότι η κατάταξη έγινε με προεπιλογή, ώστε ο Mentor να μπορεί να το αναφέρει διακριτικά. Το δεύτερο είναι η αναγνώριση της πρόθεσης όταν εκκρεμεί προαγωγή, δηλαδή αν ο φοιτητής θέλει να προχωρήσει ή να μείνει στην ίδια ενότητα. Και οι δύο αφορούν αποφάσεις που πρέπει να έχουν ληφθεί πριν καν συγκροτηθεί η κατάσταση που θα δει το γράφημα.

Οι μετρητές της τρέχουσας άσκησης, πόσες υποδείξεις και πόσες προσπάθειες έχουν γίνει, δεν διαβάζονται από αποθηκευμένα πεδία. Υπολογίζονται εκ νέου κάθε φορά, με την ίδια αρχή ορίου που περιγράφηκε στην ενότητα 4.5 για τον Mentor: το ιστορικό διαβάζεται προς τα πίσω, μετρώντας τα αντίστοιχα σύμβολα, μέχρι να συναντηθεί σύμβολο έναρξης άσκησης ή προαγωγής. Ο λόγος είναι πρακτικός. Κάθε μήνυμα κρατά μόνιμα, στη δική του γραμμή, πόσες προσπάθειες έγιναν σε αυτό, ένα ακατέργαστο γεγονός που ποτέ δεν αλλάζει. Το άθροισμα για την τρέχουσα άσκηση όμως δεν αποθηκεύεται πουθενά, υπολογίζεται ξανά σε κάθε αίτημα. Έτσι δεν χρειάζεται κανένας χειροκίνητος μηδενισμός σε κάθε αλλαγή άσκησης, το όριο της μέτρησης προκύπτει αυτόματα από το πού σταματά η σάρωση.

Από τους δύο αυτούς μετρητές παράγεται κι ένας τρίτος, που δεν αντιστοιχεί σε τίποτα αποθηκευμένο: ένας δείκτης δυσκολίας που δίνεται στον Mentor, ίσος με το πλήθος των υποδείξεων, αυξημένο κατά ένα, με ανώτατο όριο το τρία. Ο δείκτης αυτός καθορίζει αν η επόμενη απάντηση θα ξεκινήσει αναγνωρίζοντας την προσπάθεια του φοιτητή, πριν προχωρήσει στο περιεχόμενό της.

4.8.2 Μεταβάσεις και προσαρμογή δυσκολίας

Η προαγωγή σε επόμενη ενότητα δεν γίνεται τη στιγμή της σωστής απάντησης. Όταν μια λύση κριθεί σωστή, το σύστημα δεν αυξάνει αμέσως το αναγνωριστικό της ενότητας. Θέτει μια ένδειξη εκκρεμούς προαγωγής και καθαρίζει την ενεργή άσκηση. Η πραγματική μετάβαση συμβαίνει στο επόμενο μήνυμα, αφού πρώτα διαπιστωθεί τι θέλει ο φοιτητής.

Η ταξινόμηση της πρόθεσης εδώ οδηγεί σε δύο εκβάσεις. Αν ο φοιτητής θέλει να προχωρήσει, η ενότητα αυξάνεται, η ένδειξη σβήνει, και μηδενίζονται μαζί οι μετρητές υποδείξεων και προσπαθειών, η ένδειξη έναρξης άσκησης, και η αναμονή αποριών, αφού ξεκινά κάτι καινούριο. Αν αντίθετα ζητά επιπλέον εξάσκηση στην ίδια ενότητα, είτε το δηλώνει ρητά είτε απλώς ζητά άσκηση, η ένδειξη σβήνει χωρίς αύξηση, και το κείμενο της προηγούμενης άσκησης κρατιέται προσωρινά ώστε η επόμενη να μην είναι η ίδια. Σε κάθε άλλη απάντηση δεν αλλάζει τίποτα, η ένδειξη παραμένει εκκρεμής και ο φοιτητής ξαναρωτιέται. Συμπληρωματικά υπάρχει και ένας απλός ντετερμινιστικός έλεγχος καταφατικής απάντησης, που καλύπτει συνεδρίες οι οποίες είχαν ξεκινήσει πριν προστεθεί η ένδειξη στη βάση.

Η επιλογή της δυσκολίας γίνεται με μια σειρά προτεραιότητας: η πρώτη συνθήκη που ισχύει καθορίζει το αποτέλεσμα, ανεξάρτητα από τις υπόλοιπες. Αν εκκρεμεί δοκιμαστική αλλαγή επιπέδου, αυτή υπερισχύει όλων. Αλλιώς, αν οι προσπάθειες στην τρέχουσα άσκηση έχουν φτάσει τις τρεις, η επόμενη είναι εύκολη, ότι επίπεδο κι αν έχει επίσημα ο φοιτητής. Αλλιώς, αν είναι προχωρημένος ή λύνει πολύ γρήγορα, δηλαδή τουλάχιστον τρεις συνεχόμενες λυμένες με μέσο χρόνο κάτω από τριάντα δευτερόλεπτα, είναι δύσκολη. Σε κάθε άλλη περίπτωση είναι εύκολη. Ο κανόνας για τη γρήγορη λύση επιτρέπει σε έναν επίσημα αρχάριο που τα πηγαίνει πολύ άνετα να λάβει δυσκολότερη άσκηση, χωρίς να έχει αλλάξει το επίσημο επίπεδό του.

Η δοκιμαστική αλλαγή δυσκολίας υλοποιείται με ένα και μόνο πεδίο κειμένου, το οποίο παίρνει μία από τρεις τιμές: κενό, δοκιμή προς τα πάνω, ή δοκιμή προς τα κάτω. Για παράδειγμα, έστω ότι ένας αρχάριος λύνει σωστά δύο ασκήσεις στη σειρά, με ισχυρή κατανόηση και στις δύο. Το πεδίο γίνεται δοκιμή προς τα πάνω, και η επόμενη άσκηση που θα του ανατεθεί είναι δύσκολη, σαν τεστ. Αν τη λύσει σωστά, το επίπεδό του γίνεται μόνιμα προχωρημένο και το πεδίο αδειάζει. Αν αποτύχει, το πεδίο απλά αδειάζει χωρίς καμία μόνιμη αλλαγή, και συνεχίζει κανονικά ως αρχάριος. Αντίστοιχα, ένας ήδη προχωρημένος που δείχνει επανειλημμένα ανάγκη υποστήριξης σημειώνεται για δοκιμή προς τα κάτω, παίρνει προσωρινά ευκολότερες ασκήσεις, και αν δείξει ξανά ισχυρή ή καλή κατανόηση, το πεδίο αδειάζει και επιστρέφει στο κανονικό επίπεδο, χωρίς να έχει χάσει ποτέ επίσημα τον τίτλο του προχωρημένου.

4.8.3 Εκτέλεση και αποθήκευση

Πριν καλέσει το γράφημα, το σύστημα ετοιμάζει δύο πράγματα. Πρώτα, μια συνολική εικόνα της πορείας του φοιτητή σε όλο το πρόγραμμα, πόσες προσπάθειες συνολικά, πόσο χρόνο κατά μέσο όρο, ποια λάθη επαναλαμβάνει, ξεχωριστή από τα στοιχεία της τρέχουσας άσκησης. Έπειτα συγκεντρώνει όλα τα υπόλοιπα δεδομένα που χρειάζονται οι πράκτορες σε ένα πακέτο, με δύο μικρές εξαιρέσεις. Ο κώδικας και ο χρόνος της προηγούμενης άσκησης καθαρίζονται σκόπιμα, ώστε να μην περάσουν κατά λάθος σαν απάντηση στην καινούρια.

Η κλήση του γραφήματος έχει ένα ανώτατο χρονικό όριο εξήντα δευτερολέπτων. Αν κάτι πάει στραβά μέσα σε αυτό το διάστημα, ο φοιτητής δεν μένει χωρίς απάντηση, παίρνει ένα από δύο

προκαθορισμένα μηνύματα ανάλογα με το πώς προέκυψε το πρόβλημα, ενθαρρυντικό αν δεν είχε ζητήσει τίποτα ο ίδιος, ή μια απλή δικαιολογία τεχνικής δυσκολίας αν είχε.

Μετά την απάντηση, το σύστημα περνάει στην αποθήκευση. Το μήνυμα του φοιτητή μπαίνει στο ιστορικό όπως είναι. Η απάντηση του συστήματος αποθηκεύεται σε δύο εκδοχές, μία με όλα τα εσωτερικά σύμβολα, για να μπορεί το σύστημα να την ξαναδιαβάσει αργότερα, και μία καθαρή, χωρίς σύμβολα, που είναι αυτή που βλέπει ο φοιτητής.

Αν η λύση δεν ήταν σωστή, ενημερώνεται και το ιστορικό λαθών: η κατηγορία λάθους μπαίνει σε έναν μετρητή, και οι δέκα συχνότερες κρατούνται. Εδώ ακριβώς κλείνει ο κύκλος ανάμεσα στους τρεις πράκτορες, ένα λάθος που εντόπισε ο Debugger σε μία άσκηση γίνεται πληροφορία που θα χρησιμοποιήσει ο Mentor σε μια άλλη, ακόμα και σε διαφορετική ενότητα.

Τέλος, δύο σημεία ελέγχουν αν το πρόγραμμα έχει τελειώσει. Στην αρχή, αν ο φοιτητής έχει ήδη ξεπεράσει όλες τις ενότητες, το γράφημα δεν καλείται καν, επιστρέφεται απευθείας το συγκεντρωτικό μήνυμα στατιστικών. Στο τέλος, τη στιγμή που λύνεται σωστά η τελευταία άσκηση της τελευταίας ενότητας, η κανονική απάντηση αντικαθίσταται από το ίδιο μήνυμα.

4.8.4 Εξάσκηση, Αξιολόγηση Κώδικα και έναρξη συνεδρίας

Οι δύο συμπληρωματικές λειτουργίες ακολουθούν πολύ πιο σύντομη διαδρομή. Στην Εξάσκηση, η ανάθεση άσκησης και η υποβολή είναι δύο ξεχωριστά σημεία εισόδου. Το πρώτο δέχεται τις ενότητες που επέλεξε ο φοιτητής και τις διασταυρώνει με όσες έχει πράγματι ολοκληρώσει σύμφωνα με τη βάση, κρατώντας μόνο τις έγκυρες και απορρίπτοντας το αίτημα αν δεν μείνει καμία. Από αυτές διαλέγει τυχαία μία και παράγει άσκηση, με δυσκολία που ορίζεται από το τρέχον σερί σωστών απαντήσεων, δύσκολη από τρεις συνεχόμενες και πάνω. Το δεύτερο δέχεται τον κώδικα μαζί με την εκφώνηση και τα κριτήρια που είχαν σταλεί, συγκροτεί μικρή κατάσταση μόλις πέντε πεδίων, και καλεί το ίδιο ακριβώς γράφημα. Μετά την απάντηση ενημερώνει δύο μετρητές, τον γενικό και εκείνον της συγκεκριμένης ενότητας, αυξάνοντας και τους δύο σε σωστή απάντηση και μηδενίζοντάς τους σε λάθος.

Στην Αξιολόγηση Κώδικα η διαδρομή είναι ακόμα πιο σύντομη. Ο κώδικας ελέγχεται πρώτα ως προς την ύπαρξη και το μέγεθός του, με ανώτατο όριο δύο χιλιάδες χαρακτήρες. Έπειτα συγκροτείται κατάσταση έξι πεδίων, όπου η προαιρετική περιγραφή που έγραψε ο φοιτητής μπαίνει στη θέση της εκφώνησης, και τα κριτήρια επιτυχίας μένουν κενά. Δεν διαβάζεται ιστορικό, δεν ενημερώνεται κανένα πεδίο, δεν αποθηκεύεται τίποτα. Η λειτουργία αυτή δεν αφήνει κανένα ίχνος στην πορεία του φοιτητή, κάτι που αντιστοιχεί ακριβώς στον σκοπό της.

Ένα τελευταίο σημείο εισόδου εκτελείται μία φορά σε κάθε σύνδεση. Δημιουργεί το αναγνωριστικό της νέας συνεδρίας, συνθέτει το μήνυμα καλωσορίσματος, και υπολογίζει το μοντέλο προόδου που εξετάζεται στην επόμενη ενότητα. Για φοιτητή που επιστρέφει, το μήνυμα καλωσορίσματος περιέχει την περίληψη της προηγούμενης συνεδρίας. Η κλήση αυτή έχει δικό της, πολύ αυστηρότερο χρονικό όριο, οκτώ δευτερόλεπτα αντί για εξήντα, ακριβώς επειδή είναι το πρώτο πράγμα που περιμένει ο φοιτητής μόλις συνδεθεί, δεν έχει τίποτα άλλο στην οθόνη να κοιτάζει όσο περιμένει. Σε περίπτωση αποτυχίας χρησιμοποιείται σταθερό κείμενο καλωσορίσματος που αναφέρει τουλάχιστον σε ποια ενότητα βρίσκεται ο φοιτητής.

Το επίπεδο ενορχήστρωσης απαντά επομένως σε ερωτήματα που έχουν μία σωστή απάντηση: ποια ενότητα είναι η τρέχουσα, πόσες προσπάθειες έγιναν, ποια άσκηση ισχύει, τι πρέπει να αποθηκευτεί. Είναι γι' αυτό σχεδόν εξ ολοκλήρου ντετερμινιστικό, με μόνη εξαίρεση τις δύο κλήσεις ταξινόμησης

και την περίληψη συνεδρίας, όπου το ερώτημα αφορά πια τι εννοεί ο φοιτητής ή πώς θα διατυπωθεί κάτι σε φυσική γλώσσα. Είναι, δηλαδή, το ίδιο μοτίβο ντετερμινιστικής θεμελίωσης και κρίσης LLM που καθιερώθηκε από την ενότητα 3.3, εφαρμοσμένο τώρα έξω από τους πράκτορες αντί μέσα σε έναν από αυτούς. Ο ρόλος του απέναντί τους είναι διπλός: εξασφαλίζει ότι λαμβάνουν ακριβή πληροφορία για την κατάσταση του φοιτητή, και ότι τα συμπεράσματά τους επιβιώνουν αφού τελειώσει η κλήση.

4.9 Open Learner Model – Υλοποίηση

Όπως περιγράφηκε στο 3.7, το μοντέλο προόδου δεν αποθηκεύεται πουθενά ξεχωριστά, παράγεται εκ νέου κάθε φορά που ζητείται, από δεδομένα που υπάρχουν ήδη για άλλους λόγους. Η παρούσα ενότητα δείχνει πώς ακριβώς γίνεται αυτός ο υπολογισμός.

Γίνεται σε δύο σημεία, με τον ίδιο ακριβώς τρόπο και τις ίδιες τρεις συναρτήσεις και στα δύο, ώστε να μην υπάρχει περίπτωση να δείχνουν διαφορετικά πράγματα. Το πρώτο είναι η έναρξη κάθε συνεδρίας, ώστε η διεπαφή να έχει το μοντέλο διαθέσιμο από την αρχή. Το δεύτερο είναι ξεχωριστό σημείο εισόδου, που καλείται όταν ο φοιτητής ανοίγει ρητά την προβολή της προόδου του.

Το ποσοστό κάθε ενότητας προκύπτει από τη θέση της σε σχέση με την τρέχουσα. Όσες προηγούνται λαμβάνουν εκατό, όσες έπονται μηδέν, και η τρέχουσα λαμβάνει τιμή που εξαρτάται από το επίπεδο κατανόησης, μέσω ενός σταθερού πίνακα αντιστοίχισης: είκοσι στην ανάγκη υποστήριξης, πενήντα στην εξέλιξη, εβδομήντα πέντε στην καλή κατανόηση, ενενήντα πέντε στην ισχυρή. Ο πίνακας αυτός ορίζεται μέσα στον Assessor και όχι στο σημείο που παράγει το μοντέλο προόδου, ώστε τα τέσσερα επίπεδα και οι αριθμητικές τους αντιστοιχίες να δηλώνονται μαζί, σε ένα μόνο σημείο.

Οι ενδείξεις δυσκολίας ανά ενότητα υπολογίζονται διαφορετικά. Το ιστορικό διαβάζεται εδώ από την αρχή προς το τέλος, όχι προς τα πίσω, με δύο μετρητές να αθροίζουν προσπάθειες και υποδείξεις καθώς προχωρούν. Κάθε φορά που συναντάται σύμβολο προαγωγής, οι δύο μετρητές συγκρίνονται με τα κατώφλια, το αποτέλεσμα καταγράφεται για την ενότητα που μόλις έκλεισε, και οι μετρητές μηδενίζονται για την επόμενη. Το ποια ενότητα είναι αυτή δεν διαβάζεται από πουθενά, προκύπτει απλά από το πλήθος των συμβόλων προαγωγής που έχουν συναντηθεί μέχρι εκείνο το σημείο: το πρώτο σύμβολο κλείνει την πρώτη ενότητα, το δεύτερο τη δεύτερη, και ούτω καθεξής. Ένα ιστορικό με τρία σύμβολα προαγωγής περιγράφει έτσι τρεις ολοκληρωμένες ενότητες, με τη σειρά που εμφανίζονται.

Τα κατώφλια, τρεις προσπάθειες και δύο υποδείξεις για να ενεργοποιηθεί η ένδειξη δυσκολίας, πέντε συνεχόμενες σωστές απαντήσεις στην Εξάσκηση για να αρθεί, δηλώνονται ως ονομασμένες σταθερές σε ένα σημείο, όχι ως αριθμοί σκόρπιοι μέσα σε συνθήκες. Η απόσταση ανάμεσα στα δύο όρια είναι σχεδιαστική επιλογή, όχι σύμπτωση, για τον λόγο που εξηγήθηκε στην ενότητα 3.7. Οι ίδιες τιμές χρησιμοποιούνται και από τον Assessor, εκεί όμως γραμμένες απευθείας μέσα στη συνθήκη, όχι μέσω της ίδιας ονομασμένης σταθεράς. Πρόκειται για την ίδια μορφή κατανεμημένης γνώσης, σε μικρότερη κλίμακα, με αυτή που περιγράφεται στην ενότητα 4.11, στην πρόκληση του συγχρονισμού της κοινής ονοματολογίας.

Η άρση της ένδειξης εφαρμόζεται στο τελευταίο στάδιο του υπολογισμού. Αφού συγκεντρωθούν οι ενδείξεις δυσκολίας, ελέγχεται για καθεμία αν ο μετρητής σωστών απαντήσεων της αντίστοιχης ενότητας στην Εξάσκηση έχει φτάσει το κατώφλι, οπότε η ένδειξη αφαιρείται πριν καν φτάσει στη διεπαφή. Οι μετρητές αυτοί αποθηκεύονται ως μικρή δομή αντιστοίχισης ενότητας σε αριθμό, μέσα σε πεδίο κειμένου, με τον ίδιο τρόπο που περιγράφηκε στην ενότητα 4.3 για τα υπόλοιπα μικρά σύνολα δεδομένων.

Το τρίτο στοιχείο του μοντέλου, η σύγκριση με τους υπόλοιπους χρήστες, προκύπτει από ένα μόνο ερώτημα στη βάση, το οποίο ανακτά την τρέχουσα ενότητα όλων των χρηστών. Από αυτή τη λίστα υπολογίζεται, για κάθε ενότητα, το ποσοστό όσων έχουν φτάσει σε αυτήν ή την έχουν προσπεράσει.

Το τελικό αποτέλεσμα είναι μια λίστα με μία εγγραφή ανά ενότητα, που περιέχει αναγνωριστικό, τίτλο, ποσοστό κατάκτησης, ένδειξη δυσκολίας, και ποσοστό της συνολικής ομάδας χρηστών. Παραδίδεται στη διεπαφή όπως είναι.

Σε ολόκληρη τη διαδικασία δεν καλείται LLM, και η επιλογή είναι σκόπιμη. Πρόκειται για την καταγραφή της ίδιας της πορείας του φοιτητή, την οποία εκείνος θα χρησιμοποιήσει για να αποφασίσει πού χρειάζεται περισσότερη δουλειά. Ότι του παρουσιάζεται εδώ πρέπει να αντιστοιχεί ακριβώς σε όσα έχουν καταγραφεί, χωρίς καμία ενδιάμεση διατύπωση που θα μπορούσε να το αλλοιώσει. Οι αριθμοί και οι ενδείξεις φτάνουν στη διεπαφή ακατέργαστοι, και η μόνη ερμηνεία τους γίνεται από τον ίδιο τον φοιτητή.

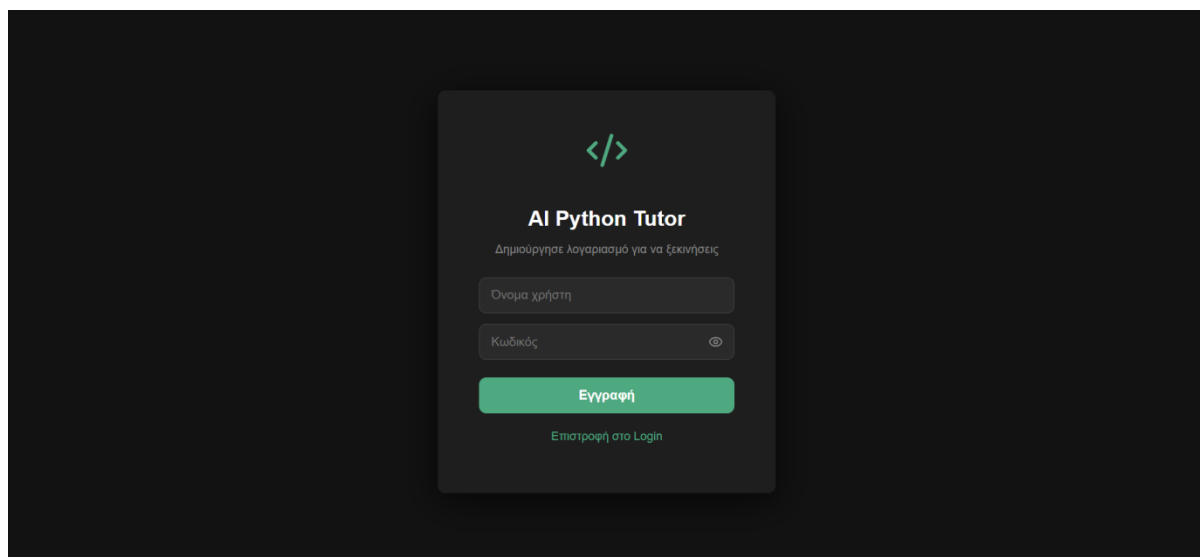
Η μοναδική στιγμή που το ίδιο υλικό περνά από γλωσσική επεξεργασία είναι το τελικό μήνυμα ολοκλήρωσης του προγράμματος, το οποίο όμως συντίθεται από σταθερό πρότυπο και όχι από μοντέλο. Αντλεί τα συγκεντρωτικά στοιχεία από τα πεδία του φοιτητή, μεταφράζει τις τρεις συχνότερες κατηγορίες λαθών σε αναγνώσιμες περιγραφές μέσω του κοινού πίνακα ετικετών, και προσθέτει πρόταση για στοχευμένη εξάσκηση αν κάποια ενότητα έχει παραμείνει σημειωμένη ως δύσκολη. Ακόμα κι εκεί, δηλαδή, το περιεχόμενο παραμένει ακριβώς αυτό που έχει καταγραφεί.

4.10 Frontend & Διεπαφή Χρήστη

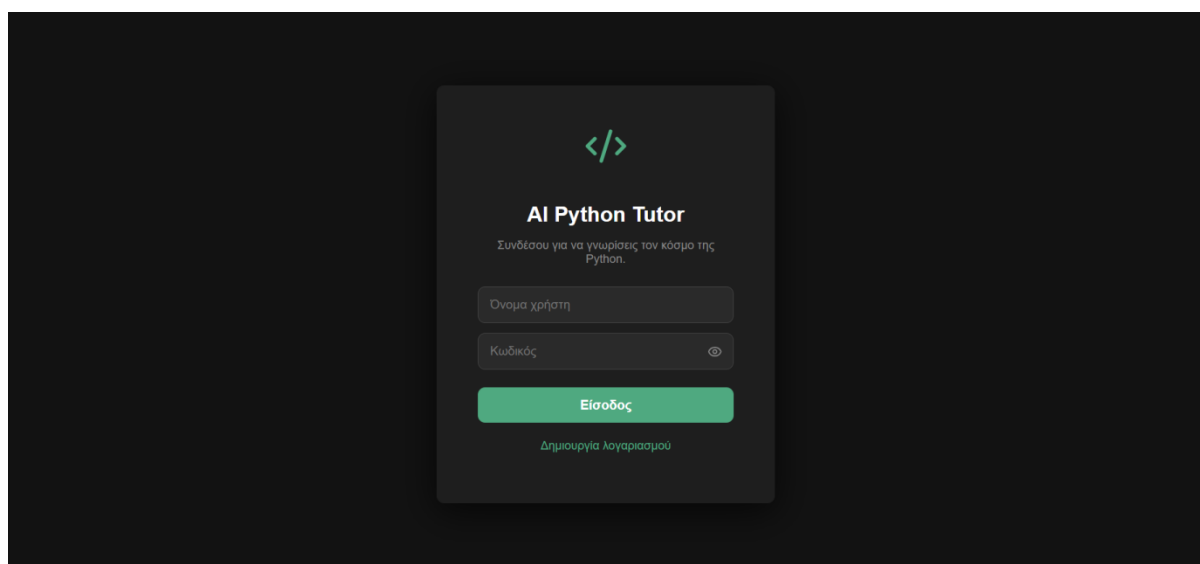
Η διεπαφή είναι μονοσέλιδη εφαρμογή React χωρίς βιβλιοθήκη δρομολόγησης, με πέντε οθόνες. Η Οθόνη Σύνδεσης είναι ορατή όσο δεν υπάρχει συνδεδεμένος χρήστης, ενώ ποια από τις υπόλοιπες τέσσερις εμφανίζεται, η Αρχική Οθόνη, η Διδασκαλία, η Εξάσκηση ή η Αξιολόγηση Κώδικα, καθορίζεται από μία μεταβλητή κατάστασης. Η επιλογή αυτή είναι συνεπής με το ότι η μόνιμη κατάσταση ζει εξ ολοκλήρου στο backend. Το frontend δεν κρατά δικό του μοντέλο της προόδου του φοιτητή, κρατά μόνο ό,τι χρειάζεται για να αποδώσει την τρέχουσα οθόνη, και το αποθηκευμένο περιεχόμενο του φυλλομετρητή καθαρίζεται τόσο στην εκκίνηση όσο και στην αποσύνδεση, ώστε να μην παραμένει καμία πληροφορία χρήστη τοπικά.

4.10.1 Πιστοποίηση και αρχική οθόνη

Η Οθόνη Σύνδεσης εξυπηρετεί τόσο την εγγραφή (Σχήμα 4.2) όσο και τη σύνδεση (Σχήμα 4.3), εναλλάσσοντας τα στο ίδιο σημείο εισόδου. Τα μηνύματα σφάλματος διακρίνονται ανάλογα με την απόκριση του server, με χωριστό κείμενο για λανθασμένα στοιχεία σύνδεσης και για όνομα χρήστη που υπάρχει ήδη.



Σχήμα 4.2. Οθόνη εγγραφής

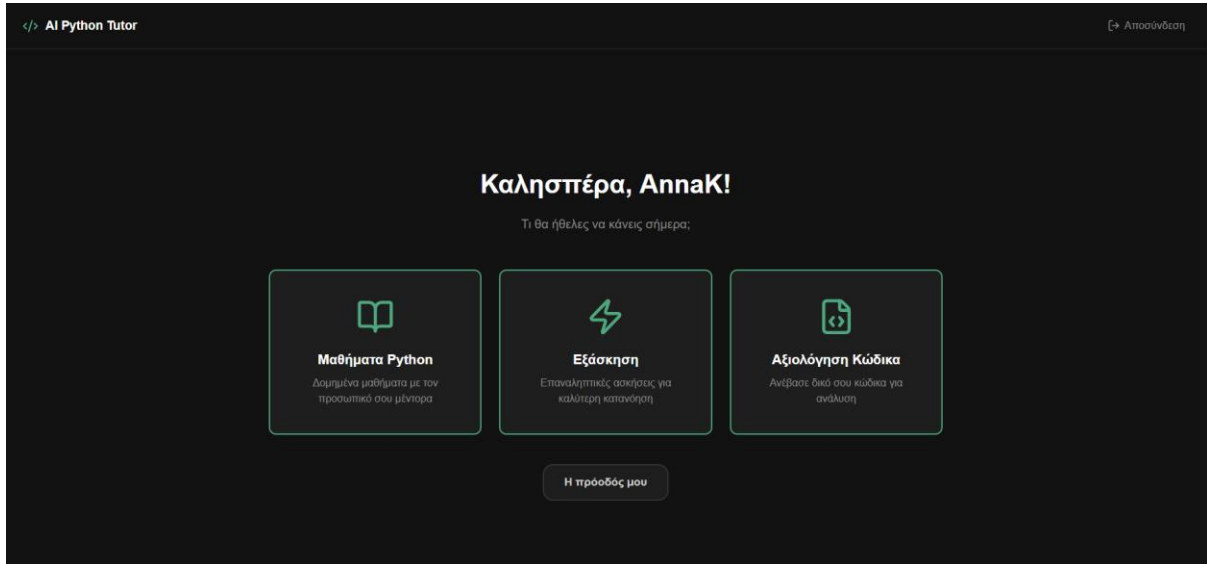


Σχήμα 4.3. Οθόνη σύνδεσης

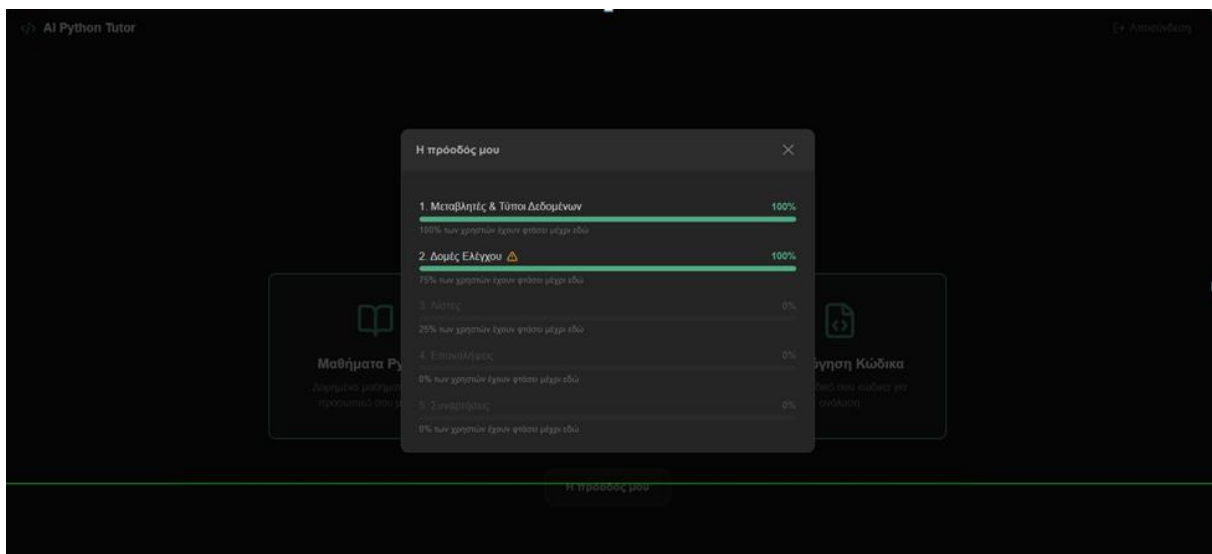
Δύο μηχανισμοί εδώ προέκυψαν από το ίδιο το περιβάλλον φιλοξενίας, το οποίο αναστέλλει την εφαρμογή όταν μένει αδρανής για ένα διάστημα, με αποτέλεσμα το πρώτο αίτημα μετά από τέτοια περίοδο να καθυστερεί αισθητά. Ο πρώτος μηχανισμός είναι ένα αίτημα αφύπνισης προς το backend, το οποίο στέλνεται σιωπηλά μόλις φορτώσει η σελίδα, πριν καν ο φοιτητής πατήσει οτιδήποτε, ώστε ο server να έχει ήδη ξεκινήσει να ξυπνά. Ο δεύτερος είναι μια αυτόματη επανάληψη του αιτήματος πιστοποίησης μετά από έξι δευτερόλεπτα, αν το πρώτο αποτύχει για λόγο που δεν αφορά λανθασμένα στοιχεία, συνοδευόμενη από μήνυμα που εξηγεί στον φοιτητή ότι ο server ξυπνά. Χωρίς αυτά τα δύο, η πρώτη προσπάθεια σύνδεσης μετά από περίοδο αδράνειας θα εμφανιζόταν στον φοιτητή σαν απλή αποτυχία, χωρίς καμία εξήγηση.

Η αρχική οθόνη (Σχήμα 4.4) δίνει πρόσβαση στις τρεις λειτουργίες και φιλοξενεί την προβολή του μοντέλου προόδου. Σε αυτό κάθε ενότητα εμφανίζεται με μπάρα, ποσοστό και τίτλο, με το χρώμα της μπάρας να αλλάζει κλιμακωτά ανάλογα με την τιμή. Μια ενότητα που δεν έχει ξεκινήσει καθόλου δεν

εμφανίζει γεμάτη μπάρα και ο τίτλος της αποδίδεται σε πιο σβηστό χρώμα, ώστε να διακρίνεται οπτικά από μια ενότητα με απλά χαμηλό ποσοστό, η οπτική αντιστοιχία της διάκρισης που περιγράφηκε στην προηγούμενη ενότητα. Δίπλα στον τίτλο εμφανίζεται η ένδειξη δυσκολίας όπου υπάρχει, με επεξηγηματικό κείμενο κατά την επισήμανση, και κάτω από κάθε μπάρα το ποσοστό των υπόλοιπων χρηστών που έχουν φτάσει μέχρι εκείνη την ενότητα (Σχήμα 4.5).



Σχήμα 4.4. Αρχική οθόνη

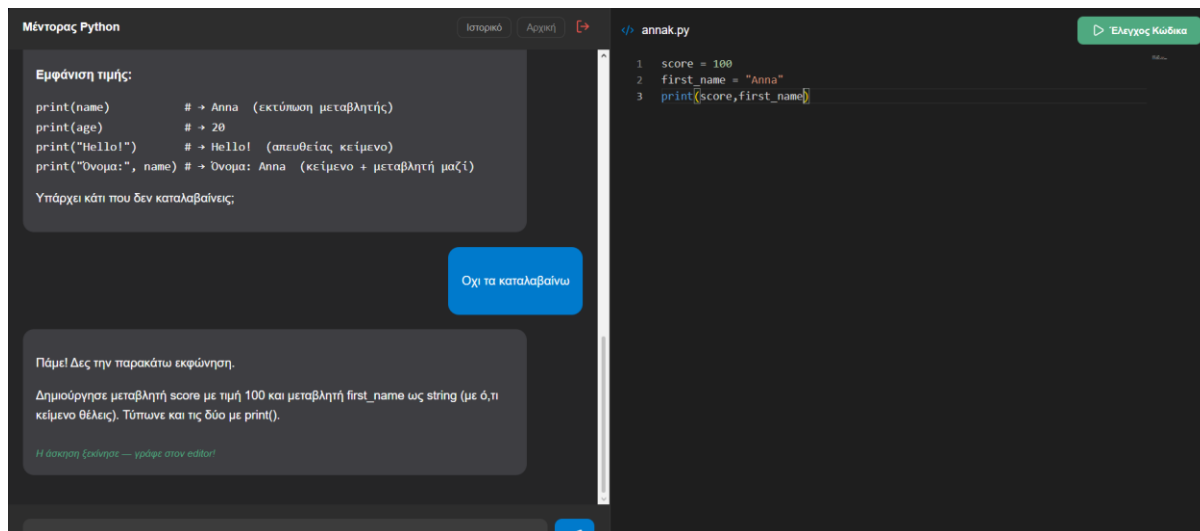


Σχήμα 4.5. Μοντέλο προόδου με ένδειξη δυσκολίας

4.10.2 Ο επεξεργαστής κώδικα και ο χρονιστής αδράνειας

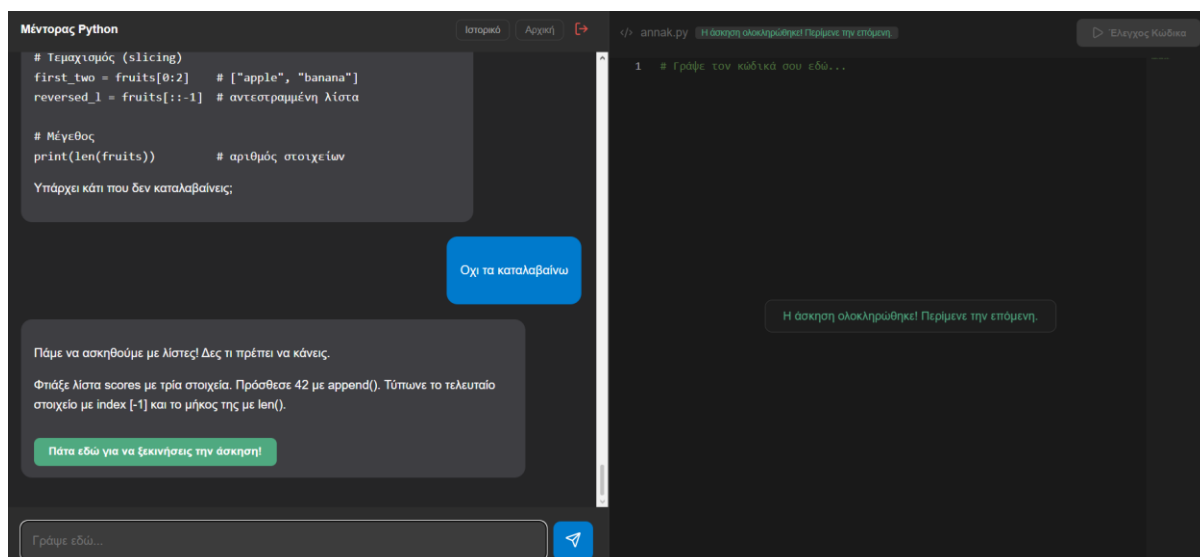
Η οθόνη διδασκαλίας χωρίζεται σε δύο μέρη, τη συνομιλία και τον επεξεργαστή κώδικα (editor) (Σχήμα 4.6). Ο επεξεργαστής είναι το Monaco, ρυθμισμένο για Python με σκούρο θέμα, και το όνομα του αρχείου που εμφανίζεται παράγεται από το όνομα χρήστη. Ο Mentor γράφει τις απαντήσεις του χρησιμοποιώντας Markdown όπου χρειάζεται, δηλαδή σύμβολα για έντονα γράμματα, λίστες ή μπλοκ

κώδικα μέσα στο κείμενο. Το react-markdown αποδίδει σωστά αυτή τη μορφοποίηση, τόσο σε ελεύθερο κείμενο του μοντέλου όσο και στα προδιατυπωμένα κείμενα που περιέχουν σκόπιμα έντονα σημεία και αποσπάσματα κώδικα.



Σχήμα 4.6. Ενεργή άσκηση, ο φοιτητής γράφει κώδικα

Ο επεξεργαστής παραμένει κλειδωμένος όσο δεν υπάρχει ενεργή άσκηση. Προκειμένου αυτό το κλειδώμα να είναι εμφανές στον χρήστη χωρίς να προκαλέσει σύγχυση, πάνω από τον επεξεργαστή εμφανίζεται επικάλυψη με ένα μήνυμα που τον ενημερώνει σχετικά (Σχήμα 4.7).



Σχήμα 4.7. Κλειδωμένος editor όσο δεν υπάρχει ενεργή άσκηση

Το κουμπί έναρξης της άσκησης εμφανίζεται όταν το κείμενο του μηνύματος περιέχει το αντίστοιχο εσωτερικό σύμβολο, το `[BUTTON:START_TASK]`. Η διεπαφή πρέπει επομένως να κάνει δύο πράγματα πάνω σε κάθε μήνυμα που λαμβάνει. Πρώτον, να αφαιρέσει όλα τα σύμβολα πριν το

εμφανίσει στον φοιτητή, και δεύτερον αναλόγως ποια σύμβολα είναι αυτά να αποφασίσει τα στοιχεία ελέγχου που πρέπει να εμφανίσει.

Ο χρονιστής αδράνειας είναι ο μηχανισμός με τον οποίο το σύστημα παίρνει πρωτοβουλία χωρίς να το ζητήσει ο φοιτητής. Η παιδαγωγική λογική πίσω από αυτόν τεκμηριώθηκε ήδη στην ενότητα 3.4.2. Λειτουργεί σε τρία στάδια που κλιμακώνονται, με χρόνους που διαφέρουν ανάλογα με το επίπεδο εμπειρίας (Πίνακας 4.1).

Πίνακας 4.1: Χρονικά όρια του χρονιστή αδράνειας ανά στάδιο

Στάδιο	Προχωρημένος	Αρχάριος
1 (ενθάρρυνση)	40s	70s
2 (γενική υπόδειξη)	60s	90s
3 (συγκεκριμένη υπόδειξη)	90s	120s

Ο αρχάριος έχει σε κάθε στάδιο περισσότερο χρόνο, αφού χρειάζεται περισσότερη ώρα πριν η σιωπή του σημαίνει πραγματική δυσκολία και όχι απλή σκέψη. Ο χρονιστής μετρά από τη χρονική στιγμή της τελευταίας ενέργειας του φοιτητή. Αν ο φοιτητής έγραψε κάτι πριν είκοσι δευτερόλεπτα, ο χρονιστής μετρά ξανά από εκεί, έτσι ώστε όταν για παράδειγμα κάνει κάποια θεωρητική ερώτηση κατά τη διάρκεια της άσκησης, να μη μετρήσει σαν χαμένος χρόνος. Όταν ο φοιτητής υποβάλει κώδικα, ο χρονιστής παύει να ενεργοποιείται μέχρι την επόμενη άσκηση.

Κάθε ολοκλήρωση του χρονιστή στέλνει αίτημα χωρίς κώδικα, με ειδική ένδειξη τύπου γεγονότος, ώστε το backend να καταλαβαίνει ότι δεν πρόκειται για ερώτηση του φοιτητή. Η ένδειξη αυτή οδηγεί εκεί σε τρεις διαφορετικές περιγραφές κατάστασης, μία για κάθε στάδιο. Η πρώτη ζητά μόνο ενθάρρυνση, με ρητή οδηγία να μη δοθεί ακόμα καμία υπόδειξη. Η δεύτερη ζητά μία υπόδειξη για το τι χρειάζεται, χωρίς κώδικα. Η τρίτη ζητά πιο συγκεκριμένη υπόδειξη για τη δομή (Σχήμα 4.8). Από το δεύτερο στάδιο και μετά προστίθεται και το προηγούμενο λάθος του φοιτητή, εφόσον υπάρχει καταγεγραμμένο. Μετά το τρίτο στάδιο ο χρονιστής δεν ενεργοποιείται ξανά, μέχρι ο φοιτητής να κάνει κάποια ενέργεια ή να υποβάλει κώδικα.

Υποβολή κώδικα:

```
score = 60
if score >= 60:
    print("High")
else:
    print("Low")
```

Καταλαβαίνω ότι η άσκηση φαίνεται δύσκολη· είναι απολύτως φυσιολογικό. Ας δούμε:

Το πρόβλημά σου είναι η συνθήκη του `if`. Ελέγχεις αν η τιμή του `score` είναι μεγαλύτερη ή ίση με 60, ενώ η εκφώνηση ζητά να ελέγξεις αν είναι μεγαλύτερη από 100· έτσι για την τιμή 60 τυπώνεται «High» αντί για «Low». Διόρθωσε τη σύγκριση ώστε να συγκρίνει το `score` με 100, όχι με 60.

Σχήμα 4.8. Παράδειγμα συγκεκριμένης υπόδειξης

Στη διεπαφή ανήκει τέλος και η μέτρηση δύο μεγεθών που χρησιμοποιεί αργότερα ο Assessor, ο χρόνος και η ένδειξη προσπάθειας. Ο server δεν μπορεί να τα υπολογίσει μόνος του, γνωρίζει μόνο πότε έφτασε το αίτημα, όχι πότε ο φοιτητής άρχισε να σκέφτεται ή να γράφει. Ο χρόνος μετρίεται λοιπόν στον browser, από τη στιγμή που εμφανίστηκε η άσκηση μέχρι την υποβολή. Ένα πλήρες παράδειγμα επιτυχούς ολοκλήρωσης φαίνεται στο Σχήμα 4.9.

Μέντορας Python

Ιστορικό
Άρχει
➔

Πάμε! Δες την παρακάτω εκφώνηση.

Δημιούργησε μεταβλητή `score` με τιμή 100 και μεταβλητή `first_name` ως string (με ό,τι κείμενο θέλεις). Τύπωνε και τις δύο με `print()`.

Πάτα εδώ για να ξεκινήσεις την άσκηση!

Υποβολή κώδικα:

```
score = 100
first_name = "Anna"
print(score, first_name)
```

Μπράβο σου! Η λύση σου στην άσκηση «Μεταβλητές & Τύποι Δεδομένων» ήταν τέλεια. Νιώθω περήφανος για την προσπάθειά σου. Θέλεις να προχωρήσουμε στις «Δομές Ελέγχου» και να δούμε πώς λειτουργούν οι βρόχοι και οι συνθήκες;

➔

annak.py
Η άσκηση ολοκληρώθηκε! Περίμενε την επόμενη.
Ελέγχος Κώδικα

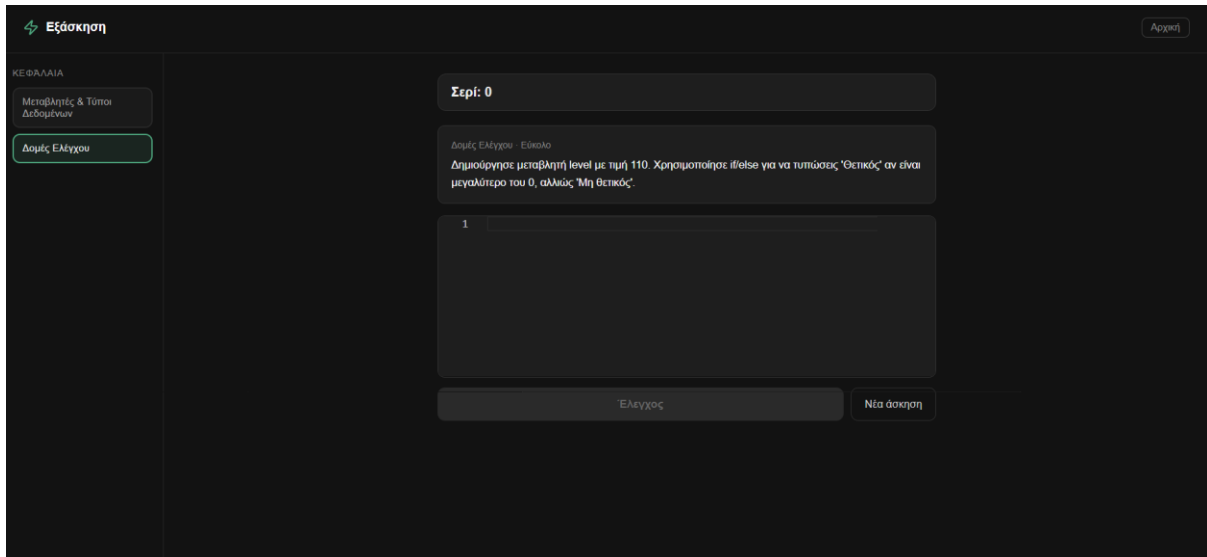
```
1 score = 100
2 first_name = "Anna"
3 print(score, first_name)
```

Η άσκηση ολοκληρώθηκε! Περίμενε την επόμενη.

Σχήμα 4.9. Επιτυχής ολοκλήρωση άσκησης

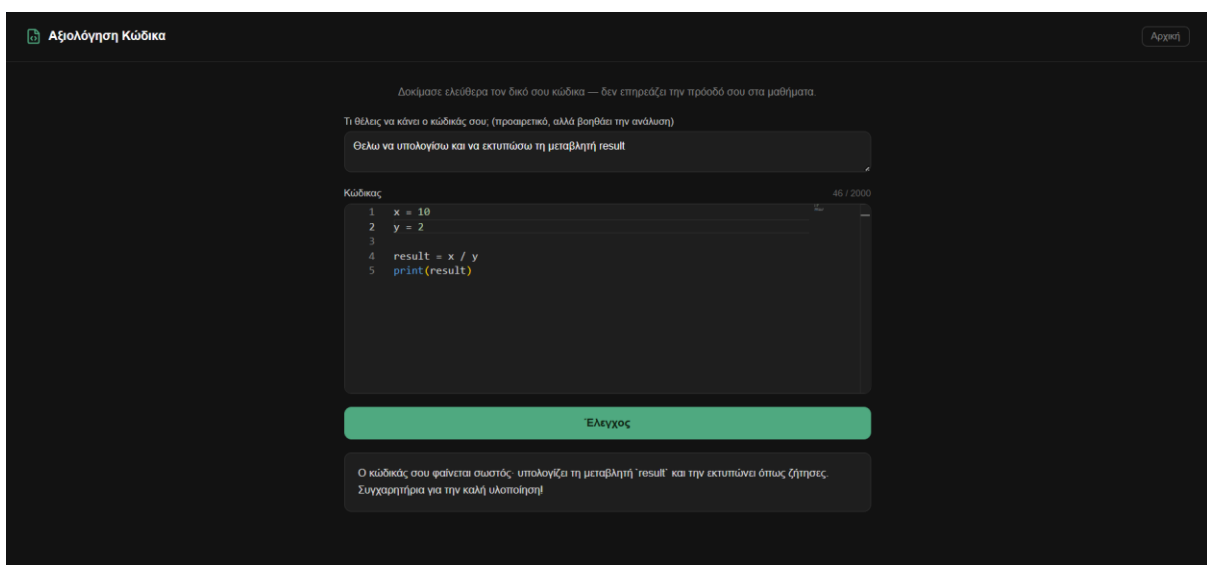
4.10.3 Συμπληρωματικές οθόνες και ιστορικό

Η οθόνη της Εξάσκησης (Σχήμα 4.10) ξεκινά με επιλογή ενοτήτων. Εμφανίζονται μόνο όσες έχουν ποσοστό εκατό, δηλαδή έχουν ολοκληρωθεί, και ο φοιτητής μπορεί να επιλέξει περισσότερες από μία. Το σερί συνεχόμενων σωστών απαντήσεων εμφανίζεται διαρκώς. Μετά από σωστή απάντηση ο επεξεργαστής κλειδώνει, ώστε ο κώδικας που μόλις κρίθηκε να μην αλλάξει πριν ζητηθεί η επόμενη άσκηση.



Σχήμα 4.10. Οθόνη Εξάσκησης

Η οθόνη της Αξιολόγησης Κώδικα (Σχήμα 4.11) δέχεται τον κώδικα και μια προαιρετική περιγραφή του τι υποτίθεται ότι κάνει. Πάνω από τον επεξεργαστή εμφανίζεται ζωντανός μετρητής χαρακτήρων με το ανώτατο όριο των δύο χιλιάδων, ο οποίος αλλάζει χρώμα μόλις αυτό ξεπεραστεί, ώστε ο φοιτητής να το αντιληφθεί καθώς γράφει και όχι μετά την υποβολή.



Σχήμα 4.11. Οθόνη Αξιολόγησης Κώδικα

Και οι τρεις λειτουργίες προστατεύονται από ακούσια έξοδο. Αν ο φοιτητής επιχειρήσει να επιστρέψει στην αρχική οθόνη ενώ έχει ενεργή άσκηση ή γραμμένο κώδικα, εμφανίζεται επιβεβαίωση. Στην κύρια ροή, η επιβεβαιωμένη έξοδος καλεί το σημείο εισόδου που καθαρίζει την ενεργή άσκηση, και σημειώνεται ώστε κατά την επιστροφή να ζητηθεί καινούργια άσκηση αντί να συνεχιστεί η προηγούμενη.

Η διεπαφή δίνει τέλος πρόσβαση στις προηγούμενες συνεδρίες. Ένα πλευρικό πλαίσιο εμφανίζει τη λίστα τους, εξαιρώντας την τρέχουσα, με την ημερομηνία καθεμιάς αποδοσμένη σε αναγνώσιμη μορφή που διακρίνει το σήμερα και το χθες από τις παλαιότερες. Επιλέγοντας μία, ο φοιτητής βλέπει ολόκληρη τη συνομιλία εκείνης της φοράς σε ξεχωριστό παράθυρο. Η δυνατότητα αυτή στηρίζεται στο ότι κάθε σύνδεση παράγει δικό της αναγνωριστικό συνεδρίας, όπως περιγράφηκε στην ενότητα 4.3, χωρίς να χρειάζεται καμία πρόσθετη δομή αποθήκευσης.

Ο ρόλος του frontend είναι σαφώς οριοθετημένος και τον εξυπηρετεί πλήρως. Αποδίδει ό,τι λαμβάνει, μετρά τον χρόνο και τις ενέργειες του φοιτητή, μεταφράζει τα εσωτερικά σύμβολα σε στοιχεία ελέγχου της οθόνης, και ειδοποιεί το backend κάθε στιγμή που απαιτεί απόκριση. Οι παιδαγωγικές αποφάσεις, πότε δίνεται υπόδειξη, ποια άσκηση ανατίθεται, αν μια λύση είναι σωστή, ανήκουν εξ ολοκλήρου στους πράκτορες, και η διεπαφή σχεδιάστηκε ώστε να μην τις αναπαράγει πουθενά. Ο χρονιστής αδράνειας είναι το μόνο σημείο όπου η διεπαφή παίρνει πρωτοβουλία, ακριβώς επειδή είναι και το μόνο σημείο του συστήματος που ξέρει ότι ο φοιτητής δεν έκανε τίποτα.

4.11 Προκλήσεις Υλοποίησης

Η ανάπτυξη του συστήματος ανέδειξε δυσκολίες που δεν ήταν ορατές στον σχεδιασμό. Η ενότητα αυτή εξετάζει όσες επηρέασαν ουσιαστικά την υλοποίηση, και πώς αντιμετωπίστηκαν.

Η πρώτη δυσκολία αφορά το χρονικό όριο ενός πλήρους κύκλου. Μια υποβολή κώδικα ενεργοποιεί τρεις διαδοχικές κλήσεις μοντέλου, μία για κάθε πράκτορα, και καμία δεν μπορεί να ξεκινήσει πριν ολοκληρωθεί η προηγούμενη, αφού κάθε μία στηρίζεται στο αποτέλεσμα της προηγούμενης. Ο συνολικός χρόνος απόκρισης είναι επομένως ένα άθροισμα. Η διαπίστωση αυτή καθόρισε τρεις αποφάσεις: την επιλογή παρόχου με κριτήριο την ταχύτητα και όχι μόνο την ποιότητα, τον ορισμό ανώτατου ορίου εξήντα δευτερολέπτων για ολόκληρο τον κύκλο, ώστε μια αργή κλήση να μην κρατά τον φοιτητή σε αναμονή επ' αόριστον, και την ύπαρξη προκαθορισμένου κειμένου σε κάθε σημείο όπου μπορεί να χαθεί μια απάντηση. Το ίδιο σκεπτικό εφαρμόστηκε ακόμα πιο αυστηρά στην περίληψη της προηγούμενης συνεδρίας, όπου το όριο είναι μόλις οκτώ δευτερόλεπτα, επειδή εκεί ο φοιτητής μόλις έχει συνδεθεί και δεν έχει τίποτα άλλο να κοιτάζει στην οθόνη.

Η δεύτερη δυσκολία αφορά τη φυσική γλώσσα των χρηστών. Οι ελληνόφωνοι χρήστες γράφουν συχνά με λατινικούς χαρακτήρες, με ορθογραφικά λάθη, και χωρίς τόνους. Οι εκτενείς κατάλογοι φράσεων που περιγράφηκαν στην ενότητα 4.5 δεν σχεδιάστηκαν εξ αρχής στη μορφή που έχουν, διαμορφώθηκαν σταδιακά καθώς εμφανίζονταν διατυπώσεις που το σύστημα δεν αναγνώριζε. Μια ιδιαίτερη κατηγορία δυσκολίας αφορά την άρνηση, η οποία στα ελληνικά δεν σημαίνει πάντα άρνηση. Μια απάντηση όπως «όχι, αυτά τα κατάλαβα» ξεκινά με αρνητικό μόριο ενώ δηλώνει ετοιμότητα. Η αντιμετώπιση δεν ήταν να αφεθεί το ζήτημα εξ ολοκλήρου στο μοντέλο, αλλά να αναγνωρίζονται ρητά οι πιο συχνές τέτοιες δομές, ώστε η ερμηνεία τους να είναι σταθερή και όχι να εξαρτάται από την εκάστοτε απόκριση.

Η τρίτη δυσκολία είναι ο συγχρονισμός της κοινής ονοματολογίας. Οι κατηγορίες σφαλμάτων ορίζονται στον Debugger αλλά χρησιμοποιούνται σε τρία σημεία, για την επιλογή παιδαγωγικής εξήγησης στον Mentor, για την καταγραφή στο ιστορικό λαθών, και για την εμφάνιση στο τελικό μήνυμα ολοκλήρωσης. Η γνώση των ονομάτων είναι επομένως κατανοητή, και ο κώδικας φέρει ρητό σχόλιο που υπενθυμίζει ότι τα τρία σημεία πρέπει να μένουν συγχρονισμένα. Κατά τον έλεγχο του συστήματος διαπιστώθηκε ότι το τελικό μήνυμα χρησιμοποιούσε δικό του, ξεχωριστό πίνακα μεταφράσεων, ο οποίος είχε αποκλίνει από τις πραγματικές ονομασίες, με αποτέλεσμα ορισμένες κατηγορίες να εμφανίζονται με τον εσωτερικό τους κωδικό αντί για αναγνώσιμη περιγραφή. Η διόρθωση δεν ήταν η ενημέρωση του δεύτερου πίνακα, ήταν η κατάργησή του, ώστε να υπάρχει ένας μόνο κατάλογος περιγραφών για όλο το σύστημα. Το περιστατικό επιβεβαίωσε κάτι γενικότερο: σε κατανοητή γνώση ο συγχρονισμός δεν εξασφαλίζεται με υπενθύμιση, εξασφαλίζεται με την απλοποίηση της επανάληψης.

Η τέταρτη δυσκολία είναι η εξάρτηση από εξωτερικό πάροχο. Κατά τη διάρκεια της εργασίας ο πάροχος απέσυρε δύο φορές μοντέλα που χρησιμοποιούσε το σύστημα. Τη δεύτερη φορά η ανακοίνωση αφορούσε ταυτόχρονα και τα δύο μοντέλα που εξυπηρετούσαν τότε το σύνολο των κλήσεων, με προθεσμία μίας ημέρας. Το γεγονός έδειξε ότι η επιλογή μοντέλου δεν είναι μόνιμη σχεδιαστική απόφαση, είναι εξωτερική εξάρτηση με δικό της κύκλο ζωής. Καμία από τις δύο μεταβάσεις δεν απαιτούσε αλλαγή στη λογική των πρακτόρων. Η πρώτη ωστόσο απαιτούσε επέμβαση σε κάθε αρχείο που δήλωνε μοντέλο ξεχωριστά, γεγονός που οδήγησε στη συγκεντρωση της δήλωσης σε ένα σημείο, όπως περιγράφηκε στην ενότητα 4.2, ώστε η επόμενη μετάβαση να είναι μονογραμμική. Η επαλήθευση μετά από κάθε μετάβαση δεν αφορούσε την ποιότητα του κειμένου, αφορούσε τη συμμόρφωση με τη μορφή εξόδου, αν δηλαδή οι σταθερές ετικέτες και ο κλειστός κατάλογος κατηγοριών εξακολουθούν να τηρούνται, αφού μια αποτυχία εκεί δεν εμφανίζεται ως σφάλμα, εμφανίζεται ως σιωπηλή πτώση στο εναλλακτικό μονοπάτι.

Η πέμπτη και τελευταία δυσκολία είναι η φιλοξενία με αναστολή. Η υπηρεσία φιλοξενίας αναστέλλει την εφαρμογή όταν παραμείνει αδρανής, και το πρώτο αίτημα μετά από τέτοια περίοδο καθυστερεί αισθητά. Η δυσκολία εδώ δεν είναι τεχνική, είναι ζήτημα εντύπωσης του χρήστη, μια αργή πρώτη σύνδεση μοιάζει με αποτυχία. Οι δύο μηχανισμοί που περιγράφηκαν στην ενότητα 4.10.1 αντιμετωπίζουν ακριβώς αυτό, μετατρέποντας μια ανεξήγητη αναμονή σε κατανοητή.

Ένα κοινό στοιχείο διατρέχει όλες αυτές τις δυσκολίες. Καμία δεν αφορούσε την ικανότητα του μοντέλου να παράγει σωστό κείμενο. Αφορούσαν τον χρόνο, την ονοματοδοσία, τη διατήρηση της κατάστασης, τη γλώσσα των χρηστών, το περιβάλλον εκτέλεσης, δηλαδή το σύστημα γύρω από το μοντέλο. Η παρατήρηση αυτή είναι συνεπής με τη θέση που διατρέχει όλη την εργασία, ότι η αξιοπιστία ενός συστήματος βασισμένου σε Μεγάλα Γλωσσικά Μοντέλα κρίνεται σε μεγάλο βαθμό έξω από το ίδιο το μοντέλο.

4.12 Επίλογος

Από την τεχνολογική στοίβα μέχρι τη διεπαφή που τελικά βλέπει ο φοιτητής, αυτό το κεφάλαιο μετέφερε την αρχιτεκτονική περιγραφή του Κεφαλαίου 3 στο επίπεδο της υλοποίησης. Ξεκίνησε από την τεχνολογική στοίβα και την τεκμηρίωση κάθε επιλογής (4.2), συνέχισε με τον τρόπο αποθήκευσης των δεδομένων (4.3) και τον σχεδιασμό των prompts μέσω των οποίων επικοινωνεί το σύστημα με το μοντέλο (4.4). Ακολούθησε η υλοποίηση καθενός από τους τρεις πράκτορες (4.5 έως 4.7), έπειτα το επίπεδο που τους περιβάλλει και διατηρεί την κατάστασή τους (4.8), το μοντέλο προόδου που

εκτίθεται στον φοιτητή (4.9), και η διεπαφή μέσα από την οποία γίνονται όλα ορατά (4.10). Έκλεισε με τις δυσκολίες που προέκυψαν κατά την ανάπτυξη (4.11).

Ο συνδυασμός ντετερμινιστικής θεμελίωσης και κρίσης LLM, η αρχή που το Κεφάλαιο 3 έθεσε ως κεντρικό μοτίβο σχεδίασης (3.3), δεν έμεινε θεωρητική δήλωση. Στην υλοποίηση εμφανίστηκε ξανά και ξανά, σε κάθε επίπεδο: στη ρύθμιση της θερμοκρασίας, στη μορφή της απάντησης, στη διαδρομή των υποδείξεων, όπου ο Mentor αποφασίζει μόνος του πότε δεν χρειάζεται καν να καλέσει το μοντέλο, στην επικύρωση της απόφασης του Assessor, στο μοντέλο προόδου που φτάνει στον φοιτητή χωρίς καμία γλωσσική μεσολάβηση. Δεν ήταν μία απόφαση που πάρθηκε μία φορά στην αρχή του σχεδιασμού, ήταν ένας κανόνας που ξαναξετάστηκε ξεχωριστά σε κάθε σημείο όπου το σύστημα έπρεπε να αποφασίσει ποιος θα δώσει την τελική απάντηση, το μοντέλο ή ο ντετερμινιστικός κώδικας.

Μια δεύτερη αρχή διατρέχει παράλληλα την υλοποίηση, στη διαχείριση της κατάστασης: αποθηκεύεται πάντα το ακατέργαστο γεγονός, και το συγκεντρωτικό αποτέλεσμα υπολογίζεται όποτε χρειάζεται, όχι νωρίτερα. Η μόνη εξαίρεση είναι εκεί όπου το κόστος του υπολογισμού θα μεγάλωνε χωρίς όριο, οι μέσοι όροι που αφορούν όλη τη διάρκεια του προγράμματος.

Σε αριθμούς, το σύστημα πραγματοποιεί δώδεκα διαφορετικές κλήσεις μοντέλου, αναγνωρίζει είκοσι δύο κατηγορίες τεχνικών προβλημάτων, συγκροτεί κατάσταση είκοσι επτά πεδίων για την κύρια ροή, και παρουσιάζει το σύνολο μέσα από πέντε οθόνες.

Κεφάλαιο 5ο: Αξιολόγηση

5.1 Εισαγωγή

Τα δύο προηγούμενα κεφάλαια παρουσίασαν τον σχεδιασμό και την υλοποίηση του συστήματος. Το παρόν κεφάλαιο περνά από την περιγραφή στην πράξη, και εξετάζει με συγκεκριμένες μετρήσεις ποιο είναι το ουσιαστικό όφελος αυτής της αρχιτεκτονικής.

Το ερώτημα αυτό είναι κρίσιμο. Το σύστημα χρησιμοποιεί τρεις διακριτούς πράκτορες, διαδοχικές κλήσεις προς το μοντέλο και πολλαπλά επίπεδα ντετερμινιστικών ελέγχων, εκεί όπου μια απλούστερη προσέγγιση με μία μόνο κλήση θα μπορούσε επίσης να επιστρέψει μια απάντηση. Η αξία αυτής της πολυπλοκότητας δεν μπορεί να κριθεί μόνο από το αν η τελική ετυμηγορία για τον κώδικα είναι σωστή. Κρίνεται από το σύνολο της συμπεριφοράς γύρω από αυτή την ετυμηγορία, την παιδαγωγική απόφαση, την επικύρωση, τη θεμελίωση των ισχυρισμών. Η πειραματική αξιολόγηση πραγματοποιείται ακριβώς για να δείξει πού βρίσκεται η πραγματική διαφορά, και με ποιο υπολογιστικό κόστος.

Οι δύο ερευνητικοί στόχοι της ενότητας 1.2, η μελέτη της αποτελεσματικότητας των Agentic Workflows στην εκπαιδευτική διαδικασία και η μείωση των ψευδαισθήσεων των Μεγάλων Γλωσσικών Μοντέλων μέσω δομημένης γνώσης, διαρθρώνονται εδώ σε τέσσερα ερευνητικά ερωτήματα, καθένα μεταφρασμένο σε κάτι που μπορεί να μετρηθεί αντικειμενικά.

- E1: Διατηρεί η πολυπρακτορική αρχιτεκτονική υψηλή ακρίβεια στην αξιολόγηση της ορθότητας του κώδικα;
- E2: Μπορεί η αρχιτεκτονική αυτή να εξασφαλίσει συμπεριφορές και δομημένη πληροφορία που ένα σύστημα μίας κλήσης δεν προσφέρει με την ίδια αξιοπιστία;

- E3: Ποια είναι η συνεισφορά κάθε επιμέρους συνιστώσας της αρχιτεκτονικής στο τελικό αποτέλεσμα;
- E4: Περιορίζει η χρήση δομημένης πληροφορίας τις ψευδαισθήσεις του μοντέλου, και ειδικότερα τους ισχυρισμούς για τον κώδικα του φοιτητή που δεν στηρίζονται σε αυτόν;

Τα τρία πρώτα ερωτήματα υπηρετούν τον πρώτο στόχο, τη μελέτη της αποτελεσματικότητας των Agentic Workflows, μέσα από τη σύγκριση μιας πολυπρακτορικής αρχιτεκτονικής με μια απλούστερη εναλλακτική. Το τέταρτο υπηρετεί τον δεύτερο, τη μείωση των ψευδαισθήσεων μέσω δομημένης γνώσης, μεταφράζοντας το πρόβλημα της ενότητας 2.6 σε κάτι μετρήσιμο. Η μέτρηση αφορά ισχυρισμούς για δομές του ίδιου του κώδικα που επαληθεύονται αντικειμενικά, όχι το σύνολο των πιθανών ψευδαισθήσεων ενός γλωσσικού μοντέλου.

Κάθε ένα από τα τέσσερα ερωτήματα απαντιέται με μία από δύο μεθόδους, οι οποίες περιγράφονται αναλυτικά στην επόμενη ενότητα. Το κεφάλαιο ακολουθεί την εξής δομή: πρώτα οι μέθοδοι και τα εργαλεία τους (5.2) και τα δείγματα πάνω στα οποία εφαρμόζονται (5.3) μετά τα ευρήματα της πρώτης (5.4), και της δεύτερης μεθόδου (5.5) και το υπολογιστικό κόστος που τα συνοδεύει (5.6), και τέλος η σύνθεση, τα συμπεράσματα και οι περιορισμοί της μελέτης (5.7).

5.2 Μέθοδοι Αξιολόγησης

Η πρώτη μέθοδος αξιολόγησης που χρησιμοποιείται είναι η σύγκριση με ένα σύστημα αναφοράς (baseline). Πρόκειται για μια απλούστερη υλοποίηση που εκτελεί την ίδια εργασία με διαφορετικό τρόπο. Εξετάζει αν το προτεινόμενο σύστημα αποδίδει διαφορετικά από μια απλούστερη εναλλακτική. Η δεύτερη μέθοδος είναι τα πειράματα αφαίρεσης (ablation studies). Αντί να συγκρίνεται το σύστημα με κάτι εξωτερικό, απενεργοποιείται ένα συστατικό του κάθε φορά, με ρητό διακόπτη ενεργό μόνο για τη μέτρηση, και το ίδιο ακριβώς σύστημα εκτελείται ξανά στα ίδια δείγματα. Το πλεονέκτημα εδώ είναι ότι και οι δύο συνθήκες, με και χωρίς το συστατικό, μοιράζονται το ίδιο μοντέλο, τα ίδια prompts και τα ίδια δεδομένα. Η μόνη μεταβλητή είναι η παρουσία ή η απουσία του συγκεκριμένου συστατικού, οπότε όποια διαφορά προκύψει μπορεί να αποδοθεί σε αυτό και μόνο.

Οι δύο μέθοδοι είναι επομένως συμπληρωματικές και όχι εναλλακτικές. Η σύγκριση με το σύστημα αναφοράς απαντά τα E1 και E2, αν η αρχιτεκτονική στο σύνολό της αποδίδει διαφορετικά από μια απλούστερη λύση. Τα πειράματα αφαίρεσης απαντούν τα E3 και E4, ποιο συγκεκριμένο κομμάτι μέσα σε αυτή την αρχιτεκτονική ευθύνεται για τη διαφορά. Χρησιμοποιώντας τις δύο μαζί, η αξιολόγηση δεν στηρίζεται σε μία μόνο οπτική γωνία, ούτε αφήνει ανοιχτό το ερώτημα ποιο ακριβώς στοιχείο παράγει το αποτέλεσμα που θα καταγραφεί.

Κάθε ερώτημα μεταφράζεται σε συγκεκριμένη μετρική. Στο E1 απαντά η ορθότητα κρίσης, το ποσοστό των δειγμάτων στα οποία η ετυμηγορία για τον κώδικα συμφωνεί με την προκαθορισμένη ετικέτα. Καταγράφονται χωριστά τα δύο είδη σφάλματος, η αποδοχή λανθασμένου κώδικα και η απόρριψη σωστού, γιατί το παιδαγωγικό τους κόστος διαφέρει: το πρώτο αφήνει τον φοιτητή να προχωρήσει με λανθασμένη αντίληψη, το δεύτερο τον αποθαρρύνει άδικα.

Στο E2 απαντούν δύο μετρικές. Η αποκάλυψη λύσης είναι το ποσοστό απαντήσεων που περιέχουν έτοιμο κώδικα λύσης. Η ορθότητα της παιδαγωγικής απόφασης προόδου είναι το ποσοστό

περιπτώσεων όπου η απόφαση, advance, repeat ή support, συμφωνεί με τη δηλωμένη πολιτική δεδομένου του ιστορικού του φοιτητή. Αυτή η δεύτερη μετρική χρησιμοποιείται ξανά και για το E3, όχι πια σε σύγκριση με το σύστημα αναφοράς αλλά σε πειράματα αφαίρεσης μέσα στο ίδιο σύστημα, όπως θα φανεί στην ενότητα 5.5.

Το μήκος της ανατροφοδότησης καταγράφεται συμπληρωματικά στο E2, ως παρατηρούμενη διαφορά ύψους και όχι ως άμεση απόδειξη. Είναι χρήσιμο να καταγραφεί γιατί ο γνωστικός φόρτος έχει συνδεθεί εμπειρικά με τη μαθησιακή επίδοση σε εισαγωγικό προγραμματισμό Python, με μαθητές χαμηλού φόρτου να επιτυγχάνουν υψηλότερες επιδόσεις [41].

Στο E4 απαντά η θεμελίωση δομικών αναφορών, το ποσοστό απαντήσεων στις οποίες κάθε δομή του κώδικα που αναφέρει το κείμενο υπάρχει πράγματι σε αυτόν, μέσω σύγκρισης με το AST του κώδικα. Το εύρος της χρειάζεται ρητή οριοθέτηση: δεν μετρά τη συνολική ορθότητα της απάντησης ούτε το σύνολο των πιθανών ψευδαισθήσεων, μόνο μία συγκεκριμένη, αντικειμενικά επαληθεύσιμη κατηγορία, τις αναφορές σε δομές όπως βρόχους, δομές επιλογής, λίστες και ορισμούς συναρτήσεων.

Η θεμελίωση μετρήθηκε από άνθρωπο και όχι από δεύτερο μοντέλο, επιλογή που χρειάζεται δικαιολόγηση. Η αξιολόγηση εξόδων ενός Μεγάλου Γλωσσικού Μοντέλου από ένα δεύτερο μοντέλο, γνωστή ως LLM-as-judge, είναι διαδεδομένη και μειώνει το κόστος, η σχετική έρευνα όμως δείχνει ότι οι αποκλίσεις από ανθρώπινους κριτές μεγαλώνουν στις περιπτώσεις που απαιτούν βαθύτερο συλλογισμό, με τους ανθρώπους να παρουσιάζουν εκεί μεγαλύτερη συνέπεια, και καταλήγει στην ανάγκη ανθρώπινης εποπτείας σε σύνθετες εκπαιδευτικές εργασίες [47]. Η κρίση για το αν ένας ισχυρισμός θεμελιώνεται στον κώδικα δεν είναι απλό ζήτημα γεγονότος, απαιτεί σύγκριση του νοήματος μιας πρότασης με τη δομή ενός δέντρου AST, και ανήκει επομένως ακριβώς στην κατηγορία όπου η ίδια η βιβλιογραφία γύρω από το LLM-as-judge προτείνει ανθρώπινη εποπτεία [47]. Η διαδικασία είχε δύο στάδια: πρώτα ένα εργαλείο συνέκρινε αυτόματα τις δομές που αναφέρει κάθε πρόταση με τις δομές που περιέχει το AST, εξαιρώντας αναφορές σε επόμενες ενότητες και προτροπές για το τι πρέπει να γραφτεί, όπου μια δομή αναφέρεται ακριβώς επειδή λείπει. Έπειτα κάθε υποψήφιο εύρημα επιβεβαιώθηκε με ανάγνωση, χωρίς γνώση της συνθήκης που είχε παραγάγει κάθε έξοδο.

Το κόστος, τέλος, δεν αντιστοιχεί σε κανένα από τα τέσσερα ερωτήματα, καταγράφεται όμως γιατί κάθε όφελος που θα παρουσιαστεί στη συνέχεια πρέπει τελικά να σταθμιστεί απέναντί του, και περιλαμβάνει tokens ανά υποβολή, χρόνο απόκρισης και πλήθος κλήσεων μοντέλου.

Πέρα από το τι μετριέται, έχει σημασία και το πώς μετρήθηκε. Τα Μεγάλα Γλωσσικά Μοντέλα παρουσιάζουν στοχαστικότητα, δηλαδή η ίδια είσοδος μπορεί να δώσει ελαφρώς διαφορετική έξοδο σε δύο εκτελέσεις, οπότε οι συνθήκες κάθε μέτρησης καταγράφονται ώστε η αξιολόγηση να μπορεί να επαναληφθεί. Όλες οι μετρήσεις έγιναν με το μοντέλο openai/gpt-oss-120b μέσω του παρόχου Groq, με τη θερμοκρασία να ακολουθεί την ίδια διάκριση με την ενότητα 4.2, μηδέν για τις κλήσεις κλειστού συνόλου και 0.1 για όσες συνθέτουν κρίση ή κείμενο. Πριν από τις μετρήσεις, το μοντέλο ελέγχθηκε ξεχωριστά ως προς τη συμμόρφωση με την απαιτούμενη μορφή εξόδου, την πειθαρχία στον κλειστό κατάλογο κατηγοριών και την παραγωγή ελληνικού κειμένου. Και οι δύο προσεγγίσεις εκτελέστηκαν στα ίδια δείγματα, με την ίδια σειρά, χωρίς τυχαιοποίηση, με το ίδιο ανώτατο χρονικό όριο ανά κλήση, και χωρίς ανώτατο όριο μήκους απάντησης σε καμία από τις δύο, ώστε η μέτρηση του μήκους να αποτυπώνει τη φυσική τους συμπεριφορά. Κάθε εκτέλεση καταγράφει την ετυμολογία, την απόφαση, το πλήρες κείμενο της ανατροφοδότησης, την κατηγορία σφάλματος, τα tokens, τον χρόνο και το πλήθος των κλήσεων, και τα αρχεία αποτελεσμάτων διατηρούνται, ώστε κάθε αριθμός του κεφαλαίου να μπορεί να ανιχνευθεί πίσω στη συγκεκριμένη γραμμή που τον παρήγαγε.

Οι περισσότερες μετρήσεις προέρχονται από μία εκτέλεση ανά δείγμα. Έξι δείγματα εκτελέστηκαν επιπλέον τρεις φορές το καθένα, ώστε να εκτιμηθεί η επίδραση της στοχαστικότητας. Και οι δύο προσεγγίσεις έδωσαν ταυτόσημη ετυμηγορία σε κάθε επανάληψη, και το πολυπρακτορικό σύστημα διατήρησε επιπλέον την ίδια κατηγορία σφάλματος, γεγονός που δείχνει ότι στη συγκεκριμένη ρύθμιση θερμοκρασίας η μεταβλητότητα παραμένει περιορισμένη.

5.3 Δείγματα

Κάθε ερευνητικό ερώτημα χρειάζεται διαφορετικό υλικό για να απαντηθεί. Η ορθότητα κρίσης δεν ελέγχεται με τα ίδια δείγματα που ελέγχουν την απόφαση προόδου, ούτε η θεμελίωση των ισχυρισμών με δείγματα φτιαγμένα για να δοκιμάσουν ανθεκτικότητα. Για τον λόγο αυτό κατασκευάστηκαν τέσσερα σύνολα, καθένα σχεδιασμένο για συγκεκριμένο σκοπό.

Ο πυρήνας αποτελείται από 24 δείγματα και καλύπτει και τις πέντε ενότητες του προγράμματος και τα δύο επίπεδα δυσκολίας. Περιέχει σωστές λύσεις, σωστές λύσεις γραμμένες με ασυνήθιστο ύφος, συντακτικά σφάλματα, σφάλματα που θα εμφανίζονταν μόνο κατά την εκτέλεση, λάθος τιμές, αντεστραμμένη λογική και δομές που λείπουν. Η αναλογία σωστών προς λανθασμένων είναι ισορροπημένη, ώστε ένα σύστημα με σταθερή απάντηση να μην μπορεί να πετύχει υψηλό σκορ από τύχη.

Το σύνολο ανθεκτικότητας αποτελείται από 8 δείγματα και εξετάζει τι συμβαίνει όταν η υποβολή δεν είναι κανονικός κώδικας, περιπτώσεις που εμφανίζονται τακτικά σε πραγματική χρήση: κενή υποβολή, μόνο κενοί χαρακτήρες, τυχαίο πάτημα πλήκτρων, μόνο σχόλιο, κώδικας γραμμένος σε άλλη γλώσσα προγραμματισμού, πρόταση σε φυσική γλώσσα αντί για κώδικα, παρένθεση που μένει ανοιχτή, και μια υποβολή που περιέχει μέσα της οδηγία η οποία επιχειρεί να παρακάμψει την αξιολόγηση. Το τελευταίο δείγμα εξετάζει συγκεκριμένα αν το σύστημα αντιμετωπίζει τον κώδικα ως δεδομένο προς ανάλυση και όχι ως οδηγία προς εκτέλεση.

Το σύνολο επιφανειακών μοτίβων αποτελείται από 6 δείγματα και είναι το πιο στοχευμένο από τα τέσσερα. Κάθε δείγμα είναι φτιαγμένο ώστε η επιφανειακή μορφή του κώδικα να οδηγεί σε λάθος συμπέρασμα, αν η κρίση στηριχθεί σε αυτήν και όχι στη δομή. Τρία δείγματα φαίνονται σωστά ενώ έχουν πρόβλημα, όπως μια λέξη-κλειδί κρυμμένη μέσα σε συμβολοσειρά ή μια εντολή εκτύπωσης που βρίσκεται μέσα σε συνάρτηση η οποία δεν καλείται ποτέ. Τρία φαίνονται λανθασμένα ενώ είναι σωστά, όπως το δείγμα S6, μια εμφωλευμένη δομή if που είναι λειτουργικά ισοδύναμη με elif. Η ισορροπία ανάμεσα στις δύο κατηγορίες είναι σκόπιμη: ένα σύνολο μόνο με την πρώτη θα ανταμείβε την υπερβολική αυστηρότητα, ενώ ένα σύνολο μόνο με τη δεύτερη θα ανταμείβε τη χαλαρότητα.

Το σύνολο απόφασης προόδου αποτελείται από 4 δείγματα και έχει μια ιδιαιτερότητα που το κάνει ασυνήθιστα καθαρό ως πείραμα. Και στα τέσσερα δείγματα ο κώδικας είναι ο ίδιος και είναι σωστός. Εκείνο που αλλάζει είναι αποκλειστικά το ιστορικό του φοιτητή (Πίνακας 5.1).

Πίνακας 5.1: Δείγματα του συνόλου απόφασης προόδου (D1-D4)

Δείγμα	Προσπάθειες	Υποδείξεις	Χρόνος	Αναμενόμενη απόφαση
D1	1	0	30s	advance
D2	3	1	120s	advance
D3	4	2	150s	repeat
D4	5	4	240s	repeat

Η αναμενόμενη απόφαση προκύπτει από τη ρητή πολιτική του Assessor (3.6): η προαγωγή υποβαθμίζεται σε επανάληψη όταν χρειάστηκαν τέσσερις ή περισσότερες προσπάθειες ή τρεις ή περισσότερες υποδείξεις. Η πολιτική αυτή δεν είναι αυθαίρετη, στηρίζεται στην αρχή της mastery learning που αναλύθηκε στην ενότητα 3.6: μια τελικά σωστή απάντηση δεν αποδεικνύει από μόνη της ότι ο φοιτητής έχει κατακτήσει την ύλη, αν χρειάστηκε δυσανάλογη προσπάθεια για να φτάσει εκεί. Το D2 τοποθετήθηκε σκόπιμα λίγο κάτω από αυτά τα κατώφλια και το D3 ακριβώς πάνω τους, ώστε το σύνολο να ξεετάζει το ίδιο το όριο και όχι μόνο τις προφανείς περιπτώσεις.

Ο σχεδιασμός αυτός έχει μια χρήσιμη ιδιότητα, η οποία τον καθιστά τεστ και όχι απλή επίδειξη. Ένα σύστημα που αγνοεί το ιστορικό θα απαντήσει advance και στις τέσσερις περιπτώσεις, αφού ο κώδικας είναι πάντα σωστός και τίποτα άλλο δεν διαφοροποιεί τα δείγματα. Η ίδια η μορφή του συνόλου αποκαλύπτει επομένως αν το ιστορικό χρησιμοποιείται πράγματι κατά την απόφαση ή απλώς υπάρχει στην κατάσταση του συστήματος χωρίς να επηρεάζει τίποτα.

Σε όλα τα σύνολα, η ετικέτα ορθότητας κάθε δείγματος καθορίστηκε πριν από οποιαδήποτε εκτέλεση, από την εκφώνηση και τα κριτήρια, χωρίς γνώση του τι θα απαντούσε οποιοδήποτε από τα δύο συστήματα.

Ο Πίνακας 5.2 δείχνει πού χρησιμοποιείται καθένα από τα τέσσερα σύνολα στις επόμενες ενότητες.

Πίνακας 5.2: Χρήση κάθε συνόλου δειγμάτων ανά ενότητα αξιολόγησης

Σύνολο	Χρησιμοποιείται στην ενότητα
Πυρήνας	5.4, 5.5
Ανθεκτικότητα	5.4
Επιφανειακά μοτίβα	5.4, 5.5
Απόφαση προόδου (D1-D4)	5.4, 5.5

5.4 Πρώτη Μέθοδος: Σύγκριση με το Σύστημα Αναφοράς

Αντικείμενο της σύγκρισης είναι, από τη μία πλευρά, η τελική έκδοση του συστήματος όπως περιγράφηκε στα Κεφάλαια 3 και 4 στο σύνολό της, με όλους τους ντετερμινιστικούς ελέγχους και όλα τα επίπεδα επικύρωσης ενεργά, και από την άλλη ένα σύστημα αναφοράς (baseline) φτιαγμένο ειδικά για τη σύγκριση, ένα μεμονωμένο πρακτορικό σύστημα (single-agent) που εκτελεί ολόκληρη την εργασία μέσω μίας μόνο κλήσης στο μοντέλο. Λαμβάνει την εκφώνηση, τα κριτήρια επιτυχίας και τον κώδικα του φοιτητή, και επιστρέφει την κρίση ορθότητας μαζί με την ανατροφοδότηση.

Η επιλογή αυτή αντιστοιχεί ακριβώς στη διάκριση της ενότητας 2.3. Ένα μοντέλο μπορεί να λάβει οδηγία να υιοθετήσει διαδοχικά διαφορετικούς ρόλους μέσα στο ίδιο prompt, με αποτέλεσμα που μοιάζει εξωτερικά με συνεργασία, χωρίς όμως οι ρόλοι αυτοί να διαθέτουν δική τους κατάσταση ή να ενεργοποιούνται από ρητή δρομολόγηση. Σε ένα πολυπρακτορικό σύστημα, αντίθετα, κάθε πράκτορας αποτελεί ξεχωριστό υπολογιστικό βήμα με δικό του prompt και δικό του αποτέλεσμα, το οποίο καταγράφεται πριν περάσει στον επόμενο. Το σύστημα αναφοράς υλοποιεί την πρώτη περίπτωση, και η σύγκριση θέτει τις δύο μορφές οργάνωσης αντιμέτωπες.

Ότι δεν αφορά την ίδια την αρχιτεκτονική διατηρήθηκε σταθερό: ίδιο μοντέλο, ίδια θερμοκρασία, ίδια εκφώνηση αυτολεξεί, ίδια κριτήρια αυτολεξεί, ίδιος κώδικας, οι ίδιοι παιδαγωγικοί κανόνες μέσα στο prompt, μαζί με τη ρητή απαγόρευση αποκάλυψης της λύσης. Όποια διαφορά προκύψει δεν μπορεί επομένως να αποδοθεί σε άνιση πληροφόρηση. Τέσσερις μηχανισμοί απουσιάζουν από το σύστημα

αναφοράς, και καθένας αντιστοιχεί σε συγκεκριμένο κομμάτι της αρχιτεκτονικής των Κεφαλαίων 3 και 4. Δεν διαθέτει στατική ανάλυση, οπότε ό,τι γνωρίζει για τον κώδικα προκύπτει διαβάζοντάς τον ως κείμενο. Δεν επικυρώνει την απάντησή του, η έξοδος αναλύεται μόνο ως προς τη μορφή της. Δεν παράγει σταθερή κατηγορία σφάλματος, η περιγραφή του προβλήματος διατυπώνεται κάθε φορά ελεύθερα. Και η ίδια κλήση κρίνει τον κώδικα και γράφει την απάντηση, έχοντας τον πλήρη κώδικα μπροστά της, ενώ στο προτεινόμενο σύστημα ο Mentor που συντάσσει την απάντηση λαμβάνει την κρίση χωρίς τον κώδικα, όπως περιγράφηκε στην ενότητα 3.4.2. Οι απουσίες αυτές δεν αποτελούν αδυναμία της υλοποίησης, είναι το ίδιο το αντικείμενο της σύγκρισης, αφού ένα σύστημα μίας κλήσης δεν έχει ενδιάμεσο σημείο στο οποίο θα μπορούσαν να τοποθετηθούν.

Το E1 αφορούσε τη βασική ακρίβεια, και απαντάται πρώτο επειδή αποτελεί προϋπόθεση για όσα ακολουθούν (Πίνακας 5.3).

Πίνακας 5.3: Ορθότητα κρίσης ανά σύνολο δειγμάτων

Σύνολο	n	Πολυπρακτορικό	Αναφοράς
Πυρήνας	24	24/24	24/24
Ανθεκτικότητα	8	8/8	8/8
Επιφανειακά μοτίβα	6	5/6	5/6
Σύνολο	38	37/38	37/38

Οι δύο προσεγγίσεις παρουσίασαν ταυτόσημη επίδοση σε κάθε ένα από τα τρία σύνολα. Στα 38 δείγματα, καθεμία έκρινε σωστά τα 37, με το ίδιο δείγμα να αστοχεί και στις δύο, το S6 του συνόλου επιφανειακών μοτίβων. Η ταύτιση επεκτείνεται και στο είδος του σφάλματος. Καμία από τις δύο δεν δέχτηκε λανθασμένο κώδικα, και καθεμία απέρριψε ένα σωστό. Το ίδιο ισχύει και για τη σταθερότητα σε τρεις επαναλήψεις: και οι δύο προσεγγίσεις έδωσαν την ίδια ετυμηγορία και την ίδια κατηγορία σφάλματος και στα έξι δείγματα του πειράματος επανάληψης.

Το αποτέλεσμα χρειάζεται προσεκτική ερμηνεία. Η κρίση ορθότητας είναι εργασία στην οποία ένα σύγχρονο Μεγάλο Γλωσσικό Μοντέλο αποδίδει ήδη πολύ καλά, εφόσον διαθέτει τον κώδικα και τα κριτήρια, και οι δύο προσεγγίσεις πλησιάζουν το ανώτατο δυνατό σκορ των συνόλων. Η εργασία δεν ισχυρίζεται πλεονέκτημα σε αυτή τη διάσταση, η αρχιτεκτονική δεν σχεδιάστηκε για να κάνει το μοντέλο καλύτερο σε κάτι που ήδη κάνει καλά. Αυτό που δείχνει η ταύτιση είναι ότι οι ιδιότητες που παρουσιάζονται στη συνέχεια της ενότητας προστίθενται χωρίς παρατηρούμενη απώλεια στη βασική ικανότητα. Το σύστημα δεν αντάλλαξε ακρίβεια για λειτουργικότητα. Η απουσία παρατηρούμενης υποβάθμισης δεν αποτελεί βεβαίως απόδειξη ισοδυναμίας, αποτελεί όμως την προϋπόθεση που έπρεπε να ικανοποιηθεί ώστε να έχει νόημα η συνέχεια.

Η ταύτιση της ετυμηγορίας αφορά το αν ο κώδικας κρίθηκε σωστός. Το κείμενο που φτάνει στον φοιτητή όμως δεν είναι η ετυμηγορία, είναι μια παιδαγωγική απάντηση, και εκεί οι δύο προσεγγίσεις διαφέρουν συστηματικά. Ως προς τον Σωκρατικό περιορισμό, καμία από τις δύο δεν αποκάλυψε τη λύση σε κανένα από τα 38 δείγματα, ούτε καν στις οκτώ εχθρικές υποβολές του συνόλου ανθεκτικότητας. Η ρητή απαγόρευση μέσα στο prompt τηρήθηκε και από τις δύο, και η μετρική αυτή δεν διαχωρίζει τις προσεγγίσεις στα συγκεκριμένα σύνολα. Η διαφορά εντοπίζεται στην έκταση της απάντησης, και είναι σταθερή σε κάθε σύνολο χωρίς εξαίρεση (Πίνακας 5.4).

Πίνακας 5.4: Μέσο μήκος ανατροφοδότησης ανά σύνολο δειγμάτων

Σύνολο	Πολυπρακτορικό	Αναφοράς
Πυρήνας	36.8 λέξεις	53,0 λέξεις

Ανθεκτικότητα	29,1 λέξεις	77,6 λέξεις
Επιφανειακά μοτίβα	31,7 λέξεις	58,2 λέξεις

Η ανατροφοδότηση του πολυπρακτορικού συστήματος είναι από μιάμιση έως σχεδόν τρεις φορές συντομότερη, και αξίζει να δοθεί προσοχή στο πού μεγαλώνει περισσότερο η διαφορά: στο σύνολο ανθεκτικότητας, ακριβώς εκεί όπου η υποβολή είναι πιο ασαφής και ένα σύστημα έχει το μεγαλύτερο περιθώριο να επεκταθεί. Η παρατηρούμενη διαφορά είναι συμβατή με την υπόθεση ότι την καθορίζει η κατανομή των ρόλων και όχι κάποια ρύθμιση μήκους, καθώς σε κανένα από τα δύο συστήματα δεν ορίζεται όριο λέξεων. Στο σύστημα αναφοράς η ίδια κλήση καλείται ταυτόχρονα να κρίνει τον κώδικα, να εντοπίσει το πρόβλημα, να το εξηγήσει και να καθοδηγήσει τον φοιτητή, έχοντας τον πλήρη κώδικα μπροστά της. Μπροστά σε μια ασαφή υποβολή, όπου δεν είναι σαφές τι ακριβώς πήγε στραβά, η φυσική της τάση είναι να καλύψει όσα περισσότερα ενδεχόμενα μπορεί. Στο πολυπρακτορικό σύστημα η κρίση έχει ήδη ολοκληρωθεί σε προηγούμενο βήμα, και ο Mentor που συντάσσει την απάντηση παραλαμβάνει ένα συγκεκριμένο εύρημα και μια συγκεκριμένη απόφαση, με στενά ορισμένη δουλειά, να διατυπώσει παιδαγωγικά κάτι που έχει ήδη αποφασιστεί, χωρίς να χρειάζεται να καλύψει ενδεχόμενα που έχουν ήδη επιλεγεί. Η μέτρηση αφορά την έκταση και μόνο. Η παρούσα αξιολόγηση δεν εξέτασε αν η συντομότερη ανατροφοδότηση γίνεται ευκολότερα κατανοητή ή μειώνει τον γνωστικό φόρτο, κάτι τέτοιο απαιτεί μέτρηση με πραγματικούς φοιτητές.

Η παιδαγωγική απόφαση προόδου αφορά κάτι διαφορετικό από την ορθότητα κρίσης και το μήκος της ανατροφοδότησης, και εδώ η σύγκριση παίρνει διαφορετική μορφή, γιατί το σύστημα αναφοράς δεν παράγει καθόλου τέτοια απόφαση. Η μία κλήση του επιστρέφει κρίση ορθότητας και ανατροφοδότηση, τίποτα παραπάνω, αφού δεν του ζητήθηκε ποτέ να την παραγάγει. Το να καταγραφεί αυτό ως μηδέν σωστές αποφάσεις θα ήταν παραπλανητικό. Το πολυπρακτορικό σύστημα, αντίθετα, εκτελέστηκε στο σύνολο απόφασης προόδου της ενότητας 5.3 και παρήγαγε τη σωστή απόφαση και στις τέσσερις περιπτώσεις: προήγαγε τον φοιτητή που τα κατάφερε με μία προσπάθεια, προήγαγε και εκείνον που χρειάστηκε τρεις προσπάθειες και μία υπόδειξη, αλλά κράτησε στην ίδια ενότητα εκείνον που χρειάστηκε τέσσερις προσπάθειες και δύο υποδείξεις, όπως και εκείνον που χρειάστηκε πέντε προσπάθειες και τέσσερις υποδείξεις. Αυτό απαντά άμεσα στο Ε2: υπάρχει τουλάχιστον μία λειτουργία, η αναγνώριση ότι ο ίδιος σωστός κώδικας σημαίνει διαφορετικά πράγματα ανάλογα με το πόσο δύσκολα έφτασε εκεί ο φοιτητής, την οποία το σύστημα αναφοράς δεν προσφέρει καν, όχι επειδή αποδεδειγμένα αδυνατεί, αλλά επειδή η μορφή του δεν την προβλέπει. Το αν θα μπορούσε να την προσφέρει αν του ζητηθεί ρητά είναι διαφορετικό ερώτημα, και εξετάζεται δίκαια στην ενότητα 5.5, όπου το ίδιο σύστημα αναφοράς επεκτείνεται ειδικά για αυτόν τον σκοπό.

5.5 Δεύτερη Μέθοδος: Πειράματα Αφαίρεσης

Οι μετρήσεις της ενότητας αυτής δεν εμπλέκουν καθόλου το σύστημα αναφοράς. Συγκρίνουν το προτεινόμενο σύστημα με τον εαυτό του, με ένα συστατικό απενεργοποιημένο κάθε φορά. Το πλεονέκτημα της μεθόδου, όπως εξηγήθηκε στην ενότητα 5.2, είναι ότι εδώ δεν μένει κανένα ανοιχτό ερώτημα για ενδεχόμενο προβάδισμα της μιας συνθήκης απέναντι στην άλλη, αφού πρόκειται κυριολεκτικά για το ίδιο σύστημα. Τα ευρήματα που ακολουθούν στηρίζονται επομένως σε ισχυρότερο πειραματικό σχεδιασμό από εκείνα της προηγούμενης ενότητας.

Εξετάστηκαν τρία συστατικά. Το ιστορικό του φοιτητή, το οποίο τροφοδοτεί την απόφαση προόδου, η επικύρωση της απάντησης του μοντέλου, όπως περιγράφηκε στην ενότητα 3.6, και η δομημένη πληροφορία προς τον Mentor, όπως περιγράφηκε στην ενότητα 3.4.2.

Στο σύνολο απόφασης προόδου εξετάστηκαν δύο παραλλαγές. Στην πρώτη ο Assessor εκτελείται κανονικά αλλά χωρίς το ιστορικό του φοιτητή, δηλαδή χωρίς τις προσπάθειες, τις υποδείξεις και τον χρόνο. Στη δεύτερη καταγράφεται η ωμή απόφαση του μοντέλου πριν περάσει από τους ελέγχους επικύρωσης, κάτι που δεν απαιτεί καν ξεχωριστή εκτέλεση καθώς η τιμή αυτή καταγράφεται ούτως ή άλλως. Η πρώτη παραλλαγή αντιστοιχεί επομένως σε πραγματική επανεκτέλεση του συστήματος με απενεργοποιημένο συστατικό, ενώ η δεύτερη σε παρατήρηση μιας τιμής που υπάρχει ήδη μέσα στην κανονική ροή, πριν το στάδιο ελέγχου (Πίνακας 5.5).

Πίνακας 5.5: Αποτελέσματα πειράματος αφαίρεσης στην απόφαση προόδου

Δείγμα	Ιστορικό	Αναμενόμενη	Πλήρες σύστημα	Χωρίς ιστορικό	Χωρίς επικύρωση
D1	1 προσπάθεια, 0 υποδείξεις	advance	advance	advance	advance
D2	3 προσπάθειες, 1 υπόδειξη	advance	advance	advance	advance
D3	4 προσπάθειες, 2 υποδείξεις	repeat	repeat	advance	advance
D4	5 προσπάθειες, 4 υποδείξεις	repeat	repeat	advance	support
Σύνολο			4/4	2/4	2/4

Στο συγκεκριμένο σύνολο, η αφαίρεση καθενός από τα δύο συστατικά μείωσε την ορθότητα της απόφασης από τέσσερις σωστές σε δύο. Το εύρημα αποτελεί ένδειξη της συνεισφοράς των δύο μηχανισμών και δεν επιτρέπει γενίκευση σε μεγαλύτερα σύνολα, αλλά η μορφή των αστοχιών είναι διαφωτιστική και αξίζει χωριστή ανάγνωση, γιατί δεν πρόκειται για τον ίδιο τύπο σφάλματος.

Χωρίς το ιστορικό, το σύστημα απαντά advance και στις τέσσερις περιπτώσεις. Η συμπεριφορά είναι απολύτως λογική από τη σκοπιά του συστήματος: ο κώδικας είναι πάντα σωστός, και χωρίς πληροφορία για τη διαδρομή του φοιτητή τα τέσσερα δείγματα είναι μεταξύ τους πανομοιότυπα, δεν υπάρχει κανένα στοιχείο πάνω στο οποίο θα μπορούσε να στηριχθεί μια διαφορετική απόφαση. Το εύρημα αυτό επιβεβαιώνει κάτι που διαφορετικά θα έμενε ισχυρισμός. Το ιστορικό του φοιτητή λειτουργεί ως ουσιαστική είσοδος στην απόφαση και όχι ως διακοσμητικό συμφραζόμενο που απλώς υπάρχει στο prompt.

Χωρίς την επικύρωση, οι δύο αστοχίες είναι διαφορετικού είδους, και η διάκριση δείχνει ότι δεν πρόκειται για έναν μηχανισμό αλλά για δύο. Στο D3 το μοντέλο πρότεινε advance για φοιτητή με τέσσερις προσπάθειες και δύο υποδείξεις, χωρίς αντιφατική απάντηση, απλώς αγνόησε το κατώφλι. Στο πλήρες σύστημα ο ντετερμινιστικός έλεγχος κατωφλίου υποβάθμισε την απόφαση σε repeat, αφήνοντας την ετυμηγορία για τον κώδικα αμετάβλητη, καθώς η υποβάθμιση αφορά μόνο την παιδαγωγική κίνηση. Στο D4 το μοντέλο επέστρεψε ταυτόχρονα «σωστός κώδικας» και support, συνδυασμό αντιφατικό αφού η ενεργή υποστήριξη προϋποθέτει ότι ο φοιτητής δεν έχει ακόμα λύσει την άσκηση. Εκεί ο έλεγχος συνέπειας απέρριψε ολόκληρη την απάντηση και ενεργοποιήθηκε το ντετερμινιστικό εναλλακτικό μονοπάτι, το οποίο παρήγαγε repeat από τα ίδια αντικειμενικά ευρήματα. Στο συγκεκριμένο σύνολο, επομένως, η επικύρωση δεν παρέμεινε θεωρητική δικλείδα ασφαλείας. Ενεργοποιήθηκε σε δύο από τις τέσσερις περιπτώσεις και διόρθωσε δύο αστοχίες διαφορετικού τύπου, μία παράβαση κατωφλίου και μία εσωτερική αντίφαση.

Το δεύτερο πείραμα αφορά τη θεμελίωση, και απαντά στο E4, δηλαδή στον δεύτερο στόχο της εργασίας, τη μείωση των ψευδαισθήσεων μέσω δομημένης γνώσης. Το σημείο του συστήματος όπου

κρίνεται αυτό είναι ο Mentor, λόγω της ιδιαίτερης θέσης του μέσα στη ροή: είναι ο μόνος πράκτορας του οποίου η έξοδος φτάνει αυτούσια στον φοιτητή, και ταυτόχρονα ο μόνος που εργάζεται χωρίς να έχει δει τον κώδικα, καλείται όμως να μιλήσει για αυτόν. Η αρχιτεκτονική απάντηση σε αυτή την ένταση, όπως περιγράφηκε στην ενότητα 3.4.2, είναι να λαμβάνει από τη στατική ανάλυση μια ρητή λίστα των δομών που υπάρχουν πραγματικά στον κώδικα και εκείνων που απουσιάζουν. Για να μετρηθεί τι κάνει αυτή η λίστα, το σύστημα εκτελέστηκε δύο φορές, με και χωρίς αυτήν, στα ίδια δείγματα και με το ίδιο μοντέλο. Η ορθότητα κρίσης παρέμεινε αμετάβλητη και στα δύο πειράματα, όπως στον πίνακα της ενότητας 5.4. Οι αριθμοί που ακολουθούν μετρούν αποκλειστικά τη θεμελίωση των εξόδων (Πίνακας 5.6).

Πίνακας 5.6: Θεμελίωση εξόδων με και χωρίς δομημένη πληροφορία

Πλήρως θεμελιωμένες εξοδοί ανά σύνολο	Με θεμελίωση	Χωρίς θεμελίωση
Πυρήνας (24 εξοδοί)	24/24	21/24
Επιφανειακά μοτίβα (6 εξοδοί)	6/6	5/6
Σύνολο (30 εξοδοί)	30/30	26/30

Με τη δομημένη πληροφορία, και οι τριάντα εξοδοί θεμελιώνονται πλήρως στον κώδικα. Χωρίς αυτήν, τέσσερις περιέχουν ισχυρισμό που ο ίδιος ο κώδικας διαψεύδει, και όλες ανήκουν στον ίδιο τύπο, το κείμενο επικαλείται δομή που απουσιάζει εντελώς από το πρόγραμμα: «δείχνει ότι έχεις κατακτήσει τις συνθήκες και τους βρόχους» σε κώδικα χωρίς κανέναν βρόχο, «κατάφερες να εφαρμόσεις σωστά τις συνθήκες και τους βρόχους» σε κώδικα χωρίς κανέναν βρόχο, «Πρόσεξε πώς διαχειρίζεσαι τις μεταβλητές και τις δομές ελέγχου» σε κώδικα χωρίς καμία δομή ελέγχου, «Πρόσεξε τη δομή των βρόχων» σε κώδικα που περιέχει μόνο ορισμό συνάρτησης. Δύο από τους τέσσερις αξίζουν ξεχωριστή προσοχή, γιατί εμφανίζονται σε μηνύματα επιβράβευσης. Ο φοιτητής είχε απαντήσει σωστά, η κρίση ήταν σωστή, και η ανατροφοδότηση ενθαρρυντική. Το πρόβλημα βρίσκεται αλλού: όταν το μοντέλο καλείται να μιλήσει για κάτι που δεν βλέπει, συμπληρώνει το κενό με λεπτομέρειες που ακούγονται εύλογες για το συγκεκριμένο μάθημα. Ένας αρχάριος που διαβάσει ότι χειρίστηκε σωστά βρόχους τους οποίους δεν έγραψε ποτέ λαμβάνει επιβεβαίωση για κάτι που δεν έχει ακόμα μάθει, και αυτό είναι παιδαγωγικά πιο επιζήμιο από μια λανθασμένη υπόδειξη, γιατί δεν υπάρχει τίποτα στην οθόνη που να τον κάνει να το αμφισβητήσει.

Οι μη τεκμηριωμένες δομικές αναφορές δεν εμφανίστηκαν παντού μέσα στο σύστημα. Εμφανίστηκαν ακριβώς στο σημείο όπου έλειπε η θεμελίωση, και εξαφανίστηκαν μόλις προστέθηκε. Το ίδιο μοντέλο, μέσα στο ίδιο σύστημα, με την ίδια ακριβώς εργασία, συμπεριφέρθηκε διαφορετικά ανάλογα με το αν διέθετε δεδομένα ή όχι. Η θέση ότι ένα Μεγάλο Γλωσσικό Μοντέλο χρειάζεται εξωτερική θεμελίωση για να είναι αξιόπιστο [3], [19], [21] επιβεβαιώνεται εδώ όχι ως γενική αρχή αλλά ως μετρημένη διαφορά μέσα στην ίδια υλοποίηση, εντός του εύρους που ορίστηκε στην ενότητα 5.2.

Τα δύο ευρήματα αυτής της ενότητας γεννούν μια εύλογη ένσταση, η οποία εξετάστηκε πειραματικά και όχι μόνο θεωρητικά. Αν οι ιδιότητες που κερδίζονται είναι η απόφαση προόδου και η τήρηση μιας πολιτικής, γιατί να μην ενσωματωθούν απλώς σε ένα πλουσιότερο prompt μίας κλήσης, γλιτώνοντας τον επιπλέον χρόνο και κόστος της ενότητας 5.6; Εδώ χρησιμοποιείται διαφορετικό σύστημα αναφοράς από αυτό της ενότητας 5.4, και η διαφορά χρειάζεται διευκρίνιση. Το σύστημα αναφοράς της 5.4 σχεδιάστηκε ώστε να εκτελεί τη λειτουργία που είναι κοινή στις δύο προσεγγίσεις, την κρίση του κώδικα και την παραγωγή ανατροφοδότησης, και για τον λόγο αυτό δεν του ζητείται καν απόφαση προόδου. Το πείραμα που ακολουθεί χρησιμοποιεί μια διευρυμένη εκδοχή του ίδιου συστήματος αναφοράς, στην οποία ζητείται ρητά και η απόφαση. Τα δύο δεν πρέπει να συγχέονται,

απαντούν σε διαφορετικό ερώτημα. Το πρώτο εξετάζει την κοινή λειτουργία, το δεύτερο εξετάζει αν η πρόσθετη συμπεριφορά της αρχιτεκτονικής μπορεί να μεταφερθεί σε μία κλήση, δοσμένη η ίδια πληροφορία.

Το πείραμα έγινε σε δύο βήματα, καθένα πλουσιότερο από το προηγούμενο. Στο πρώτο, στη διευρυμένη εκδοχή ζητείται η απόφαση προόδου, με αυτολεξεί την ίδια περιγραφή της πολιτικής που έχει και ο Assessor. Στο δεύτερο προστέθηκαν και τα ίδια αριθμητικά κατώφλια, μαζί με ρητή απαγόρευση των δύο αντιφατικών συνδυασμών που περιγράφηκαν παραπάνω (Πίνακας 5.7).

Πίνακας 5.7: Σύγκριση τήρησης πολιτικής: πλήρες σύστημα έναντι ενιαίου prompt

Σύστημα	D1	D2	D3	D4	Σύνολο
Πλήρες σύστημα	✓	✓	✓	✓	4/4
Μία κλήση με την πολιτική	✓	✗	✗	✗	1/4
Μία κλήση με πολιτική και κατώφλια	✗	✓	✓	✓	3/4

Η προσθήκη των κατωφλίων βελτίωσε μετρήσιμα την τήρηση της πολιτικής, από μία στις τέσσερις σε τρεις στις τέσσερις. Παρατηρείται όμως κάτι που δεν φαίνεται στο σύνολο, μόνο στη λεπτομέρεια. Η δεύτερη, πλουσιότερη εκδοχή αστόχησε ακριβώς στο δείγμα που η πρώτη, φτωχότερη εκδοχή είχε απαντήσει σωστά, το D1, το ευκολότερο δείγμα ολόκληρου του συνόλου. Η προσθήκη πληροφορίας που διόρθωσε τα D2 έως D4 συνέπεσε με απώλεια ακριβώς εκεί που δεν χρειαζόταν καμία διόρθωση. Το συμπέρασμα που στηρίζεται άμεσα από τα δεδομένα είναι συγκεκριμένο: η ρητή οδηγία μέσα σε ένα prompt βοηθά, χωρίς όμως να εξασφαλίζει.

Πέρα από αυτό το εύρημα, υπάρχει και ένα δεύτερο σημείο, ανεξάρτητο από το πόσο καλά γραμμένο είναι το prompt. Η προτεινόμενη αρχιτεκτονική διαχωρίζει την παραγωγή της απάντησης από την επικύρωσή της σε δύο διακριτά στάδια, και αυτό επιτρέπει κάτι που καμία διατύπωση prompt δεν μπορεί να προσφέρει από μόνη της: την εφαρμογή ντετερμινιστικών κανόνων πάνω στην απόφαση αφού αυτή έχει ήδη παραχθεί. Ο έλεγχος κατωφλίου και ο έλεγχος συνέπειας που περιγράφηκαν παραπάνω δεν είναι οδηγίες προς το μοντέλο τις οποίες αυτό καλείται να ακολουθήσει, είναι κώδικας που εξετάζει το τελικό αποτέλεσμα. Ένα ενιαίο prompt μπορεί βεβαίως να ζητήσει από το μοντέλο να ελέγξει και να αναθεωρήσει το ίδιο την απάντησή του πριν την επιστρέψει, ο έλεγχος αυτός όμως παραμένει κρίση του ίδιου μοντέλου που μόλις έκανε το σφάλμα. Στην προτεινόμενη αρχιτεκτονική πρόκειται για ανεξάρτητο και επαναλήψιμο κανόνα, ο οποίος δεν μπορεί να παρακαμφθεί από το ίδιο σφάλμα που καλείται να εντοπίσει.

5.6 Κόστος και Χρόνος Απόκρισης

Οι ιδιότητες που περιγράφηκαν στις δύο προηγούμενες ενότητες έχουν τίμημα, και είναι μετρήσιμο (Πίνακας 5.8).

Πίνακας 5.8: Κόστος και χρόνος απόκρισης ανά υποβολή

Μέτρηση, σύνολο πυρήνα	Πολυπρακτορικό	Αναφοράς
Tokens ανά υποβολή	4.646	779
Χρόνος απόκρισης	30,1 s	2,6 s
Κλήσεις μοντέλου ανά υποβολή	3,3	1,0

Ο χρόνος απόκρισης είναι η σοβαρότερη επιβάρυνση, και προκύπτει άμεσα από τη σειριακή εξάρτηση των σταδίων στην τρέχουσα σχεδίαση. Μια υποβολή κώδικα ενεργοποιεί διαδοχικές κλήσεις, καθεμία από τις οποίες στηρίζεται στο αποτέλεσμα της προηγούμενης, οπότε ο συνολικός χρόνος είναι άθροισμα και όχι μέγιστο, όπως αναλύθηκε στην ενότητα 4.11. Τριάντα δευτερόλεπτα αναμονής είναι αισθητά για έναν φοιτητή που περιμένει μπροστά στην οθόνη, και η μείωσή τους αποτελεί την πρώτη προτεραιότητα μελλοντικής εργασίας. Το κόστος σε tokens είναι σαφώς πιο διαχειρίσιμο, αφορά υπολογιστικούς πόρους και όχι εμπειρία αναμονής, οπότε ο χρόνος παραμένει ο πραγματικός περιορισμός.

Ο μέσος όρος των 3,3 κλήσεων ανά υποβολή αξίζει επίσης εξήγηση, καθώς η αρχιτεκτονική περιγράφει τρεις πράκτορες και θα ανέμενε κανείς σταθερά τρεις κλήσεις. Το πλήθος όμως προσαρμόζεται στην περίπτωση, και η κατανομή του αποκαλύπτει κάτι για τη λογική του συστήματος (Πίνακας 5.9).

Πίνακας 5.9: Κατανομή πλήθους κλήσεων μοντέλου ανά περίπτωση

Κλήσεις	Δείγματα	Περίπτωση
2	2	Συντακτικό σφάλμα
3	12	Σωστή λύση
4	10	Σημασιολογικό λάθος

Στο συντακτικό σφάλμα ο Debugger δεν καλεί καθόλου μοντέλο, καθώς ο parser της Python εντοπίζει το πρόβλημα με βεβαιότητα, και ο Mentor χρησιμοποιεί προδιατυπωμένη παιδαγωγική εξήγηση αντί να παραγάγει νέα. Δύο κλήσεις εξοικονομούνται επομένως από τον ντετερμινισμό. Στη σωστή λύση απαιτούνται τρεις. Στο σημασιολογικό λάθος τέσσερις, γιατί ο Mentor παράγει χωριστά την εισαγωγική φράση και την ίδια την υπόδειξη. Η κατανομή αυτή αποτελεί την αριθμητική αποτύπωση της αρχής που διατυπώθηκε στο Κεφάλαιο 3: το φθηνότερο μονοπάτι του συστήματος είναι ακριβώς η περίπτωση στην οποία η ντετερμινιστική ανάλυση επαρκεί από μόνη της, χωρίς να χρειάζεται καθόλου κρίση μοντέλου.

Η ερμηνεία του συνολικού κόστους δεν μπορεί να γίνει σε κενό, πρέπει να σταθμιστεί απέναντι στις λειτουργίες που αγοράζει, και στην παρούσα αξιολόγηση αυτές είναι συγκεκριμένες και μετρημένες, όχι υποθετικές: παιδαγωγική απόφαση που στηρίζεται στο ιστορικό και τηρεί ρητή πολιτική σε τέσσερις από τέσσερις περιπτώσεις, δύο μηχανισμοί επικύρωσης που ενεργοποιήθηκαν σε δύο από τις τέσσερις και διόρθωσαν αστοχίες διαφορετικού τύπου, τριάντα στις τριάντα εξόδους πλήρως θεμελιωμένες στον κώδικα, και συστηματικά συντομότερη ανατροφοδότηση σε κάθε σύνολο.

5.7 Σύγκριση, Συμπεράσματα και Περιορισμοί

Οι δύο μέθοδοι της αξιολόγησης δεν έδειξαν το ίδιο πράγμα, και ακριβώς επειδή δεν το έδειξαν, ο συνδυασμός τους είναι πιο πειστικός από όσο θα ήταν η καθεμία μόνη της. Η σύγκριση με το σύστημα αναφοράς έδειξε ότι η αρχιτεκτονική διατηρεί ανέπαφη τη βασική ικανότητα κρίσης κώδικα, ενώ προσφέρει, με σαφώς μεγαλύτερο κόστος, λειτουργίες που το σύστημα αναφοράς δεν παρέχει καν στη βασική του μορφή. Αυτό όμως δεν αρκούσε για να δειχθεί ποιο κομμάτι της αρχιτεκτονικής ευθύνεται για τη διαφορά, ούτε αν η ίδια συμπεριφορά θα μπορούσε να αναπαραχθεί με μια απλή επέκταση του συστήματος αναφοράς. Τα πειράματα αφαίρεσης κάλυψαν ακριβώς αυτό το κενό,

δείχνοντας ότι το ιστορικό, η επικύρωση και η θεμελίωση συνεισφέρουν το καθένα μετρήσιμα και ανεξάρτητα, μέσα στο ίδιο σύστημα, χωρίς καμία από τις δύο συνθήκες σύγκρισης να έχει πλεονέκτημα λόγω άνισης πληροφόρησης. Και το πείραμα του ενιαίου prompt έδειξε ότι ακόμα κι όταν το σύστημα αναφοράς εμπλουτίζεται ρητά με την ίδια πολιτική και τα ίδια κατώφλια, φτάνει τρεις στις τέσσερις σωστές αποφάσεις, όχι τέσσερις, και μάλιστα αστοχεί ακριβώς εκεί όπου η απλούστερη εκδοχή του είχε πετύχει.

Με βάση αυτά, οι απαντήσεις στα τέσσερα ερευνητικά ερωτήματα είναι οι εξής. Ως προς το E1, η αρχιτεκτονική διατήρησε τη βασική ακρίβεια χωρίς να την υποβαθμίσει ή να την ξεπεράσει, ισοφαρίζοντας το σύστημα αναφοράς σε κάθε ένα από τα τρία σύνολα δοκιμής. Για το E2, παρήγαγε παιδαγωγική απόφαση προόδου στηριγμένη στο ιστορικό, κάτι που το σύστημα αναφοράς δεν προσφέρει στη βασική του μορφή, και ανατροφοδότηση σταθερά συντομότερη σε κάθε σύνολο. Στο E3, η αφαίρεση είτε του ιστορικού είτε της επικύρωσης μείωσε την ορθότητα της απόφασης από τέσσερις σωστές σε δύο, με δύο διαφορετικού τύπου αστοχίες να εμφανίζονται όταν έλειπε η επικύρωση. Ως προς το E4, η δομημένη πληροφορία προς τον Mentor θεμελίωσε πλήρως και τις τριάντα εξόδους, έναντι είκοσι έξι στις τριάντα χωρίς αυτήν, με τις μη τεκμηριωμένες αναφορές να εμφανίζονται ακριβώς εκεί όπου έλειπε η πληροφορία και πουθενά αλλού.

Οι τέσσερις αυτές απαντήσεις, μαζί, απαντούν με σαφήνεια και στους δύο αρχικούς στόχους της εργασίας. Ως προς τη μελέτη της αποτελεσματικότητας των Agentic Workflows στην εκπαιδευτική διαδικασία (E1 έως E3), η αξιολόγηση έδειξε ότι η πολυπρακτορική αρχιτεκτονική διατηρεί την ίδια ακρίβεια κρίσης με μια απλούστερη εναλλακτική, προσφέρει πάνω σε αυτήν λειτουργίες που η εναλλακτική δεν παρέχει με την ίδια αξιοπιστία, και ότι οι λειτουργίες αυτές προέρχονται μετρήσιμα από συγκεκριμένα συστατικά της αρχιτεκτονικής και όχι από τύχη. Ως προς τη μείωση των ψευδαισθήσεων μέσω δομημένης γνώσης (E4), η αξιολόγηση έδειξε άμεσα και ελεγχόμενα ότι η παρουσία δομημένης πληροφορίας εξαλείφει τους ψευδείς ισχυρισμούς που αλλιώς εμφανίζονται, μέσα στο ίδιο ακριβώς σύστημα, με μόνη μεταβλητή την παρουσία ή απουσία της.

Αυτή είναι, σε πρακτική μορφή, η αρχή του bounded determinism [9] της ενότητας 2.3: τα Μεγάλα Γλωσσικά Μοντέλα προσφέρουν αξία σε εκπαιδευτικά συστήματα όταν ο ρόλος τους είναι σαφώς οριοθετημένος και συνοδεύεται από ντετερμινιστικό έλεγχο. Η παρούσα εργασία προσθέτει σε αυτή τη διαπίστωση ένα στοιχείο. Ο έλεγχος δεν τοποθετήθηκε σε ένα σημείο της ροής, επαναλήφθηκε σε καθέναν από τους τρεις πράκτορες, και η αξιολόγηση μέτρησε τι απέδωσε αυτή η επανάληψη, με πρόσθετο κόστος σχεδόν δωδεκαπλάσιου χρόνου απόκρισης και εξαπλάσιας κατανάλωσης tokens σε σχέση με τη μία κλήση.

Οι τέσσερις αυτές μετρημένες ιδιότητες αντιστοιχούν η καθεμία σε συγκεκριμένο παιδαγωγικό μηχανισμό, ήδη τεκμηριωμένο στη βιβλιογραφία που παρουσιάστηκε στα προηγούμενα κεφάλαια. Η παιδαγωγική απόφαση προόδου εφαρμόζει την αρχή της mastery learning, η οποία αξιοποιείται για την αντιμετώπιση του Προβλήματος των Δύο Τυπικών Αποκλίσεων με το οποίο ανοίγει η εργασία (1.1). Η προαγωγή προϋποθέτει επίδειξη ανεξάρτητης επάρκειας, όχι απλώς τελικά ορθή απάντηση. Η ενότητα 5.5 έδειξε ότι το σύστημα την τηρεί με συνέπεια όταν διαθέτει το ιστορικό του φοιτητή, και τη χάνει όταν αυτό αφαιρεθεί. Η συντομότερη ανατροφοδότηση συνδέεται με χαμηλότερο γνωστικό φόρτο, ο οποίος έχει συσχετιστεί εμπειρικά με υψηλότερη επίδοση σε εισαγωγικό προγραμματισμό Python [41]. Η μη αποκάλυψη της λύσης εφαρμόζει τη Σωκρατική μέθοδο, όπου η πρόωγη αποκάλυψη στερεί από τον μαθητή την ευκαιρία να σκεφτεί μόνος του [31]. Και η θεμελίωση των ισχυρισμών αποτρέπει το είδος παραπλανητικής βεβαιότητας που η βιβλιογραφία γύρω από τις ψευδαισθήσεις σε εκπαιδευτικό πλαίσιο συνδέει με βλάβη στη μάθηση [19].

Η αξιολόγηση μέτρησε επομένως με πόση πιστότητα το σύστημα εφαρμόζει μηχανισμούς που η βιβλιογραφία συνδέει με μαθησιακά αποτελέσματα. Το αν αυτή η εφαρμογή μεταφράζεται σε μετρήσιμη βελτίωση με πραγματικούς φοιτητές παραμένει ανοιχτή κατεύθυνση για το Κεφάλαιο 6.

Η μελέτη έχει περιορισμούς που πρέπει να αναγνωριστούν ρητά. Τα σύνολα δοκιμής είναι μικρά, και το μικρότερο από αυτά, τα τέσσερα δείγματα απόφασης προόδου, αποτελεί τον σημαντικότερο περιορισμό. Τα αποτελέσματα δείχνουν σαφή κατεύθυνση και συνοδεύονται από άμεση παρατήρηση της επίδρασης που είχε η αφαίρεση κάθε συστατικού, γεγονός που παρέχει ισχυρή ένδειξη της συνεισφοράς τους, δεν επιτρέπουν όμως γενίκευση σε ποσοστά, ούτε ισχυρισμό αιτιότητας με την αυστηρή έννοια, κάτι που θα απαιτούσε σαφώς μεγαλύτερα σύνολα και πολλαπλές επαναλήψεις. Οι περισσότερες μετρήσεις προέρχονται από μία εκτέλεση ανά δείγμα. Το πέρασμα σταθερότητας εξέτασε έξι δείγματα σε τρεις επαναλήψεις το καθένα, με τον πλήρη αγωγό και για τα δύο συστήματα, και έδωσε απολύτως ταυτόσημο αποτέλεσμα και στις έξι περιπτώσεις, γεγονός που στηρίζει την αξιοπιστία των υπολοίπων μετρήσεων του κεφαλαίου χωρίς βεβαίως να την αποδεικνύει για κάθε δυνατό δείγμα. Η μετρική της αποκάλυψης λύσης χρειάζεται ιδιαίτερη προσοχή, πρόκειται για σπάνιο γεγονός, και σε ένα σύνολο οκτώ δειγμάτων μία μεμονωμένη εμφάνιση θα μετέβαλλε δραματικά το ποσοστό. Στην τελική μέτρηση καμία από τις δύο προσεγγίσεις δεν αποκάλυψε τη λύση, αλλά το μέγεθος του συνόλου δεν επαρκεί για να τις διακρίνει με βεβαιότητα ως προς αυτήν.

Το σύστημα αναφοράς αποτελεί μια συγκεκριμένη ελάχιστη προσέγγιση με μία διαμόρφωση, και τα ευρήματα αφορούν αυτήν και όχι τα μεμονωμένα πρακτορικά συστήματα γενικά. Αντίστοιχα, το πείραμα του ενιαίου prompt εξέτασε μία συγκεκριμένη διατύπωση των κανόνων μέσα στο prompt, και μια διαφορετική διατύπωση θα μπορούσε να δώσει άλλο αποτέλεσμα. Η αξιολόγηση εξέτασε επίσης τη συμπεριφορά του συστήματος απέναντι σε προκατασκευασμένες υποβολές. Το αν οι συντομότερες και θεμελιωμένες υποδείξεις οδηγούν σε καλύτερα μαθησιακά αποτελέσματα απαιτεί μελέτη με πραγματικούς φοιτητές, η οποία προτείνεται στο επόμενο κεφάλαιο. Η επισημείωση της θεμελίωσης έγινε από έναν κριτή, με υποβοήθηση από αυτόματο εργαλείο, και δεν υπολογίστηκε συμφωνία μεταξύ κριτών. Τέλος, όλες οι μετρήσεις έγιναν με ένα μοντέλο και έναν πάροχο, και η επανάληψη με άλλα μοντέλα θα έδειχνε κατά πόσο τα ευρήματα, ιδίως η θεμελίωση και η επίδραση της επικύρωσης, διατηρούνται ανεξάρτητα από τη συγκεκριμένη επιλογή.

Το συνολικό συμπέρασμα του κεφαλαίου είναι συγκεκριμένο και οριοθετημένο. Η αξιολόγηση δεν έδειξε ότι η πολυπρακτορική αρχιτεκτονική κρίνει ακριβέστερα τον κώδικα από μια μοναδική κλήση. Έδειξε ότι προσθέτει γύρω από αυτή την κρίση λειτουργίες που εξαρτώνται από ιστορικό, ντετερμινιστική επικύρωση και δομημένα ευρήματα και ότι οι λειτουργίες αυτές συνοδεύονται από σημαντικό κόστος σε χρόνο.

Κεφάλαιο 6ο: Συμπεράσματα και Μελλοντική Εργασία

6.1 Εισαγωγή

Τα προηγούμενα κεφάλαια περιέγραψαν τον σχεδιασμό, την υλοποίηση και την αξιολόγηση του προτεινόμενου συστήματος. Το κεφάλαιο αυτό κλείνει την εργασία, συνοψίζοντας τη συνεισφορά της και απαντώντας ρητά στους δύο ερευνητικούς στόχους της ενότητας 1.2. Η ενότητα 6.2 ανακεφαλαιώνει τι κατασκευάστηκε και τι έδειξε η αξιολόγηση. Η ενότητα 6.3 προτείνει μια συγκεκριμένη επέκταση του μηχανισμού επικύρωσης, η οποία θα διεύρυνε περαιτέρω την εφαρμογή της ντετερμινιστικής θεμελίωσης. Η ενότητα 6.4 παρουσιάζει ευρύτερες κατευθύνσεις μελλοντικής εργασίας, και η ενότητα 6.5 κλείνει την εργασία.

6.2 Ανακεφαλαίωση και Συνεισφορά

Η εργασία ξεκίνησε από μια διαπίστωση που δεν αφορά μόνο τον προγραμματισμό. Η εξατομικευμένη διδασκαλία λειτουργεί καλύτερα από τη μαζική, όπως έδειξε το πρόβλημα των δύο τυπικών αποκλίσεων, αλλά δεν κλιμακώνεται εύκολα σε τάξεις με πολλούς φοιτητές [1]. Τα Μεγάλα Γλωσσικά Μοντέλα προσφέρθηκαν ως πιθανή λύση, όχι όμως χωρίς κόστος, καθώς δεν έχουν σχεδιαστεί ειδικά για εκπαιδευτικά πλαίσια και τείνουν να παράγουν πειστική αλλά εσφαλμένη πληροφορία [3]. Η βιβλιογραφική διαδρομή του Κεφαλαίου 2 κατέληξε σε μια σαφή κατεύθυνση, η υβριδική ενσωμάτωση δομημένης γνώσης μαζί με τη γλωσσική ικανότητα ενός μοντέλου είναι προτιμότερη από την ανεξέλεγκτη χρήση του. Η παρούσα εργασία υλοποίησε ακριβώς αυτή την κατεύθυνση, μέσα από μια πολυπρακτορική αρχιτεκτονική τριών εξειδικευμένων πρακτόρων.

Το κεντρικό μοτίβο σχεδίασης, ο συνδυασμός ντετερμινιστικής θεμελίωσης και κρίσης του μοντέλου, εφαρμόστηκε με συνέπεια και στους τρεις πράκτορες. Ο Debugger συγκεντρώνει αντικειμενικά ευρήματα από στατική ανάλυση πριν κρίνει τι πρόβλημα υπάρχει. Ο Assessor περνά τον κώδικα από τρία επίπεδα ντετερμινιστικού ελέγχου πριν κληθεί το μοντέλο, και επικυρώνει την απάντησή του με τα ίδια αντικειμενικά κριτήρια μετά. Ο Mentor, τέλος, λαμβάνει τη λίστα των δομών που υπάρχουν πραγματικά στον κώδικα του φοιτητή, ώστε η παιδαγωγική του απάντηση να μην βασίζεται σε εικασίες. Το σύστημα δεν εμπιστεύεται ποτέ το μοντέλο να αποφασίσει μόνο του πάνω σε δεδομένα που δεν έχουν επαληθευτεί, αλλά και δεν του αφαιρεί την ευθύνη της τελικής κρίσης, ακολουθώντας την αρχή του bounded determinism [9] που τέθηκε στο Κεφάλαιο 2.

Η αξιολόγηση του Κεφαλαίου 5 απάντησε και στους δύο ερευνητικούς στόχους, με δεδομένα και όχι με επιχειρήματα.

Ως προς τον πρώτο στόχο, τη μελέτη της αποτελεσματικότητας μιας πολυπρακτορικής αρχιτεκτονικής στην εκπαιδευτική διαδικασία, η αξιολόγηση επιβεβαίωσε ότι η προσέγγιση είναι πράγματι αποτελεσματική. Στη βασική δουλειά της κρίσης κώδικα, η αρχιτεκτονική απέδωσε εξίσου καλά με μια απλούστερη προσέγγιση μίας κλήσης, φτάνοντας το ίδιο ανώτατο δυνατό σκορ στα 32 δείγματα του πυρήνα και της ανθεκτικότητας, και ταυτόσημη επίδοση στο σύνολο επιφανειακών μοτίβων, χωρίς καμία απώλεια ακρίβειας σε σχέση με το σύστημα αναφοράς. Πάνω σε αυτή τη σταθερή βάση, η αρχιτεκτονική προσέφερε και επιπλέον, καθαρά παιδαγωγικά οφέλη που μια μοναδική κλήση δεν παρήγαγε με την ίδια αξιοπιστία. Η παιδαγωγική απόφαση προόδου, που στηρίζεται στο ιστορικό του φοιτητή και όχι μόνο στην ορθότητα του κώδικα, πέτυχε τέσσερις στις τέσσερις περιπτώσεις, ενώ η αφαίρεση είτε του ιστορικού είτε της επικύρωσης την υποβάθμιζε στο μισό. Η ρητή απαγόρευση αποκάλυψης της λύσης τηρήθηκε απόλυτα, με τη συγκράτηση να προκύπτει από το ότι ο πράκτορας που γράφει την απάντηση δεν διαθέτει καν τον κώδικα, όχι από την τήρηση μιας οδηγίας. Η ανατροφοδότηση ήταν σταθερά συντομότερη σε κάθε σύνολο δοκιμής. Η πολυπρακτορική αρχιτεκτονική δεν ήταν λοιπόν απλά εξίσου καλή στο βασικό έργο, εφάρμοσε με μεγαλύτερη συνέπεια συγκεκριμένους παιδαγωγικούς μηχανισμούς που η βιβλιογραφία συνδέει με πραγματική μάθηση, πέρα από τη σωστή κρίση κώδικα.

Ως προς τον δεύτερο στόχο, τη μείωση των ψευδαισθήσεων μέσω δομημένης γνώσης, το εύρημα ήταν το πιο καθαρό της εργασίας. Όταν ο Mentor λάμβανε τη λίστα των δομών που υπάρχουν πραγματικά στον κώδικα, καμία από τις τριάντα εξόδους που εξετάστηκαν δεν περιείχε ισχυρισμό που ο κώδικας διαψεύδει. Χωρίς αυτή τη λίστα, τέσσερις εμφάνιζαν ακριβώς αυτό το πρόβλημα, με δύο από αυτές μέσα σε μηνύματα επιβράβευσης προς φοιτητή που είχε ήδη απαντήσει σωστά. Το εύρημα δεν είναι θεωρητικό. Μετρήθηκε μέσα στο ίδιο σύστημα, με τα πάντα σταθερά εκτός από έναν διακόπτη, και

έδειξε ότι το ίδιο μοντέλο συμπεριφέρεται διαφορετικά ανάλογα με το αν διαθέτει τα δεδομένα που χρειάζεται ή αναγκάζεται να τα εικάσει.

Η συνεισφορά της εργασίας συνοψίζεται σε μία πρόταση. Η πολυπρακτορική αρχιτεκτονική δεν έκανε το μοντέλο εξυπνότερο σε ό,τι ήδη έκανε καλά, το έκανε ελέγξιμο σε ό,τι δεν μπορούσε να εμπιστευτεί κανείς σε μια μοναδική, ανεξέλεγκτη κλήση. Το τίμημα αυτού του ελέγχου, δωδεκαπλάσιος χρόνος απόκρισης και εξαπλάσιο κόστος σε tokens, είναι μετρημένο και δεν αποκρύπτεται. Το ζητούμενο δεν ήταν η ταχύτητα, ήταν η αξιοπιστία, και η εργασία δείχνει πού ακριβώς αυτή κερδίζεται.

6.3 Επέκταση του Μηχανισμού Επικύρωσης σε Σύνθετες Δομικές Συνθήκες

Ο έλεγχος κριτηρίων της ενότητας 4.7 αναγνωρίζει ήδη ισοδυναμίες ανάμεσα σε διαφορετικές δομικές μορφές του ίδιου νοήματος, μέσα από σύνθετες, πολυμεταβλητές συνθήκες. Ένα εμφωλευμένο if μέσα σε else, για παράδειγμα, αναγνωρίζεται ως λειτουργικά ισοδύναμο με elif μέσα από τη σύνθετη συνθήκη `has_if` και ταυτόχρονα `has_elif` ή `has_boolor`. Ανοίγει έτσι ένα φυσικό ερώτημα, κατά πόσο η ίδια λογική θα μπορούσε να επεκταθεί και στο επίπεδο του μηχανισμού επικύρωσης της ενότητας 3.6, ώστε να αξιοποιεί αυτού του είδους τις σύνθετες συνθήκες σε ακόμη περισσότερα σημεία της αξιολόγησης.

Ο μηχανισμός επικύρωσης λειτουργεί σήμερα συγκρίνοντας τη διατύπωση μιας απάντησης με ένα σύνολο γνωστών δομικών σημαιών, καθεμία εκ των οποίων αντιστοιχεί σε μία μεμονωμένη σημαία, όπως η απουσία εκτύπωσης με τη σημαία `has_reachable_print`. Το σύνολο αυτό καλύπτει ήδη τις πιο συχνές κατηγορίες, κλήση συνάρτησης, εκτύπωση, ορισμό συνάρτησης, βρόχο `for`, λίστα και `return`. Μια φυσική κατεύθυνση βελτίωσης είναι η γενίκευση αυτού του μηχανισμού ώστε να υποστηρίζει και σύνθετες, πολυμεταβλητές εκφράσεις σημαιών, όπως αυτή που ήδη χρησιμοποιεί ο έλεγχος κριτηρίων για το `elif`. Κάτι τέτοιο θα επέτρεπε στο σύστημα να αναγνωρίζει αυτόματα ισοδυναμίες και σε ακόμη πιο σύνθετα δομικά μοτίβα, για παράδειγμα σε συνδυασμούς εμφωλευμένων βρόχων ή σε συναρτήσεις με πολλαπλές παραμέτρους και προεπιλεγμένες τιμές, διευρύνοντας το εύρος των παιδαγωγικών ισχυρισμών που μπορούν να ελεγχθούν αυτόματα πριν φτάσουν στον φοιτητή.

Η επέκταση αυτή θα εντασσόταν φυσικά στην υπάρχουσα αρχιτεκτονική, καθώς η λογική των σύνθετων συνθηκών χρησιμοποιείται ήδη στον έλεγχο κριτηρίων της ενότητας 4.7. Θα επρόκειτο, δηλαδή, για τη μεταφορά μιας ήδη δοκιμασμένης λογικής σε ένα ακόμη σημείο της αξιολόγησης, ενισχύοντας περαιτέρω το κεντρικό μοτίβο του συνδυασμού ντετερμινιστικής θεμελίωσης και κρίσης του μοντέλου που διατρέχει ολόκληρη την εργασία.

6.4 Κατευθύνσεις Μελλοντικής Εργασίας

Η πρώτη κατεύθυνση αφορά την αξιολόγηση με πραγματικούς φοιτητές. Η παρούσα εργασία αξιολόγησε την τεχνική και λειτουργική ορθότητα της αρχιτεκτονικής, όχι την παιδαγωγική της αποτελεσματικότητα. Ότι η ανατροφοδότηση του προτεινόμενου συστήματος είναι σταθερά συντομότερη και χωρίς αποκάλυψη λύσης είναι μετρήσιμο γεγονός. Το αν αυτό βοηθά πράγματι έναν φοιτητή να μάθει καλύτερα προγραμματισμό είναι διαφορετικό ερώτημα, που απαιτεί μελέτη με πραγματικούς χρήστες και μέτρηση μαθησιακών αποτελεσμάτων, όχι μόνο χαρακτηριστικών της απάντησης.

Μια δεύτερη, εξίσου άμεση κατεύθυνση είναι η μείωση του χρόνου απόκρισης, το σαφέστερο μειονέκτημα που κατέγραψε η αξιολόγηση. Μια πιθανή λύση είναι η διαφοροποίηση του μεγέθους του μοντέλου ανάλογα με το είδος της κλήσης, κατ' αναλογία με τη διαφοροποίηση της θερμοκρασίας

(4.2). Οι κλήσεις κλειστού συνόλου, όπως η επιλογή κατηγορίας σφάλματος, δεν χρειάζονται το ίδιο μέγεθος μοντέλου με την κρίση του Assessor ή την παραγωγή κειμένου από τον Mentor, και θα μπορούσαν να εξυπηρετηθούν από ένα μικρότερο, ταχύτερο μοντέλο χωρίς απώλεια ακρίβειας. Η σταδιακή εμφάνιση της απάντησης καθώς παράγεται, αντί για αναμονή της πλήρους ολοκλήρωσης, θα μείωνε επιπλέον τον αντιληπτό χρόνο αναμονής, χωρίς να αλλάξει τον πραγματικό.

Όλες οι μετρήσεις της αξιολόγησης έγιναν με ένα μοντέλο και έναν πάροχο. Η επανάληψη της ίδιας μεθοδολογίας με μοντέλα διαφορετικού μεγέθους ή κατασκευαστή θα έδειχνε αν τα ευρήματα, ιδίως η θεμελίωση των ισχυρισμών και η επίδραση της επικύρωσης, διατηρούνται ανεξάρτητα από τη συγκεκριμένη επιλογή μοντέλου.

Η λογική παρακολούθησης προόδου και δυσκολίας που ήδη διαθέτει το σύστημα ανοίγει τον δρόμο για πρόσθετα εκπαιδευτικά modules. Ένα module αυτόματης σύστασης ασκήσεων θα μπορούσε να προτείνει στον φοιτητή τη συγκεκριμένη επόμενη άσκηση που ταιριάζει στο επίπεδό του, αντί να περιορίζεται στην αξιολόγηση της τρέχουσας. Ένα module συγκεντρωτικής ανάλυσης για τον διδάσκοντα, που θα συγκέντρωνε ανωνυμοποιημένα τα συχνότερα λάθη μιας τάξης, θα μετέτρεπε τα δεδομένα που παράγει ήδη το σύστημα σε χρήσιμη πληροφορία για τον σχεδιασμό του μαθήματος.

Η στατική ανάλυση βασίζεται σήμερα στη μονάδα ast της Python, φυσική επιλογή αφού αυτή είναι η γλώσσα διδασκαλίας. Η ίδια αρχή, δομημένη εξαγωγή χαρακτηριστικών πριν την κρίση του μοντέλου, θα μπορούσε να εφαρμοστεί και σε άλλες γλώσσες με αντίστοιχα εργαλεία στατικής ανάλυσης, διευρύνοντας το πεδίο εφαρμογής σε περισσότερα μαθήματα εισαγωγής στον προγραμματισμό.

Τέλος, δύο κατευθύνσεις αφορούν την ίδια τη μεθοδολογία αξιολόγησης. Το σύνολο απόφασης προόδου περιείχε τέσσερα δείγματα, αρκετά για να δείξει σαφή κατεύθυνση αλλά όχι για γενίκευση σε ποσοστά. Ένα μεγαλύτερο σύνολο θα επέτρεπε ισχυρότερα στατιστικά συμπεράσματα. Η επισημείωση της θεμελίωσης έγινε από έναν κριτή. Ένας δεύτερος, ανεξάρτητος κριτής σε υποσύνολο των δειγμάτων θα επέτρεπε τον υπολογισμό συμφωνίας μεταξύ κριτών.

6.5 Επίλογος

Ο τίτλος της εργασίας θέτει έναν αυτόνομο πράκτορα για την εξατομικευμένη υποστήριξη και αξιολόγηση φοιτητών στον προγραμματισμό. Το Κεφάλαιο 3 έδειξε ότι το σύστημα διαθέτει τις τέσσερις ιδιότητες που ορίζουν έναν πράκτορα, αυτονομία, αντιδραστικότητα, προδραστικότητα και κοινωνική ικανότητα. Το Κεφάλαιο 4 έδειξε την ίδια αρχή στην πράξη: η επιλογή του Mentor να μην καλέσει καθόλου μοντέλο μπροστά σε γνήσιο συντακτικό λάθος δεν είναι απουσία αυτονομίας, είναι η ίδια η αυτονομία να αναγνωρίζει πότε δεν υπάρχει κρίση να ασκηθεί. Το Κεφάλαιο 5 έδειξε κάτι διαφορετικό αλλά εξίσου απαραίτητο, ότι αυτή η ιδιότητα του πράκτορα συνοδεύεται από μετρήσιμο όφελος και όχι μόνο από αρχιτεκτονική κομψότητα.

Η εργασία δείχνει κάτι συγκεκριμένο: η οργάνωση ενός Μεγάλου Γλωσσικού Μοντέλου γύρω από ντετερμινιστική θεμελίωση και επικύρωση παράγει μετρήσιμα διαφορετική, και σε συγκεκριμένες διαστάσεις καλύτερη, συμπεριφορά από μια ανεξέλεγκτη μοναδική κλήση, με τίμημα που είναι εξίσου μετρήσιμο. Αυτό δεν σημαίνει ότι λύθηκε το ευρύτερο πρόβλημα της αξιοπιστίας αυτοματοποιημένης διδασκαλίας προγραμματισμού, μόνο ότι δείχθηκε πού και πώς κερδίζεται αξιοπιστία σε ένα συγκεκριμένο, καλά ορισμένο κομμάτι του. Η επέκταση που προτάθηκε στην ενότητα 6.3 ακολουθεί την ίδια λογική, δείχνοντας ότι η αρχή έχει ακόμη περιθώριο να εφαρμοστεί σε ευρύτερο φάσμα δομικών συνθηκών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] M. Jazayeri, “Combining Mastery Learning with Project-Based Learning in a First Programming Course: An Experience Report,” In Proc. IEEE/ACM 37th International Conference on Software Engineering (ICSE), 2015, pp. 315-318.
- [2] U. Z. Ahmed, N. Srivastava, R. Sindhgatta, and A. Karkare, “Characterizing the Pedagogical Benefits of Adaptive Feedback for Compilation Errors by Novice Programmers,” In Proc. IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), 2020, pp. 139-150.
- [3] D. Hooshyar, Y. Yang, G. Šír, T. Kärkkäinen, R. Hämäläinen, M. Cukurova, and R. Azevedo, “Why Large Language Models Alone Fall Short for Responsible Learner Modeling in K–12 Tutoring: A Case Study,” IEEE Access, vol. 14, pp. 91267-91284, Jun. 2026.
- [4] D. Ye, M. Zhang, and A. V. Vasilakos, “A Survey of Self-Organization Mechanisms in Multiagent Systems,” IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 47, no. 3, pp. 441-461, Mar. 2017.
- [5] H. Mubarak, “Developing Flexible Software Using Agent-Oriented Software Engineering,” IEEE Software, vol. 25, no. 5, pp. 13-15, Sep. 2008.
- [6] B. F. Marin, A. Hunger, and S. Werner, “Corroborating Role Theory and Intelligent Agents: A New Paradigm to Support Collaborative Learning?” In Proc. IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT’05), 2005, pp. 690-693.
- [7] P. Guo and H. Qi, “Study on Interactive System for Teaching Programming (ISTP) Based on Intelligent Agent,” In Proc. 6th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), 2010, pp. 1-4.
- [8] T. Divya, V. Prashanthi, V. Srikanth, and V. Gopalachari M, “A Survey on Multi-Agent LLM Frameworks with Semantic Memory Integration and Dynamic DAG Orchestration,” In Proc. 7th International Conference on Inventive Research in Computing Applications (ICIRCA), 2026, pp. 1922-1928.
- [9] P. Yazdanian, Y. Liu, and Z. Li, “A Comparative Study Towards Designing a Hybrid Architecture of Microservices and LLM-based Multi-Agent Systems,” In Proc. 32nd Asia-Pacific Software Engineering Conference (APSEC), 2025, pp. 761-772.
- [10] M. Becattini, R. Verdecchia, and E. Vicario, “SALLMA: A Software Architecture for LLM-Based Multi-Agent Systems,” In Proc. IEEE/ACM International Workshop on New Trends in Software Architecture (SATrends), 2025, pp. 5-8.
- [11] H. Cao, K. Liu, L. Tan, N. Chen, X. Cao, S. He, and P. Zhang, “A Think-Do Architecture for Cognitive Autonomy: LLM-Driven Agentic AI for Sustainable Resource Orchestration in 6G Satellite Networks,” IEEE Network, 2026.
- [12] A. Hajaghaie and R. K. Thulasiram, “Agentic Portfolio Construction: A Multi-Agent Architecture for LLM-Driven Financial Asset Allocation,” In Proc. 3rd International Conference on Foundation and Large Language Models (FLLM), 2025, pp. 192-201.

- [13] L. Daberao, "A Context-Aware Intelligent Tutoring Framework for Grammar Modeling Using Large Language Models," In Proc. IEEE Conference on Pervasive and Intelligent Computing (PICom), 2025, pp. 200-204.
- [14] R. Abaab and M. A. Khenissi, "Enhancing Intelligent Tutoring Systems with Large Language Models: Architecture and Implementation," In Proc. 10th International Conference on ICT & Accessibility (ICTA), 2025, pp. 1-4.
- [15] Z. Zhu, S. Du, and Y. Qiu, "Research on the Application of Large Language Models in Programming Tutoring," In Proc. IEEE 16th International Conference on Advanced Infocomm Technology (ICAIT), 2024, pp. 314-318.
- [16] K. Naha, S. Y. Rubaiat, and H. M. Jamil, "Are Large Language Models Smart Enough for SQL Tutoring and Assessment?," In Proc. IEEE International Conference on Advanced Learning Technologies (ICALT), 2025, pp. 357-359.
- [17] Z. D. Rafi, A. Purwarianti, and S. Sempena, "Development of Chatbot Module in an Intelligent Tutoring System for English Language Learning Using Large Language Model," In Proc. 28th Conference Oriental COCOSA (O-COCOSA), 2025, pp. 1-6.
- [18] N. Hinov, P. Stanchev, and R. Kabakchieva, "Large Language Models for Power Electronics Education: Retrieval-Augmented Tutoring with Automatic Verification and Simulation," In Proc. 49th International Spring Seminar on Electronics Technology (ISSE), 2026, pp. 1-6.
- [19] H.-T. Ho, D.-T. Ly, and L. V. Nguyen, "Mitigating Hallucinations in Large Language Models for Educational Application," In Proc. IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), 2024, pp. 1-4.
- [20] H. S. Shafrin Fathima, N. J. Joby, and P. Soni, "Explainable Sentence-Level Self-Correction for Hallucination Mitigation in Medical Question Answering Using Large Language Models," In Proc. IEEE International Conference on Advanced Technologies and Innovations (ICATI), 2026, pp. 1-4.
- [21] M. Pan, D. Kodakandla, and M. Farooque, "Fine-Tuned Large Language Models for Logical Translation: Reducing Hallucinations with Lang2Logic," In Proc. International Symposium on Networks, Computers and Communications (ISNCC), 2025, pp. 1-4.
- [22] Z. Duan and J. Wang, "Enhancing Multi-Agent Consensus through Third-Party LLM Integration: Analyzing Uncertainty and Mitigating Hallucinations in Large Language Models," In Proc. 8th International Conference on Advanced Algorithms and Control Engineering (ICAACE), 2025, pp. 2222-2227.
- [23] E. Joo, Y.-J. Lee, and H.-J. Choi, "Entropy-based Sentence-level Hallucination Score in Large Language Models," In Proc. IEEE International Conference on Big Data and Smart Computing (BigComp), 2025, pp. 77-78.
- [24] S. Arulmohan, M.-J. Meurs, and S. Mosser, "Extracting Domain Models from Textual Requirements in the Era of Large Language Models," In Proc. ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2023, pp. 580-587.
- [25] X. Yan, S. Cheng, and L. Guo, "Bug localization based on syntactical and semantic information of source code," *Journal of Systems Engineering and Electronics*, vol. 34, no. 1, pp. 236-246, Feb. 2023.

- [26] Y. Kim, M. Kim, and E. Lee, "Feature Combination to Alleviate Hubness Problem of Source Code Representation for Bug Localization," In Proc. 27th Asia-Pacific Software Engineering Conference (APSEC), 2020, pp. 511-512.
- [27] H. Liang, L. Sun, M. Wang, and Y. Yang, "Deep Learning with Customized Abstract Syntax Tree for Bug Localization," IEEE Access, vol. 7, pp. 116309-116320, Aug. 2019.
- [28] V. Agrahari, S. Chimalakonda, "AST[AR] - Towards Using Augmented Reality and Abstract Syntax Trees for Teaching Data Structures To Novice Programmers," in Proc. IEEE 20th Int. Conf. Advanced Learning Technologies (ICALT), 2020.
- [29] D. Choi, J. Heo, and E. Lee, "Automated feedback generation for multiple function programs," In Proc. 28th Asia-Pacific Software Engineering Conference (APSEC), 2021, pp. 582-583.
- [30] U. Z. Ahmed, R. Sindhgatta, N. Srivastava, and A. Karkare, "Targeted example generation for compilation errors," In Proc. 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 327-338.
- [31] W. Musser and S. Ghosh, "An extended investigation of large language models employing the Socratic method," In Proc. 17th International Conference on Education Technology and Computers (ICETC), 2025, pp. 257-262.
- [32] S. Zhang, W. Lai, J. Song, X. Yu, X. Liao, and J. Hao, "Scaffolding instruction design research based on zone of proximal development of learning community," In Proc. 7th International Conference of Educational Innovation through Technology (EITT), 2018, pp. 258-262.
- [33] C. Knievel, A. Bernhardt, and C. Bernhardt, "Recognition, retrieval, and response: The AITEE framework for Socratic tutoring in electrical engineering," IEEE Access, vol. 14, pp. 50109-50126, 2026.
- [34] F. V. Kowalski and S. E. Kowalski, "Embedded formative assessment in the undergraduate engineering classroom," In Proc. International Conference of Teaching, Assessment and Learning (TALE), 2014, pp. 457-461.
- [35] E. Bardone, D. Hooshyar, and M. Pedaste, "Theorising the open learner model as an educational tool: An open challenge," In Proc. IEEE 20th International Conference on Advanced Learning Technologies (ICALT), 2020, pp. 274-275.
- [36] L. Shi and A. I. Cristea, "Learners thrive using multifaceted open social learner modeling," IEEE MultiMedia, vol. 23, no. 1, pp. 36-47, Jan.-Mar. 2016.
- [37] A. Dwiarie, A. Nguyen, J. Lämsä, and S. Järvelä, "Personalized support features learners expect from self-regulated learning analytics," In Proc. IEEE International Conference on Advanced Learning Technologies (ICALT), 2023, pp. 66-67.
- [38] B. Meng and F. Yang, "The construction of a visualization-supported online self-regulated learning model," In Proc. 7th International Conference on Computer Science and Technologies in Education (CSTE), 2025, pp. 999-1003.
- [39] E. Ayedoun, Y. Hayashi, and K. Seta, "Toward personalized scaffolding and fading of motivational support in L2 learner–dialogue agent interactions: An exploratory study," IEEE Transactions on Learning Technologies, vol. 13, no. 3, pp. 604-615, Jul.-Sep. 2020.

- [40] A. Ramachandran, A. Litoiu, and B. Scassellati, "Shaping productive help-seeking behavior during robot-child tutoring interactions," In Proc. 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI), 2016, pp. 247-254.
- [41] M. Liu, L. Zhu, J. Wang, and Y. Qian, "The impact of data-driven feedback on programming learning: A cognitive load perspective," In Proc. 7th International Conference on Computer Science and Technologies in Education (CSTE), 2025, pp. 254-258.
- [42] W. Hawkins, N. Heffernan, and R. S. J. d. Baker, "Which is more responsible for boredom in intelligent tutoring systems: Students (trait) or problems (state)?," In Proc. Humaine Association Conference on Affective Computing and Intelligent Interaction (ACII), 2013, pp. 618-623.
- [43] C. V. R. Padmaja and S. Lakshminarayana, "Enhancing language models through prompt engineering — A survey," In Proc. IEEE International Conference on Intelligent Systems, Smart and Green Technologies (ICISSGT), 2024, pp. 117-121.
- [44] J.-G. Harms, P. Kucherbaev, A. Bozzon, and G.-J. Houben, "Approaches for dialog management in conversational agents," IEEE Internet Computing, vol. 23, no. 2, pp. 13-22, Mar.-Apr. 2019.
- [45] S. H. K., A. R. Jineesh, I. D. Raj, S. Kumari, and A. Mylavaram, "Evolution of knowledge tracing models: A comparative analysis of Bayesian, deep learning and transformer-based approaches," In Proc. 13th International Conference on Computing for Sustainable Global Development (INDIACom), 2026.
- [46] B. Yousuf and O. Conlan, "Assessing the impact of controllable open learner models on student engagement," In Proc. IEEE 20th International Conference on Advanced Learning Technologies (ICALT), 2020, pp. 47-49.
- [47] M. T. Kunuku and N. Dehbozorgi, "Multimodal quiz generation via RAG with LLM-as-judge evaluation," In Proc. IEEE Frontiers in Education Conference (FIE), 2025.

ΠΑΡΑΡΤΗΜΑ Α : ΑΠΟΣΠΑΣΜΑΤΑ ΚΩΔΙΚΑ

Στο παράρτημα αυτό παρατίθενται ενδεικτικά αποσπάσματα κώδικα από την υλοποίηση του συστήματος, στα οποία γίνεται αναφορά στο Κεφάλαιο 4.

```
prompt = (
    f"{DEBUGGER_SYSTEM_PROMPT}\n\n"
    f"Εκφώνηση: {current_task}\n"
    f"Κριτήρια: {criteria_text}\n"
    f"Κώδικας μαθητή:\n{student_code}\n\n"
    f"ΔΟΜΙΚΑ ΣΤΟΙΧΕΙΑ (από στατική ανάλυση — δεδομένα, ΟΧΙ γνώμη):\n{flags_lines}\n\n"
    f"ΕΥΡΗΜΑΤΑ ΚΑΝΟΝΩΝ (τι ενεργοποιήθηκε από deterministic ελέγχους βάσει κριτηρίων):\n{findings_lines}\n\n"
)
```

Σχήμα Α.1. Απόσπασμα prompt με ρητή θεμελίωση ευρημάτων (Debugger)

```
def generate_random_task(lesson, difficulty, current_task=None):
    templates_dict = lesson.get("task_templates", {})
    templates = templates_dict.get(difficulty, templates_dict.get("easy", []))
    if not templates:
```

```

return {"task_text": "Γράψε ένα απλό πρόγραμμα Python.",
        "rendered_criteria": lesson.get("success_criteria", [])}

available = templates
if current_task and len(templates) > 1:
    task_prefix = current_task[:30]
    other = [t for t in templates if not t[:30] == task_prefix]
    if other:
        available = other

template = random.choice(available)
possible_values = lesson.get("possible_values", {})
replacements = {}
for key, options in possible_values.items():
    if "{" + key + "}" in template and options:
        replacements[key] = random.choice(options)

task_text = _resolve_placeholders(template, replacements)

```

Σχήμα Α.2. Παραγωγή άσκησης από πρότυπο

```

for _while_node in ast.walk(tree):
    if not isinstance(_while_node, ast.While):
        continue
    _condition_names = {n.id for n in ast.walk(_while_node.test) if isinstance(n, ast.Name)}
    if not _condition_names:
        continue
    _modified_names = set()
    for _stmt in ast.walk(_while_node):
        if isinstance(_stmt, ast.AugAssign) and isinstance(_stmt.target, ast.Name):
            _modified_names.add(_stmt.target.id)
        elif isinstance(_stmt, ast.Assign):
            for _t in _stmt.targets:
                if isinstance(_t, ast.Name):
                    _modified_names.add(_t.id)
    _has_break = any(isinstance(n, ast.Break) for n in ast.walk(_while_node))
    if not (_condition_names & _modified_names) and not _has_break:
        facts.append((
            "Πιθανό ατέρμονο (infinite) loop...",
            "possible_infinite_loop",
        ))
    break

```

Σχήμα Α.3. Έλεγχος πιθανού ατέρμονου βρόχου while

```

def _criterion_passed(criterion: str, flags):
    c = criterion.lower()
    if "συντακ" in c:
        return True
    if "snake_case" in c:
        return flags.get("snake_case_ok", True)
    if "elif" in c or "and/or" in c or "λογικό τελεσ" in c:
        return flags["has_if"] and (flags.get("has_elif") or flags.get("has_boolop"))

```

```
if "len(" in c:  
    return flags.get("has_len_call", False)  
if "καλείται" in c or "κλήση" in c:  
    return flags.get("has_function_call", False)  
if "print" in c or "τύπων" in c:  
    return flags.get("has_reachable_print", flags["has_print"])  
if "ανάθεση" in c or "=" in c:  
    return flags["has_assign"]  
if "if" in c or "δομή" in c:  
    return flags["has_if"]
```

Σχήμα Α.4. Αντιστοίχιση κριτηρίου σε δομικό εύρημα