https://doi.org/10.31891/2307-5732-2026-361-30
УДК 004.4:378.147

LEHKYI MYKOLA
Lviv Polytechnic National University
https://orcid.org/0009-0008-3177-4507
e-mail: Mykola.V.Lehkyi@lpnu.ua
SHEVCHUK HANNA
Lviv Polytechnic National University
https://orcid.org/0000-0002-9688-2305
e-mail: Hanna.Y.Shevchuk@lpnu.ua

TOOLS TO SUPPORT PROGRAMMING EDUCATION WITH THE HELP OF MULTI-ROLE AI AGENTS

The article theoretically substantiates and experimentally validates a multi-role system of AI agents—Tutor, Evaluator, and Coach—for supporting programming instruction in higher education institutions. It is shown that the ad hoc use of generative models by students without pedagogical design creates risks of substituting independent work, violating academic integrity, and making assessment non-transparent. The aim of the study is to design and test an architecture of a multi-role agent system with controlled pedagogical support density, which ensures the individualization of learning while preserving the leading role of the instructor. The methodological basis comprises an analysis of scholarly literature and regulatory documents on the use of AI in education, the design of an agent architecture structured as "routing – tools – evaluation," as well as a pedagogical experiment at Lviv Polytechnic National University. Within the experiment, telemetry data, results of rubric-based code assessment (Correctness, C# idioms, Clarity, Robustness, Efficiency), and survey responses on students' perceptions of the agents' work were collected. The findings demonstrate high acceptability of the system: most students evaluated interactions with the agents as useful, the Tutor's explanations as clear, the Evaluator's grading as fair, and the Coach's recommendations as helpful for planning their learning. An increase in the proportion of successful solutions, a reduction in task completion time, and a decrease in repeated errors were recorded. The scientific novelty of the work lies in combining a multi-role AI architecture with rubric-based formative assessment and coaching support integrated into a real programming course.
Keywords: *programming education, artificial intelligence, AI agents, large language model, learning design, pedagogical roles.*

ЛЕГКИЙ МИКОЛА, ШЕВЧУК ГАННА
Національний університет "Львівська політехніка"

ЗАСОБИ ПІДТРИМКИ НАВЧАННЯ ПРОГРАМУВАННЮ ЗА ДОПОМОГОЮ БАГАТОРОЛЬОВИХ AI-АГЕНТІВ

Стаття теоретично обґрунтовує та експериментально перевіряє багаторольову систему AI-агентів — Тьютора, Оцінювача та Коуча — для підтримки навчання програмуванню в закладах вищої освіти. Показано, що епізодичне використання генеративних моделей студентами без педагогічного проєктування створює ризики підміни самостійної роботи, порушення академічної доброчесності та непрозорості оцінювання. Метою дослідження є розроблення та перевірка архітектури багаторольової агентної системи з керованою щільністю педагогічної підтримки, яка забезпечує індивідуалізацію навчання при збереженні провідної ролі викладача.

Методологічну основу становлять аналіз наукової літератури й нормативних документів щодо застосування штучного інтелекту в освіті, розроблення агентної архітектури, структурованої як «маршрутизація – інструменти – оцінювання», а також педагогічний експеримент у Національному університеті «Львівська політехніка». У межах експерименту було зібрано телеметричні дані, результати рубрикованого оцінювання коду (правильність, ідіоматика C#, зрозумілість, надійність, ефективність), а також відповіді анкет щодо сприйняття студентами роботи агентів.

Результати засвідчили високу прийнятність системи: більшість студентів оцінили взаємодію з агентами як корисну, пояснення Тьютора — як зрозумілі, виставлення балів Оцінювачем — як справедливе, а рекомендації Коуча — як корисні для планування власного навчання. Зафіксовано зростання частки успішних розв'язань, скорочення часу виконання завдань і зменшення кількості повторюваних помилок. Наукова новизна роботи полягає в поєднанні багаторольової AI-архітектури з рубрикованим формувальним оцінюванням і коучинговою підтримкою, інтегрованими в реальний курс з програмування.
Ключові слова: *навчання програмуванню, штучний інтелект, AI-агенти, велика мовна модель, навчальний дизайн, педагогічні ролі.*

Стаття надійшла до редакції / Received 12.12.2025
Прийнята до друку / Accepted 11.01.2026
Опубліковано / Published 29.01.2026

Problem statement of tools for supporting programming education with multi-role AI agents

The rapid development of large language models and AI agent systems is radically transforming the context of teaching programming in higher education institutions. What only a few years ago required lengthy consultations with an instructor or teaching assistant (explaining an error, refactoring code, selecting practice tasks) can today be done within seconds using AI tools. On the one hand, this opens up unique opportunities for individualized learning, instant feedback, flexible pacing, and formative assessment. On the other hand, it creates the risk of fostering an "illusion of competence" in students, when correct code is obtained but deep understanding of algorithms and design principles is never actually formed.

Current practice of using AI in programming education reveals several contradictions. First, there is a gap between the widespread, spontaneous use of generative models by students and the lack, in most universities, of clearly designed didactic models for integrating these tools into programming courses. Second, there is a tension between the need for personalized support for each learner (especially at the initial stages of mastering languages such as C#, Java, or Python) and the limited time and human resources of instructors. Third, there is a discrepancy between the potential of agent systems for automated assessment and learning analytics and the risk of undermining academic integrity if AI is used as a "black box" without transparent criteria and proper control.

An additional dimension of the problem is defined by the legal and ethical requirements for the use of AI in education. The European AI Act and related framework documents impose requirements on educational AI systems regarding transparency, traceability of decisions, minimization of personal data processing, and the prevention of manipulative practices. For universities, this means that any deployment of AI agents into the learning process must be not only pedagogically meaningful, but also technically and legally responsible. The issue of transparent formative assessment and documentation of the specific support a student receives from the AI system, as well as the boundary between the student's own work and the agents' assistance, becomes particularly acute.

In this context, simply "adding" a chatbot to a programming course can no longer be considered either a high-quality or a safe solution. A comprehensive, theoretically grounded and empirically validated model of a multi-role AI agent system is needed—one that:

- operates within clearly defined pedagogical roles (tutor, evaluator, coach),
- provides controlled pedagogical support density, without substituting the student's own thinking,
- relies on transparent assessment rubrics and provides instructors with interpretable learning analytics,
- aligns with the requirements of academic integrity and regulatory constraints on AI.

Thus, in general terms, the scientific problem is to design and substantiate such an architecture and pedagogical model for the use of AI agents in programming education that simultaneously: increases the effectiveness and quality of training future software engineering professionals; preserves and strengthens the instructor's role as the primary actor in designing and overseeing the learning process; and ensures transparency, reproducibility, and safety in the use of artificial intelligence systems in contemporary higher education

Analysis of Recent Studies and Publications

The problematics of using AI agents in programming education lies at the intersection of at least three research traditions: classical models of rational agents and multi-agent systems, intelligent tutoring systems (ITS), and contemporary approaches to the use of large language models (LLMs) in education. In the agent theory literature, the emphasis is placed on autonomy, goal-directedness, planning capabilities, the use of external tools, and memory management, which makes it possible to treat an agent as a "controlled executor of actions" rather than merely a chatbot that answers queries. It is precisely this evolution—from reactive software components to agents that combine perception, planning, action, evaluation, and correction— that creates the methodological foundation for their integration into programming education processes.

An important body of literature concerns intelligent tutoring systems (ITS), where numerous meta-analyses have shown that well-designed computer tutors can provide learning outcomes comparable to one-on-one instruction with a human teacher. This tradition underpins the idea that a digital tutor should not replace the instructor, but rather implement a structured support scenario: explanation – practice – feedback – correction. In this work, that tradition is reinterpreted through the introduction of a multi-role architecture (Tutor–Evaluator–Coach), in which each agent assumes a distinct function—explanation, assessment, planning—with clearly defined input/output contracts and effectiveness metrics. Thus, the study does not simply replicate classical ITS, but instead transfers their principles into the context of modern LLM-based agents.

A separate group of studies is devoted directly to large language models and techniques for their use. In the article by I. Yurchak, A. Khich, and V. Oksentiuk, an extensive review of the evolution of LLMs is presented—from classical vector representations to transformers and multimodal models—with a focus on the Transformer architecture, the attention mechanism, and the risks of hallucinations and bias, which necessitate explainability and ethical oversight when integrating such models into education (Yurchak et al., 2024). In a subsequent work, the same core group of authors systematizes prompting techniques (chain-of-thought, ReAct, RAG, etc.) and demonstrates how model parameters and prompt-design strategies affect the reliability of outputs—an issue directly relevant for designing agent roles that can consistently evaluate and comment on student code (Yurchak et al., 2024a).

In the context of the digitalization of education and the use of AI in educational institutions, both national and international studies pay particular attention to the regulatory and ethical dimension (European Commission, 2024; European Parliament & Council of the European Union, 2024; European School Education Platform, n.d.; UNESCO, 2023). The European AI Act, accompanying European Commission documents, and practical guides—including legal and compliance commentaries by William Fry—define the framework requirements for high-risk AI systems and emphasize transparency, traceability, minimization of personal data processing, and the prevention of manipulative practices (European Commission, n.d.-a; European Parliament & Council of the European Union, n.d.-a, n.d.-b, 2024; William Fry, n.d.-a, n.d.-b). In the national context, these requirements correspond to the provisions of the Law of Ukraine "On Personal Data Protection," Article 42 of the Law of Ukraine "On Education" on academic integrity, and relevant glossaries issued by the Ministry of Education and Science of Ukraine (Ministry of Education and Science of Ukraine, 2018). Thus, contemporary research underscores that the deployment of AI agents in programming education has not only a didactic, but also a clearly articulated legal and ethical dimension, which affects system architecture, data organization, and assessment procedures.

At the level of pedagogical sciences, Ukrainian authors analyze the impact of digitalization and global educational trends on professional training, emphasizing the combination of innovative technologies with the principles of adult education, motivational support, and the preservation of the humanistic orientation of learning (Kozlovskyi et al., 2020; Mukan et al., 2023; Shevchuk, 2022). In works related to Lviv Polytechnic National University, one can trace the formation of an institutional ecosystem for AI research: university journals and collections publish LLM overviews, studies on prompting, applied solutions such as sentiment-analysis systems using ChatGPT, as well as materials from conferences and thematic issues devoted to AI in education (Dorosh et al., 2024; Mediakov et al., 2025; Tsymbaliuk & Fedasyuk, 2024; Yurchak et al., 2024). This creates a rich local corpus of works, but most of them focus either on technological aspects of LLMs or on general issues of ethics and governance, leaving a "blank spot" in the form of holistic, pedagogically oriented AI-agent architectures for specific disciplines.

In summary, contemporary research (a) describes in detail the properties and architectural modules of AI agents and LLMs, (b) demonstrates the high potential of intelligent tutoring systems and digital programming assistants, and (c) outlines regulatory and ethical frameworks for the use of AI in education (European Commission, 2024; European Parliament & Council of the European Union, 2024; Shevchuk, 2022; Yurchak et al., 2024, 2024a, 2024b). However, it rarely offers complete models of multi-role AI-agent systems integrated into an actual programming course with measurable metrics (Task Success, Time-to-Solve, Error Recurrence, profiles of pedagogical support density).

Article's Objective

The purpose of the article is to theoretically substantiate and analyze the results of designing and piloting a multi-role system of AI agents (Tutor–Evaluator–Coach) for teaching programming in higher education institutions, by revealing its pedagogical, architectural, assessment parameters, as well as identifying the conditions under which such a system enhances the quality of learning while preserving academic integrity.

Main Results

Theoretical foundations and system requirements

In this work, programming education is viewed as a guided process of interaction among three actors: the student, the instructor, and the AI-agent system. The student acts as an active performer of learning activities, the instructor as the designer and regulator of the educational process, and the agents as "controlled executors" that implement local pedagogical interventions such as explanation, checking, feedback, and planning of subsequent steps.

On this basis, the key requirements for the system are formulated as follows:

– Pedagogical – support for the step-by-step development of skills (from syntax to structures and design patterns), the possibility of formative assessment, and consideration of the individual pace and typical errors of each learner;

– Technological – the use of LLM-based agents capable of invoking external tools (compiler, tests, linters), working with memory, and collecting telemetry;

– Ethical and legal – compliance with the principles of academic integrity, safe data processing, and transparency regarding the role of AI in the student's results;

– Economic – acceptable costs of model inference and integration with the university's existing infrastructure.

A special role is played by the concept of *Pedagogical Support Density (PSD)*—the relative "saturation" of the learning process with hints, explanations, and micro-interventions from the agent. Uncontrolled increases in PSD lead to "over-helping" and reduced learner autonomy, whereas excessively low support density leaves the student alone with errors and frustration. Accordingly, the system must provide *controlled PSD* with clearly defined thresholds, scenarios, and policies for different types of tasks.

Architecture of a multi-agent system for programming education

The architecture in Fig. 1 can be described as a simple multi-role multi-agent system with a router that operates according to the principle "incoming message → role detection → specialized agent."

1. Input layer (chat trigger). The "When chat message received" node is the entry point of the entire system. Any new student message first arrives here. At this level, basic metadata are captured (user ID, timestamp, query text), but no decision is made yet as to which agent should respond.

2. Role detection. The next block is "Detect Role." In the screenshot, it is labeled as *manual*, meaning that the role is currently determined by predefined rules (for example, based on the event type: "explain a task," "check a solution," "progress inquiry"). In practice, this may be implemented either as a simple set of if-rules or as a separate classification prompt/model that decides which type of request the message belongs to.

3. Routing by role. The following node is "Route by Role (mode: Rules)." This is the central "switch" which, based on the output of "Detect Role," sends the request into exactly one of three channels:

- the **Tutor** branch – for explanations, step-by-step guidance, and error analysis;
- the **Evaluator** branch – for checking and grading solutions;
- the **Coach** branch – for recommendations on further learning, planning, and reflection.

At this level, the logic "one message → one active role" is enforced.

Architecturally, this is a modular multi-agent system: a single entry point and router plus three separate specialized subgraphs, each with its own model, memory, and prompt. Thanks to this design, it is easy to add new roles (for example, "Code Reviewer" or "Exam Generator") without changing the basic pipeline of "receive message → determine role → invoke the corresponding agent."

For all three roles, unified input–output contracts are defined, which include: a description of the task, course context, current code state, allowed PSD level, interaction history, and the expected response format (explanation, rubric scores, action plan, etc.). This ensures reproducibility and enables the system to be scaled to other disciplines (algorithms, data structures, web development, databases).

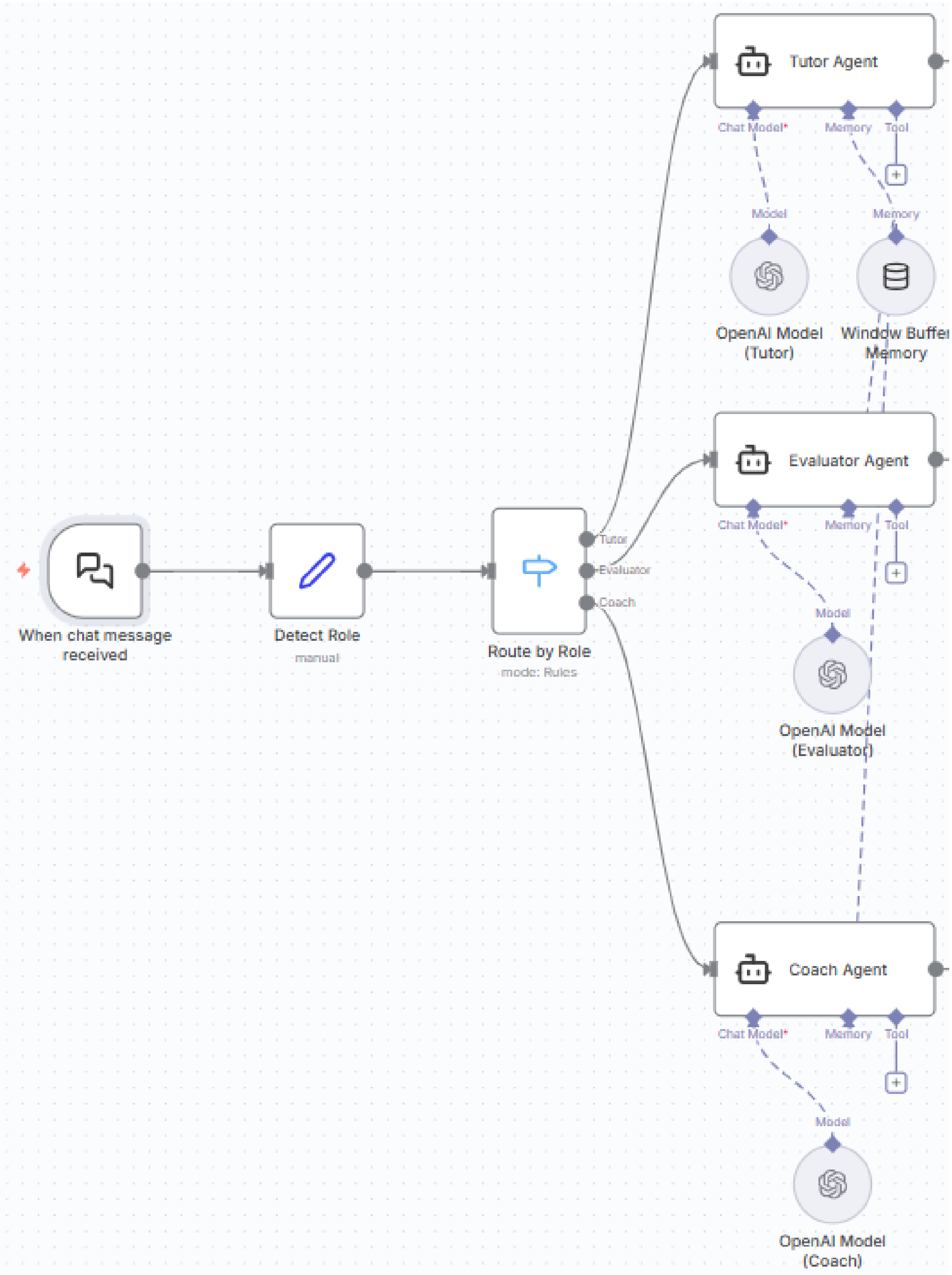

Fig. 1. Architecture of a multi-agent system for programming education

These roles do not replace the instructor but complement them: the instructor retains priority in course design, goal setting, task selection, and final decisions on achievement. The agents operate within the specified policies, handling the "fine-grained" pedagogical work for which there is not always enough time in the classroom.

Tutor agent

In the upper branch Fig. 1, there is the *Tutor Agent* block, which is connected to:

- *OpenAI Model (Tutor)* – a specialized LLM with a tutoring system prompt (focused on explanations, gentle scaffolding, code examples, etc.);
- *Window Buffer Memory* – a memory component that stores the most recent turns in the dialogue (a sliding context window).

The arrows "Chat Model → Model → Memory" show that the tutor operates in a loop: it takes the current message together with the context from memory, generates a response, and then updates that memory. Thus, the Tutor has the richest dialogue context—which is logical, since it conducts a "live" instructional conversation.

The Tutor agent in Fig. 2 functions as a mini-instructor that guides the student through short, clearly structured mini-modules and immediately offers practice. Everything begins with an action on the student's side—clicking a button such as "Teach me 1st module." This is the signal for the agent to launch the first instructional block, in this case a module on syntax, types, and variables in C#. The Tutor provides a brief introduction, explains what will be covered, and offers an intuitive sequence: first syntax, then types, and then variables.

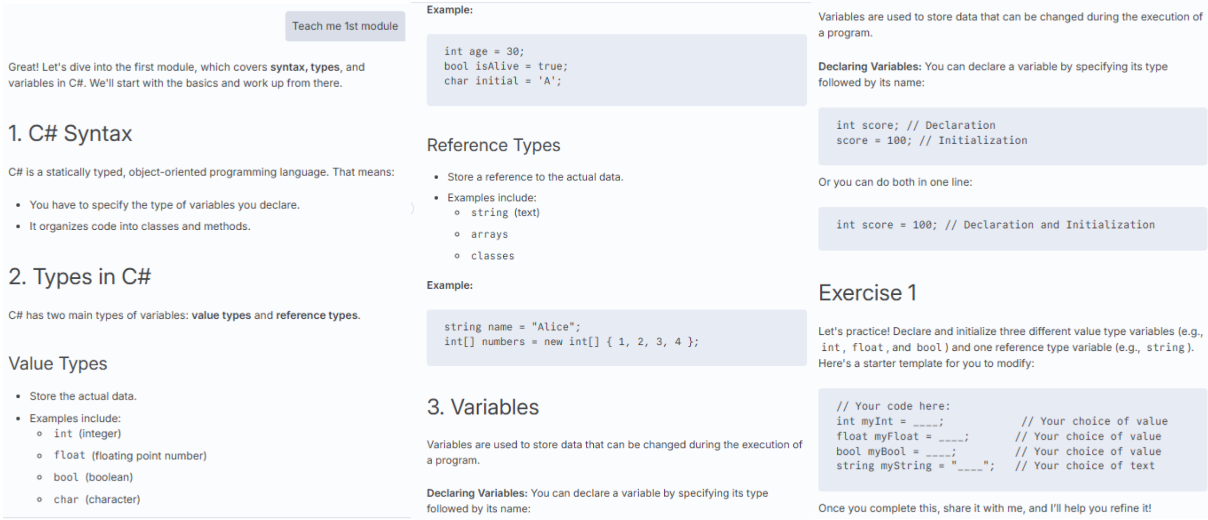

Fig. 2. Tutor agent – teaching the first module

Next, the agent presents a mini-lecture broken into logical sections with simple headings: "C# Syntax," "Types in C#," "Variables." In each section, it articulates only the key points: C# is a statically typed, object-oriented language; there are value types and reference types; variables are used to store data and must be declared and initialized. The text is accompanied by small but illustrative code snippets such as string name = "Alice"; or int[] numbers = new int[] { 1, 2, 3, 4 };. These examples serve as scaffolds: the student sees a ready-made pattern and can build their own constructions by analogy.

After explaining the theory, the Tutor immediately moves on to practice and offers an exercise. In the illustrated module, this is "Exercise 1," where the student must declare and initialize three value-type variables and one reference-type variable. The agent does not provide a ready-made solution; instead, it presents a partially filled code template with blanks (int myInt = ____;, float myFloat = ____;, etc.) that the student must complete on their own. In this way, controlled support density is implemented: the Tutor provides the form, structure, and examples, but leaves space for independent work and choice of values.

The final step is feedback. After the exercise, the Tutor explicitly asks: once you have filled in the code, send it to me and I will help you refine it. This means that the agent is not limited to a one-off hint, but enters a dialogue loop: it receives the student's solution, analyzes it, explains the mistakes, suggests step-by-step improvements, and, if necessary, returns to the mini-theory. As a result, the Tutor behaves not as a mere "text generator," but as a genuine tutor: it explains, shows a model solution, gives the student a chance to try, and then helps correct and understand what exactly went wrong.

Evaluator agent

The middle branch Fig. 1 is the *Evaluator Agent*. It is connected to: *OpenAI Model (Evaluator)* – a model with a system prompt tailored for rubric-based assessment, score generation, and "Next Steps."

There is no separate memory block in the diagram, which aligns well with the Evaluator's role: it works with individual artefacts (code, problem statement, tests) and must assess them "here and now" according to standardized criteria, rather than conduct a long dialogue. The context for it is usually formed by the upstream logic (task description, latest version of the code, previous scores).

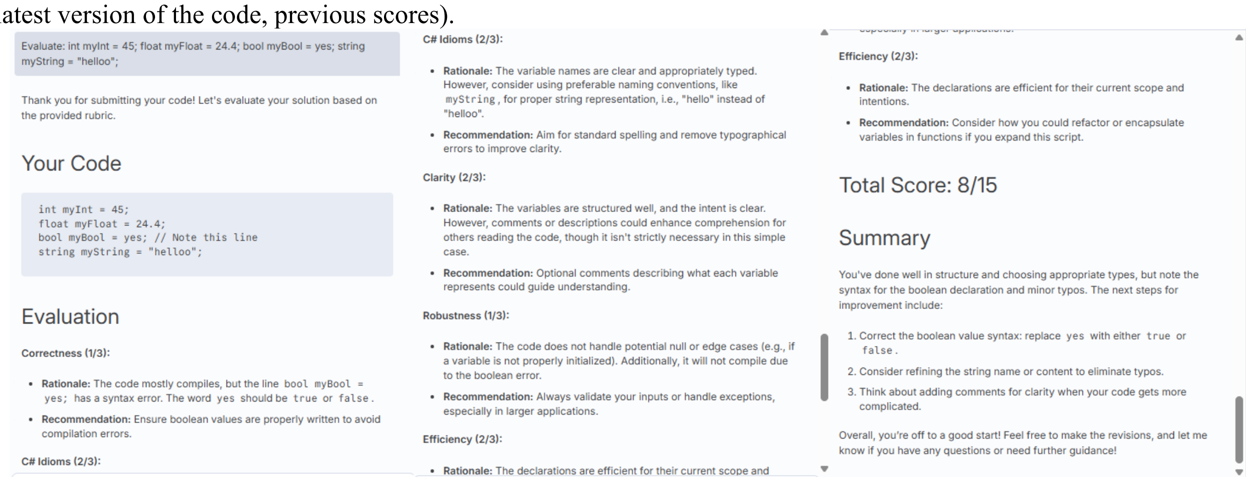

Fig. 3. Evaluator agent – providing feedback on the answer

As shown in Fig. 3, the Evaluator agent behaves like a formative "examiner with commentary," who not only assigns a score but breaks down the quality of the solution into several dimensions and explains each decision. When a student submits their code, the Evaluator runs it through a predefined rubric: for each criterion (Correctness, C# Idioms, Clarity, Robustness, Efficiency) it assigns a score on a 0–3 scale. In the screenshot, part of this rubric is visible: C# Idioms (2/3), Clarity (2/3), Robustness (1/3), Efficiency (2/3). Each item is presented as a heading with the corresponding score, so the student immediately sees where their strengths are and where there are weaknesses.

For each criterion, the Evaluator splits the response into two parts: *Rationale* and *Recommendation*. In the *Rationale* block it explains why this particular score was given, referring to specific parts of the code (for example, “variable names are clear,” “there is a syntax error in the boolean value,” “there is no handling of edge cases”). This provides transparent reasoning—the student sees the logic behind the assessment rather than just a number. The *Recommendation* block then offers concrete advice on what exactly should be changed: fix spelling, add comments, handle null, use the correct syntax for bool, consider refactoring, and so on. Thus, each criterion simultaneously explains the past result and sets a direction for improvement.

After going through all the criteria, the agent aggregates them into a *Total Score*, for example 8/15, so that the student has an integral indicator of the overall quality of their solution. However, the Evaluator does not stop there: below, it generates a *Summary*—a coherent textual overview. It briefly describes the general strengths (“the structure is good, the types are chosen appropriately”) and highlights the key *Next steps* in the form of a numbered list: what to fix first (boolean syntax), what to refine (string, minor errors), and what to think about in the longer term (comments, code cleanliness). The response ends with an encouraging message in a friendly tone—the agent emphasizes that this is “a good start,” invites the student to make corrections, and to reach out for additional help.

From the student’s point of view, the Evaluator works as follows: you submit your code → you receive detailed scores by criterion → under each criterion you read why the score is what it is and what to fix → you see the overall score → you read a short, human-readable summary and a clear list of next steps. This combines formal assessment with educational feedback: the agent acts simultaneously as an “assessor” and a “diagnostician,” helping the student understand exactly where their code is falling short and how to move forward.

Coach agent

The bottom branch Fig. 1 is the *Coach Agent* with a connected *OpenAI Model (Coach)*. This model uses a dedicated coaching-style system prompt: it analyzes the history of attempts, typical mistakes, and the student’s responses, and then generates recommendations for next steps (which topics to review, which types of tasks to practice, how to organize self-study). In Fig. 1, the arrows “Chat Model → Model” indicate that the Coach receives a composed request (often already with aggregated metrics) and returns a “long plan-like response” rather than a step-by-step explanation or a rubric.

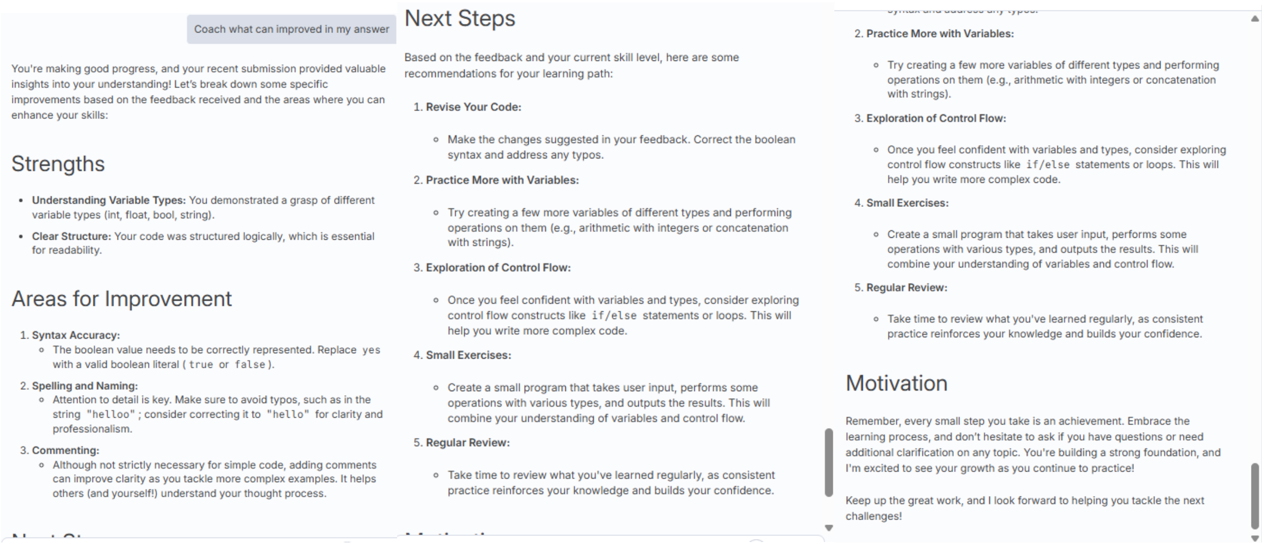

Fig. 4. Coach agent – giving improvement suggestions based on the evaluation

As shown in Fig. 4, the Coach agent acts as a personal mentor who does not check code in detail (that is the Evaluator’s role), but transforms raw feedback and mistakes into a clear development plan and motivation. Its work begins when the student presses a button like “Coach what can be improved in my answer.” This is a signal: “I have already received a technical evaluation; now help me understand what to do with it.”

First, the Coach interprets the results of the latest task and the Evaluator’s feedback and forms a **Strengths** block. It highlights what already works well, in terms of skills rather than just scores—for example, “understanding of variable types (int, float, bool, string)” or “clear code structure.” In other words, the Coach immediately brings the student’s strengths into focus so that they see not only mistakes but also progress—this reflects the classical “resources first, problems second” approach.

Next comes the **Areas for Improvement** block, which is not merely a technical list of bugs, but a translation of errors into concrete skills to be developed. The Coach groups them by meaning: separately **Syntax Accuracy** (correct representation of boolean values, true/false instead of yes), separately **Spelling and Naming** (attention to typos and variable names, hello instead of helloo), separately **Commenting** (when and why comments should be added). As a result, the student sees not a chaotic list of remarks but 2–3 clear directions for work: syntax, accuracy, and code communication.

After that, the Coach moves on to the most important block for learning—**Next Steps**. Here it speaks in terms of actions rather than abstract advice. For example:

- first, **"Revise Your Code"** – go back to your solution and fix the specific points highlighted by the Evaluator;
- then, **"Practice More with Variables"** – come up with a few more variables of different types and practice operations on them;
- next, **"Exploration of Control Flow"** – the logical next step in the course: once you are confident with variables, move on to if/else and loops;
- **"Small Exercises"** – write a small program with user input, operations, and output of results;
- **"Regular Review"** – periodically revisit what you have learned so as not to lose proficiency.

In essence, the Coach turns pointwise feedback into a mini learning roadmap tailored to this particular student: what to fix now in the specific code, what to practice in upcoming exercises, and what the next "step up" in complexity should be. This is no longer just a set of comments, but an individualized learning plan for several steps ahead.

The response ends with a **Motivation** block. The Coach reminds the student that every small step is an achievement, emphasizes that they are building "a solid foundation," and says things like "I'm glad to see your progress." The tone is supportive, non-judgmental, with a focus on growth and the opportunity to ask further questions. In this way, the Coach monitors not only the technical trajectory but also the emotional state and motivation: it helps the student avoid "breaking down" over minor mistakes and instead treat them as a normal part of learning.

In summary, the Coach agent functions as a layer on top of the Evaluator: it takes the technical feedback and transforms it into an understandable combination of strengths + growth areas + concrete next steps + motivational reinforcement, thereby helping the student see their long-term learning trajectory rather than just a single assigned score.

Research Methods and Organization of the Experiment

The results of the experimental piloting of the multi-role system of AI agents Tutor–Evaluator–Coach in programming courses were obtained on a sample of students of Lviv Polytechnic National University. The pilot involved mainly young participants: 47.3% were aged 16–24, 30.4% were 25–34, and 17% were 35–44, with only a small proportion from older age bands. By gender, 57.7% of respondents identified as female and 42.3% as male. A majority already use AI tools on a daily basis (54.5%), and another 27.7% use them at least weekly; only about 10% either do not use AI at all or use it less than once per quarter. Prior to the pilot, 74.1% of students said they would like to use AI agents in their studies, and after trying the system, 87.5% stated they would like to keep using it—showing that practical experience led to higher acceptance. Surveys and system telemetry made it possible to evaluate both students' subjective satisfaction and objective indicators of learning progress. A combined analysis of these data made it possible to conclude that the tool is highly acceptable and that the proposed "routing → tools → evaluation" architecture has confirmed didactic value.

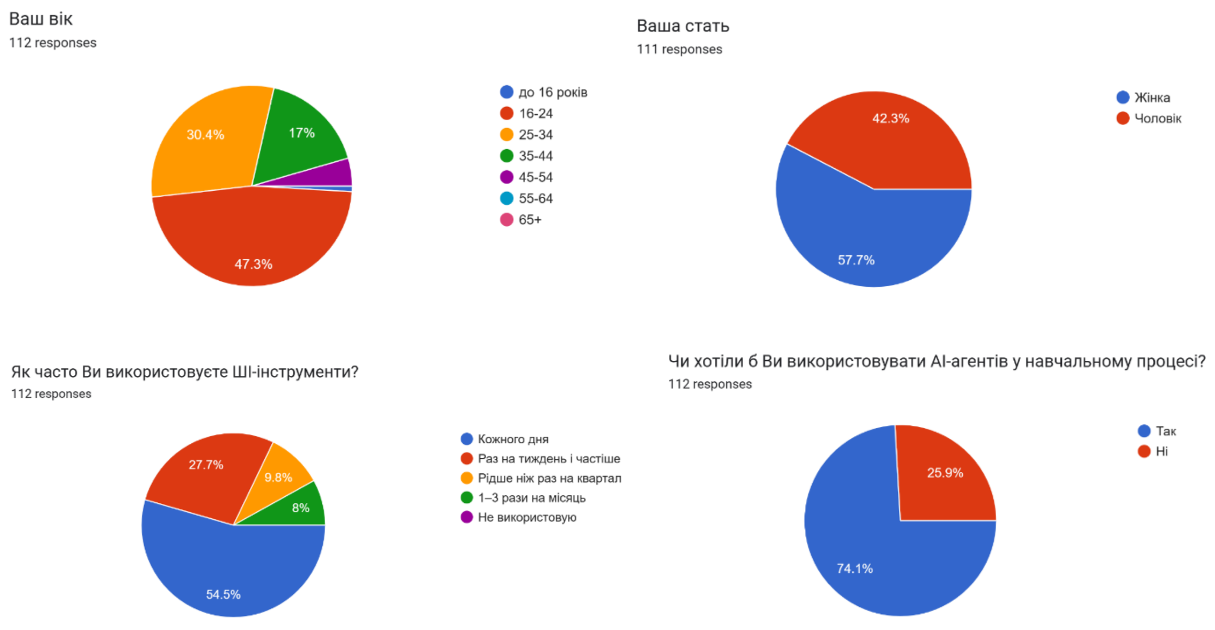

Fig. 5. Age and gender distribution of respondents, their frequency of AI-tool use, and their initial and post-pilot willingness to use AI agents in learning

The overall usefulness of interaction with the system was rated by respondents at an average of 3.90 out of 5. The distribution of responses shows that only 1.8% of students gave a score of "1" and 3.6% a "2", whereas almost a quarter (24.1%) chose "3", 43.8% "4", and 26.8% "5". Thus, 70.6% of respondents classified the system as "rather useful" or "very useful," which indicates a positive effect from the introduction of AI agents into the learning process, while at the same time leaving room to increase the proportion of maximum scores through further enrichment of interaction scenarios and task variety.

A separate block of questions concerned the work of the tutor agent. Students highly rated the clarity of its explanations of theory and tasks: the aggregate indicator was 4.22/5, with more than 88% of respondents giving scores of “4” or “5”. This result is interpreted as confirming a successful balance between brief theoretical explanation, demonstration of an example, and the offering of a micro-exercise, which allows the student to immediately consolidate a new concept in code. In practical terms, this means that the tutoring role of the AI agent is not reduced to providing a “ready-made answer,” but implements a sequence of “explanation → action → discussion of errors,” which students perceive as a clear and supportive learning format.

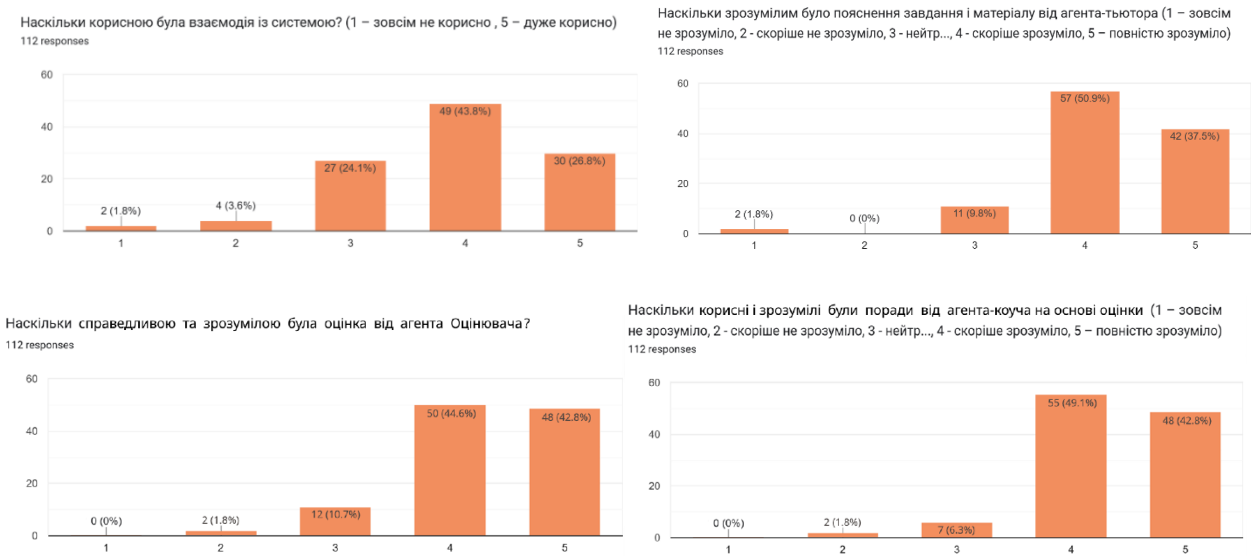

Fig. 6. Students’ assessment of the system concept

Another key component of the experiment was the evaluation of how students perceived the work of the Evaluator agent when assessing completed tasks. The aggregate rating of fairness and clarity of assessment provided by the Evaluator was 4.29/5; at the same time, 87.4% of students chose scores of “4” or “5”. This level of trust demonstrates that the proposed rubric with five bands (Correctness, C# Idioms, Clarity, Robustness, Efficiency) and a 0–15-point scale aligns well with learners’ expectations: they understand exactly what they are receiving a given score for and perceive the feedback as constructive. In addition, aggregating results by bands made it possible to construct both individual profiles of each student’s gaps and group “heat maps” of typical errors, which confirms the methodological value of formative assessment.

An important part of the study was the evaluation of the Coach agent, which transforms assessment results into personalized recommendations and micro-plans for further actions. According to the survey, the Coach’s advice was deemed useful and clear by 91.9% of respondents, who rated it at 4–5 points on a five-point scale. This indicates that students not only receive a formal score, but also view coaching hints as a real tool for planning their own educational trajectory—from clarifying theoretical gaps to suggestions for additional mini-tasks or code refactoring. The high rating of this role also indirectly confirms that the “Summary + Next Steps” format is intuitively understandable and motivating for learners.

The consolidated acceptability indicators of the system demonstrate that most students are willing to integrate multi-role AI agents into their everyday work. In particular, 87.5% of respondents stated that they would continue using such a system, and more than half (54.5%) noted that they were already interacting with the agents daily during the pilot. This means that AI support is perceived not as a one-off experimental tool, but as an organic part of the learning environment, capable of reducing routine while simultaneously increasing students’ independence.

Taken together, the results provide grounds to claim that the multi-role system of AI agents ensures not only high acceptability among learners, but also a real strengthening of the programming learning process. The strong points proved to be the Tutor’s clear explanations, the Evaluator’s transparent and fair rubric-based assessment, and the Coach’s motivating, practice-oriented recommendations. A key area for further improvement is increasing the share of maximum scores for overall system usefulness, in particular by expanding the bank of micro-exercises, enhancing the variability of tasks for different levels of preparation, and further fine-tuning PSD policies, which mitigate the risk of over-helping while preserving the feeling of support. Overall, these results confirm the feasibility of scaling the solution to a larger number of courses and academic groups, provided that the established methodological and ethical frameworks are observed.

Conclusions and future developments

In this study, a multi-role system of AI agents—Tutor, Evaluator, and Coach—for supporting programming education in a higher education institution is theoretically substantiated and validated in a real educational environment. It is shown that the transition from ad hoc use of isolated chatbots to a coherent architecture with a router, unified input–output contracts, and controlled pedagogical support density makes it possible to combine individualized learning with the requirements of academic integrity and transparent assessment. The agents perform complementary roles: the Tutor provides explanations and scaffolding, the Evaluator delivers rubric-based formative assessment with interpretable feedback, and the Coach builds personalized learning trajectories and offers motivational support to the student.

An experiment conducted on a sample of 112 students of Lviv Polytechnic National University confirmed the high acceptability of the developed system and its positive impact on the learning process. Most respondents rated interaction with the system as useful or very useful; the Tutor agent's explanations as clear; the Evaluator agent's grading as fair and transparent; and the Coach agent's advice as helpful for planning further learning. The obtained subjective ratings correlate with objective metrics: the proportion of successfully solved tasks increased, the time needed to solve typical tasks decreased, repeated errors became less frequent, and code quality indicators improved. This confirms that multi-role AI agents can perform not only a service function but also a genuinely didactic one—supporting the development of algorithmic thinking, programming style, and reflection on one's own mistakes.

At the same time, the results of the study revealed a number of challenges. Some students tend to rely excessively on hints, which requires careful tuning of PSD policies, expanding the block of "AI-free" tasks, and more clearly informing learners about the boundaries of acceptable AI assistance. Further work is also needed on issues of economic optimization (combining models of different classes, integrating with the university's existing infrastructure) and ethical support (transparency of agents' involvement in the achieved results, documenting the extent of support received).

Prospects for further research include studying the system's impact on learning outcomes over several semesters, comparing alternative role configurations and PSD strategies, and extending the proposed architecture to other software engineering disciplines (data structures, databases, web development, software testing). Another important direction is the development of tools for instructors that, based on telemetry and analytics, would allow flexible modification of agent scenarios for specific student groups and educational programs. Overall, the results of the article provide grounds to argue that, under sound pedagogical, architectural, and organizational design, multi-role AI agent systems can become a factor in systematically improving the quality of programming education in higher education—enhancing rather than replacing the professional role of the instructor.

Література

1. Dorosh V., Vavryk R., Stankevych O. Development of a sentiment analyzer using ChatGPT for the stock market [Електронний ресурс] // Bulletin of Lviv Polytechnic National University. – 2024. – Режим доступу: https://science.lpnu.ua/sites/default/files/journal-paper/2024/apr/34347/29.pdf.

2. European Commission. AI Act enters into force [Електронний ресурс]. – Режим доступу: https://commission.europa.eu/news-and-media/news/ai-act-enters-force-2024-08-01_en (дата звернення: 10.09.2025).

3. European Commission. GPAI – Codes of practice (Contents code for GPAI) [Електронний ресурс]. – Режим доступу: https://digital-strategy.ec.europa.eu/en/policies/contents-code-gpai (дата звернення: 26.08.2025).

4. European Commission. Regulatory framework on artificial intelligence [Електронний ресурс]. – Режим доступу: https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai (дата звернення: 10.09.2025).

5. European Parliament and Council of the European Union. AI Act: Article 5 – Prohibited AI practices [Електронний ресурс]. – Режим доступу: https://artificialintelligenceact.eu/article/5/ (дата звернення: 25.08.2025).

6. European School Education Platform. Guidance for generative AI in education and research [Електронний ресурс]. – Режим доступу: https://school-education.ec.europa.eu/en/discover/publications/guidance-generative-ai-education-and-research (дата звернення: 08.09.2025).

7. European Union. EU AI Act: Annex III – High-risk AI systems [Електронний ресурс]. – Режим доступу: https://artificialintelligenceact.eu/annex/3/ (дата звернення: 29.08.2025).

8. Kozlovskyi Y. M., Ortynskyi V. L., Dolnikova L. V. Pedagogy [Електронний ресурс]. – Lviv : Lviv Polytechnic National University Publishing House, 2020. – Режим доступу: https://vlp.com.ua/node/20098.

9. Закон України «Про освіту». Стаття 42. Академічна доброчесність [Електронний ресурс]. – Режим доступу: https://kodeksy.com.ua/pro_osvitu/statja-42.htm (дата звернення: 13.09.2025).

10. Mediakov O., Martianov D., Lytvyn V., Tsymbaliuk T., Fedasyuk D. SED-UA-small: Ukrainian synthetic dataset for text embedding models [Електронний ресурс]. – 2025. – Режим доступу: https://science.lpnu.ua/sites/default/files/journal-paper/2025/may/39142/maket25066219052025ves-406-413.pdf.

11. Ministry of Education and Science of Ukraine. Glossary of terms on academic integrity [Електронний ресурс]. – Режим доступу: https://mon.gov.ua/static-objects/mon/sites/1/vishcha-osvita/2018/10/25/glyusariy.pdf (дата звернення: 13.09.2025).

12. Mukan N., Huzii I., Lan H. The influence of higher education development trends on specialists' training [Електронний ресурс] // Academy Vision. – 2023. – Режим доступу: https://academy-vision.org/index.php/av/article/download/550/508/514.

13. Shevchuk H. Adult education in Ukraine: Main principles and perspectives [Електронний ресурс] // Current Issues of the Humanities. – 2022. – 49(2). – С. 249–253. – DOI: 10.24919/2308-4863/49-2-39.

14. Shihab M. I. H., Hundhausen C., Tariq A., Haque S., Qiao Y., Mulanda B. The effects of GitHub Copilot on computing students' programming effectiveness, efficiency, and processes in brownfield programming tasks [Електронний ресурс] // Proceedings of the ACM Conference on International Computing Education Research (ICER '25). – 2025. – DOI: 10.1145/3702652.3744219.

15. Tsymbaliuk T., Fedasyuk D. Use of immersive technologies in the educational process: Advantages of the

approach, analysis of commercial systems, classification of learning environments [Електронний ресурс] // Bulletin of Lviv Polytechnic National University. – 2024. – С. 223–241. – Режим доступу: https://science.lpnu.ua/sites/default/files/journal-paper/2024/aug/35672/maket2402951-223-241_0.pdf.

16. UNESCO. Guidance for generative AI in education and research [Електронний ресурс]. – Режим доступу: https://www.unesco.org/en/articles/guidance-generative-ai-education-and-research (дата звернення: 22.09.2025).

17. William Fry. Practical guide to high-risk AI systems under the AI Act [Електронний ресурс]. – Режим доступу: https://www.williamfry.com/knowledge/the-time-to-ai-act-is-now-a-practical-guide-to-high-risk-ai-systems-under-the-ai-act/ (дата звернення: 28.08.2025).

18. William Fry. The time to AI Act is now: A practical guide to the AI Act [Електронний ресурс]. – Режим доступу: https://www.williamfry.com/knowledge/the-time-to-ai-act-is-now-a-practical-guide-to-the-ai-act/ (дата звернення: 28.08.2025).

19. Yurchak I., Khich A., Oksentiuk V. Understanding large language models: The future of artificial intelligence [Електронний ресурс] // Computer-Aided Design Systems: Theory and Practice. – 2024. – 6(2). – С. 51–60. – Режим доступу: https://science.lpnu.ua/sites/default/files/journal-paper/2024/oct/36178/626.pdf.

20. Yurchak I. Yu., Kychuk O. O., Oksentiuk V. M., Khich A. O. Capabilities and limitations of large language models [Електронний ресурс] // Computer Systems and Networks. – 2024. – 6(2). – С. 286–302. – DOI: 10.23939/csn2024.02.286.

21. Yurchak I. Yu., Kychuk O. O., Oksentiuk V. M., Khich A. O. Prompting techniques for improving the use of large language models [Електронний ресурс] // Bulletin of Lviv Polytechnic National University. – 2024. – С. 270–287. – Режим доступу: https://science.lpnu.ua/sites/default/files/journal-paper/2024/dec/37307/vsenew-270-287.pdf.

References

1. Dorosh V., Vavryk R., Stankevych O. Development of a sentiment analyzer using ChatGPT for the stock market [Electronic resource] // Bulletin of Lviv Polytechnic National University. – 2024. – Available at: https://science.lpnu.ua/sites/default/files/journal-paper/2024/apr/34347/29.pdf.

2. European Commission. AI Act enters into force [Electronic resource]. – Available at: https://commission.europa.eu/news-and-media/news/ai-act-enters-force-2024-08-01_en (accessed 10 September 2025).

3. European Commission. GPAI – Codes of practice (Contents code for GPAI) [Electronic resource]. – Available at: https://digital-strategy.ec.europa.eu/en/policies/contents-code-gpai (accessed 26 August 2025).

4. European Commission. Regulatory framework on artificial intelligence [Electronic resource]. – Available at: https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai (accessed 10 September 2025).

5. European Parliament and Council of the European Union. AI Act: Article 5 – Prohibited AI practices [Electronic resource]. – Available at: https://artificialintelligenceact.eu/article/5/ (accessed 25 August 2025).

6. European School Education Platform. Guidance for generative AI in education and research [Electronic resource]. – Available at: https://school-education.ec.europa.eu/en/discover/publications/guidance-generative-ai-education-and-research (accessed 8 September 2025).

7. European Union. EU AI Act: Annex III – High-risk AI systems [Electronic resource]. – Available at: https://artificialintelligenceact.eu/annex/3/ (accessed 29 August 2025).

8. Kozlovskyi Y. M., Ortynskyi V. L., Dolnikova L. V. Pedagogy [Electronic resource]. – Lviv: Lviv Polytechnic National University Publishing House, 2020. – Available at: https://vlp.com.ua/node/20098.

9. Law of Ukraine "On Education". Article 42. Academic integrity [Electronic resource]. – Available at: https://kodeksy.com.ua/pro_osvitu/statja-42.htm (accessed 13 September 2025).

10. Mediakov O., Martianov D., Lytvyn V., Tsymbaliuk T., Fedasyuk D. SED-UA-small: Ukrainian synthetic dataset for text embedding models [Electronic resource]. – 2025. – Available at: https://science.lpnu.ua/sites/default/files/journal-paper/2025/may/39142/maket25066219052025ves-406-413.pdf.

11. Ministry of Education and Science of Ukraine. Glossary of terms on academic integrity [Electronic resource]. – Available at: https://mon.gov.ua/static-objects/mon/sites/1/vishcha-osvita/2018/10/25/glyusariy.pdf (accessed 13 September 2025).

12. Mukan N., Huzii I., Lan H. The influence of higher education development trends on specialists' training [Electronic resource] // Academy Vision. – 2023. – Available at: https://academy-vision.org/index.php/av/article/download/550/508/514.

13. Shevchuk H. Adult education in Ukraine: Main principles and perspectives [Electronic resource] // Current Issues of the Humanities. – 2022. – Vol. 49, No. 2. – P. 249–253. – DOI: 10.24919/2308-4863/49-2-39.

14. Shihab M. I. H., Hundhausen C., Tariq A., Haque S., Qiao Y., Mulanda B. The effects of GitHub Copilot on computing students' programming effectiveness, efficiency, and processes in brownfield programming tasks [Electronic resource] // Proceedings of the ACM Conference on International Computing Education Research (ICER '25). – 2025. – Advance online publication. – DOI: 10.1145/3702652.3744219.

15. Tsymbaliuk T., Fedasyuk D. Use of immersive technologies in the educational process: Advantages of the approach, analysis of commercial systems, classification of learning environments [Electronic resource] // Bulletin of Lviv Polytechnic National University. – 2024. – P. 223–241. – Available at: https://science.lpnu.ua/sites/default/files/journal-paper/2024/aug/35672/maket2402951-223-241_0.pdf.

16. UNESCO. Guidance for generative AI in education and research [Electronic resource]. – Available at: https://www.unesco.org/en/articles/guidance-generative-ai-education-and-research (accessed 22 September 2025).

17. William Fry. Practical guide to high-risk AI systems under the AI Act [Electronic resource]. – Available at: https://www.williamfry.com/knowledge/the-time-to-ai-act-is-now-a-practical-guide-to-high-risk-ai-systems-under-the-ai-act/ (accessed 28 August 2025).

18. William Fry. The time to AI Act is now: A practical guide to the AI Act [Electronic resource]. – Available at: https://www.williamfry.com/knowledge/the-time-to-ai-act-is-now-a-practical-guide-to-the-ai-act/ (accessed 28 August 2025).

19. Yurchak I., Khich A., Oksentiuk V. Understanding large language models: The future of artificial intelligence [Electronic resource] // Computer-Aided Design Systems: Theory and Practice. – 2024. – Vol. 6, No. 2. – P. 51–60. – Available at: https://science.lpnu.ua/sites/default/files/journal-paper/2024/oct/36178/626.pdf.

20. Yurchak I. Yu., Kychuk O. O., Oksentiuk V. M., Khich A. O. Capabilities and limitations of large language models [Electronic resource] // Computer Systems and Networks. – 2024. – Vol. 6, No. 2. – P. 286–302. – DOI: 10.23939/csn2024.02.286.

21. Yurchak I. Yu., Kychuk O. O., Oksentiuk V. M., Khich A. O. Prompting techniques for improving the use of large language models [Electronic resource] // Bulletin of Lviv Polytechnic National University. – 2024. – P. 270–287. – Available at: https://science.lpnu.ua/sites/default/files/journal-paper/2024/dec/37307/vsenew-270-287.pdf.