ПЕДАГОГІЧНА АКАДЕМІЯ:
НАУКОВІ ЗАПИСКИ

Інформаційно-комунікаційні технології в освіті

УДК 378.147:[004.8+004.42]

DOI https://doi.org/10.5281/zenodo.20376599

МЕТОДИЧНІ ЗАСАДИ ІНТЕГРАЦІЇ ГЕНЕРАТИВНИХ МОВНИХ МОДЕЛЕЙ У ВИКЛАДАННЯ ПРОГРАМУВАННЯ ДЛЯ СТУДЕНТІВ ТЕХНІЧНИХ СПЕЦІАЛЬНОСТЕЙ

Смірнова Оксана Ярославівна

кандидат хімічних наук, кафедра обчислювальної математики та програмування, Національний університет «Львівська політехніка» ORCID: 0000-0002-1314-0489 E-mail: Oksana.y.smirnova@lpnu.ua

Прийнято: 12.04.2026 | Опубліковано: 30.04.2026

Анотація. У статті обґрунтовано диференційований методичний каркас інтеграції генеративних мовних моделей у курси програмування для студентів інженерних, фізико-математичних, енергетичних та прикладно-математичних спеціальностей. Розмежовано базові поняття предметного поля (генеративні мовні моделі, AI-асистенти програмування, інтелектуальні навчальні системи, адаптивні платформи) та сформульовано шкалу інтеграції від епізодичного використання до системного перепроєктування курсу. Проаналізовано трансформацію цілей програмістської підготовки за умов доступності великих мовних моделей: акцент зсувається від синтаксису й шаблонних алгоритмів до декомпозиції задачі, специфікації вимог, інтерпретації згенерованого коду, його верифікації та налагодження. Розгорнуто чотири дидактичні моделі інтеграції: парне програмування з моделлю як драйвером, LLM як опонент у структурованих дебатах, перевернутий клас з AI-згенерованими прикладами, структуровані промпт-задачі з рубрикою якості запиту. Обґрунтовано

необхідність реструктурації системи оцінювання через процесні портфоліо, усний захист коду, метакогнітивні завдання та диференційовані режими доступу до AI у різних модулях курсу. Виявлено специфіку студентів технічних спеціальностей як адресатів інтеграції: ризик передчасної автоматизації перед формуванням алгоритмічної інтуїції та симетричний ризик недооцінки LLM у спеціальностях з інструментальним використанням коду. Розглянуто інституційні умови інтеграції в українській вищій технічній освіті у контексті післявоєнного відновлення й євроінтеграції, зіставлені з рамковими документами Європейської Комісії, ЮНЕСКО та ОЕСР. Запропоновано триступеневу типологію режимів інтеграції (початковий, проміжний, поглиблений), диференційовану за роллю програмування у спеціальності та етапом освітнього циклу.

Ключові слова: генеративні мовні моделі; методика навчання програмування; технічні спеціальності; дидактичні моделі; академічна доброчесність; цифрова трансформація вищої освіти; інженерія запитів; обчислювальне мислення.

METHODOLOGICAL FOUNDATIONS OF INTEGRATING GENERATIVE LANGUAGE MODELS INTO TEACHING PROGRAMMING FOR STUDENTS OF TECHNICAL SPECIALTIES

Oksana Smirnova

PhD in Chemistry, Department of Computational Mathematics and Programming, Lviv Polytechnic National University ORCID: 0000-0002-1314-0489 E-mail: Oksana.y.smirnova@lpnu.ua

Abstract

The article substantiates a differentiated methodological framework for integrating generative language models into programming courses for students of

engineering, physical-mathematical, energy, and applied-mathematical specialties. The basic concepts of the subject field are delineated (generative language models, AI programming assistants, intelligent tutoring systems, adaptive platforms), and a scale of integration is formulated, ranging from episodic use to systemic course redesign. The transformation of programming education goals under the availability of large language models is analyzed: the emphasis shifts from syntax and template algorithms to task decomposition, requirements specification, interpretation of generated code, its verification and debugging. Four didactic integration models are developed: pair programming with the model as driver, LLM as opponent in structured debates, flipped classroom with AI-generated examples, structured prompt-tasks with a query quality rubric. The necessity of restructuring the assessment system through process portfolios, oral defense of code, metacognitive tasks, and differentiated AI access modes in different course modules is justified. The specificity of technical-specialty students as recipients of integration is identified: the risk of premature automation before forming algorithmic intuition, and the symmetric risk of LLM underestimation in specialties with instrumental use of code. The institutional conditions of integration in Ukrainian higher technical education are considered in the context of post-war reconstruction and European integration, compared with framework documents of the European Commission, UNESCO, and OECD. A three-level typology of integration modes (initial, intermediate, and advanced) is proposed, differentiated by the role of programming in the specialty and the stage of the educational cycle.

Keywords: generative language models; programming pedagogy; technical specialties; didactic models; academic integrity; digital transformation of higher education; prompt engineering; computational thinking.

Постановка проблеми

Генеративні мовні моделі, що масово поширилися після оприлюднення ChatGPT наприкінці 2022 року, змінили самі умови, в яких відбувається програмістська підготовка студентів технічних спеціальностей, і поставили

дидактичну спільноту перед питанням, на яке вона ще не має узгодженої відповіді. Емпіричні дослідження зафіксували здатність моделей сімейства GPT, Claude та Gemini успішно розв'язувати завдання вступних курсів програмування на рівні верхнього квартилю студентів, що поставило під сумнів валідність значної частини традиційних навчальних і контрольних форматів. AI-асистенти програмування (Copilot, Cursor та аналогічні) стихійно поширилися у студентському середовищі, випередивши інституційну реакцію закладів вищої освіти, і викладачі опинилися в ситуації, коли політика курсу формується ad hoc, методичні засади відсутні або сформульовані вкрай загально, а академічна спільнота не встигає виробити консенсус навіть щодо базової термінології.

Для студентів технічних спеціальностей (інженерних, фізичних, енергетичних, прикладно-математичних) ситуація має додатковий рівень складності, оскільки програмування для них є не професійною метою, а інструментом моделювання, чисельного розв'язання задач та обробки даних експерименту. Якщо припустити, що методичні рішення, розроблені для підготовки програмістів-професіоналів, можна перенести на інженерні спеціальності без істотних адаптацій, то таке припущення швидко спростовується відмінністю критеріїв якості, профілю ризиків і структури компетентностей. Українська вища технічна освіта стикається з цією проблематикою в обставинах післявоєнного відновлення, нерівномірного доступу студентів до AI-інструментів та регуляторної невизначеності на національному рівні, що додатково загострює потребу в обґрунтованих методичних засадах, здатних витримати як стрес змін, так і нерівність стартових позицій.

Аналіз останніх досліджень і публікацій

Дослідницький дискурс довкола інтеграції генеративних мовних моделей у програмістську освіту умовно розпадається на три тематичні лінії: концептуальну, програмістсько-педагогічну та контекстну (зокрема українську),

кожна з яких представлена впливовими публікаціями у профільних журналах і формує власну оптику погляду на проблему.

Концептуальна лінія репрезентована синтетичними оглядами можливостей і ризиків LLM у вищій освіті загалом. Робота E. Kasneci та колег у журналі *Learning and Individual Differences* (2023) систематизує сприятливі можливості й виклики застосування великих мовних моделей в усіх ланках освіти, закладаючи термінологічну рамку, на яку спирається подальша дискусія. T. Kosar з колегами у *Mathematics* (2024) описують експеримент із інтеграцією ChatGPT у курс програмування для початківців, фіксуючи зміну характеру помилок і темпу засвоєння матеріалу. R. Yilmaz та F. G. Karaoglan Yilmaz у *Computers in Human Behavior: Artificial Humans* (2023) подають емпіричні дані щодо студентського сприйняття ChatGPT як інструмента навчання програмування, виокремлюючи фактори довіри й опору.

Програмістсько-педагогічна лінія сфокусована на конкретних інструментах і навчальних курсах. J. Prather та колеги в *ACM Transactions on Computer-Human Interaction* (2024) проаналізували юзабіліті GitHub Copilot для студентів-початківців, виявивши парадокси дисонансу очікувань, коли інструмент полегшує процес кодування, але ускладнює формування концептуального розуміння. D. Woo з колективом у *British Educational Research Journal* (2024) представили експериментальне дослідження впливу інтервенції з навчання промпт-інженерії на самоефективність і знання студентів про AI. Y.-M. Yan з колегами в *Humanities and Social Sciences Communications* (2025) описали LLM-опосередковане спільне програмування та його вплив на обчислювальне мислення й самоефективність студентів. Контрольовані експерименти у журналі *Computers and Education: Artificial Intelligence* (2025) формулюють критично важливу дисоціацію між продуктивністю розв'язання й глибиною навчальних результатів, показуючи, що AI-підтримка підвищує перше, але не друге. У журналі *Education Sciences* (2025) представлено результати масштабного опитування студентів інженерних спеціальностей з дванадцяти університетів

щодо використання LLM, що дозволяє оцінити патерни поведінки саме технічної аудиторії. Робота в *TechTrends* (2025) формулює перехідну рамку від програмування до промпт-інженерії як нової освітньої парадигми для технічних спеціальностей.

Контекстна лінія охоплює український вимір та інституційну адаптацію. Д. Оніщенко у журналі *Відкрите освітнє е-середовище сучасного університету* (2026) аналізує критерії добору інструментів штучного інтелекту для навчання програмування майбутніх учителів інформатики, формуючи дидактичну систематизацію інструментів. Н. В. Бахмат з колегами у *Віснику Херсонського національного технічного університету* (2024) розглядають використання штучного інтелекту для персоналізації дистанційного навчання студентів спеціальності «Інженерія програмного забезпечення». М. П. Шишкіна та інші у журналі *Імідж сучасного педагога* (2025) описують світовий і український досвід застосування генеративного штучного інтелекту для вдосконалення педагогічних практик, фіксуючи розрив у цифровій компетентності викладацького корпусу.

Окремо виділяються синтетичні роботи у форматі конференційних збірників: дослідження J. Finnie-Ansley з колегами (ACE'22) щодо здатності OpenAI Codex розв'язувати завдання вступного курсу програмування, B. A. Becker та колег (SIGCSE'23), які проблематизують освітні можливості й виклики AI-генерації коду, а також пізніша робота J. Prather та колег (ITiCSE'23), що пропонує синтетичну рамку для навігації революції генеративного AI в комп'ютингу. Незважаючи на численні емпіричні роботи, у літературі бракує узгодженого методичного каркаса, який враховував би специфіку студентів технічних спеціальностей як адресатів інтеграції LLM у їхній відмінності від студентів IT-напрямів, і саме цей розрив дослідження прагне закрити.

Формулювання цілей статті

Метою статті є обґрунтування диференційованого методичного каркаса інтеграції генеративних мовних моделей у курси програмування для студентів

технічних спеціальностей, який враховує роль програмування у спеціальності та етап освітнього циклу. Досягнення мети передбачає розв'язання таких завдань: розмежувати ключові поняття предметного поля та сформулювати шкалу інтеграції; проаналізувати трансформацію навчальних цілей і компетентностей у програмістській освіті за умов доступності LLM; розгорнути дидактичні моделі інтеграції з опертям на принципи скафолдингу, активного й проблемно-орієнтованого навчання; обґрунтувати необхідність реструктурації системи оцінювання та сформулювати позицію щодо політики академічної доброчесності; виявити специфіку студентів технічних спеціальностей як адресатів інтеграції; сформулювати інституційні умови інтеграції в українських ЗВО та запропонувати триступеневу типологію режимів інтеграції.

Виклад основного матеріалу

Методичні засади інтеграції починаються з розмежування понять, що в публічному й навіть фаховому дискурсі часто змішуються, хоча кожен інструмент породжує власні методичні наслідки. Генеративні мовні моделі (GPT-4o, Claude, Gemini, Llama) є універсальними великими моделями, навченими на корпусах природної мови та коду, які генерують текст за ймовірнісним принципом і не мають вбудованої педагогічної цілі [4]; AI-асистенти програмування (GitHub Copilot, Cursor, Codeium, Tabnine) – це похідні продукти, інтегровані в середовища розробки, що працюють на тих самих базових моделях, але оптимізовані під автодоповнення коду та контекст репозиторію [2; 7]; нарешті, інтелектуальні навчальні системи й адаптивні платформи становлять окремий клас рішень з педагогічно сконструйованими сценаріями, моделюванням рівня учня та траєкторіями просування, які можуть використовувати LLM як ядро, але самі по собі є дидактичною надбудовою [4]. Якщо припустити, що ці три категорії взаємозамінні, то можна було б очікувати уніфікованої методики їх застосування, проте емпірика спростовує таке припущення: LLM у браузері функціонує як відкритий партнер у діалозі, Copilot – як невидимий співавтор у редакторі, адаптивна платформа – як структуроване

навчальне середовище з керованими ризиками. Видається обґрунтованим, що методичні засади мають диференціюватися за типом інструмента, а не виводитися загально для «штучного інтелекту в освіті».

Категорія методичних засад позначає принципово вищий рівень узагальнення, ніж конкретні методи чи методика: засади задають аксіоматику (які цілі переслідуються, яка модель учня приймається, які етичні рамки накладаються), методика розгортає операційну послідовність, методи виокремлюють конкретні прийоми, а дидактична модель оформлює цілісну конфігурацію взаємодії викладача, студента й інструмента [17]. Поняття інтеграції розгортається як шкала, що починається з епізодичного звернення до ChatGPT задля пояснення помилки і закінчується системним перепроєктуванням курсу, у якому LLM закладений у формулювання завдань, критерії оцінювання та траєкторії розвитку компетентностей [10; 11]. Принципово важливо розрізняти двох адресатів, оскільки сама природа їхньої майбутньої праці визначає, що для них означатиме «знання програмування». Для студентів IT-напрямів програмування є професійною діяльністю, у якій кодування, архітектурні рішення та робота з кодовими базами становлять зміст професійного життя; для студентів інженерних, фізичних, енергетичних та прикладно-математичних спеціальностей програмування виступає інструментом моделювання, числового розв'язання задач, обробки даних експерименту, де код підпорядкований предметній моделі, а критерій якості – фізична адекватність та обчислювальна ефективність, а не промислова придатність продукту [5; 23]. Розрив у телеології програмістської підготовки задає різні режими інтеграції LLM і пояснює, чому одна універсальна політика для всіх інженерних спеціальностей виявляється методично неспроможною.

Зміни на рівні навчальних цілей і компетентностей є прямим наслідком окресленого понятійного поля, і саме вони визначають, чому стара методика втрачає здатність продукувати очікуваний результат. LLM зсунули центр ваги програмістської освіти з відтворення синтаксичних патернів на верхні рівні

когнітивної піраміди: емпіричні дослідження J. Finnie-Ansley та колег показали, що Codex розв'язує типові завдання вступного курсу програмування краще за середнього студента, потрапляючи у верхній квартиль [1], і пізніші роботи на матеріалі другого курсу підтвердили схожі результати на складніших задачах. Якщо співвіднести цей факт із переглянутою таксономією Блума, то рівень застосування щодо стандартних алгоритмів частково автоматизується, оскільки модель упевнено генерує сортування, ітерації, базові структури даних, тоді як вага рівнів аналізу, оцінювання й створення зростає у міру того, як студент має декомпозувати задачу, специфікувати вимоги, читати чужий код, верифікувати його та інтегрувати у систему [4; 9]. Це дає підстави стверджувати, що з'являються нові базові грамотності, яких раніше у програмах фактично не існувало: інженерія запитів як вміння формалізувати задачу природною мовою з достатньою повнотою контексту, обмежень і прикладів [3; 13]; оцінювання надійності AI-результату через розпізнавання галюцинацій, прихованих неефективностей, помилок у крайніх випадках і потенційних вразливостей [4; 7]; розуміння меж компетентності моделі, що для інженера означає усвідомлення систематичних збоїв LLM на нестандартних чисельних задачах, де потрібна доменна інтуїція [5].

Парадокс полягає в тому, що частина традиційних цілей не лише зберігає, а навіть посилює свою вагу за умов автоматизації нижніх рівнів. Уявлення про обчислювальну складність, інваріанти циклів, числову стійкість, поведінку плаваючої коми залишаються критичними саме тому, що тільки вони дають змогу побачити, де модель іде в небезпечну сторону [9], і так фундаменти не знецінюються, а перетворюються на детектор AI-помилок. Натомість слабшає вага окремих ритуалів (заучування синтаксису, відтворення з пам'яті стандартних алгоритмів, написання коду з білого аркуша як основного навчального продукту), що раніше становили ядро програмістської дисципліни, а тепер виглядають як симуляція компетентності, легко спростовувана секундним запитом до моделі. Звідси виникає методичне напруження, яке курс

мусить тримати у балансі: готувати студента до партнерства з моделлю та одночасно убезпечувати від атрофії базової алгоритмічної інтуїції [22], не вдаючись ні до луддитської заборони, ні до капітуляції перед автоматизацією.

Нові цілі породжують запит на нові дидактичні моделі, і синтез емпіричних робіт 2022–2025 років дає підстави виокремити кілька робочих моделей, що співвідносяться з принципами скафолдингу, активного навчання, проблемно-орієнтованого та конструктивістського підходів. Якщо припустити, що центральна функція курсу зсувається від продукування коду до його критичного оцінювання, то найбільш природним прийомом стає інверсія парного програмування: студент приймає роль навігатора, який формулює задачу та оцінює рішення, а модель грає драйвера, який генерує код, унаслідок чого когнітивне навантаження зміщується з продукування коду на його критичну оцінку [7; 21]. Така інверсія працює лише за умови попереднього навчання студента ролі рев'юера, без чого виникає ефект «менше стресу, кращі бали, те саме навчання», описаний у контрольованих експериментах: AI підвищує продуктивність розв'язання, але не поглиблює концептуальне розуміння [22]. Логіка інверсії природно розгортається в наступну модель, де LLM з оракула перетворюється на спаринг-партнера: студент отримує AI-згенероване рішення і має знайти приховані дефекти, неефективності, контрприклади, що активує метакогніцію, оскільки він мусить артикулювати, чому конкретне рішення невдале [9]; завдання типу «зламай LLM некоректним промптом» додатково оголює межі моделі та формує епістемну скромність щодо неї.

Продовжуючи цю траєкторію, перевернутий клас із LLM-згенерованими прикладами для критичного аналізу пропонує студентові заздалегідь підготовлений викладачем набір AI-розв'язків однієї задачі, що різняться алгоритмом, стилем і коректністю, а сама діяльність на занятті полягає у порівняльному аналізі, що корелює з конструктивістською тезою про побудову знання через зіткнення альтернатив, а не через демонстрацію канону [14]. Для технічних спеціальностей особливо продуктивно ставити в центр такого

зіткнення задачі чисельних методів (інтегрування, розв'язування систем лінійних алгебричних рівнянь, методи скінченних різниць), де AI-варіанти можна оцінювати ще й за стабільністю та похибкою, а не лише за стилем коду. Завершує цей ряд модель структурованих промпт-задач з рубрикою якості запиту, у якій студент здає не лише код, а й послідовність промптів і рефлексію над їх ітераціями, а оцінюється точність специфікації, повнота контексту, реакція на невдалі відповіді [3; 13]; так prompt literacy інституалізується як окрема компетентність, а когнітивна робота студента стає видимою для викладача, що раніше було неможливим за умов «продуктового» оцінювання.

Дидактичні моделі залишаються неповними без переосмислення системи оцінювання, оскільки класичні домашні завдання з кодування втрачають валідність як засіб контролю: LLM розв'язує їх за хвилини й залишає мало слідів автентичної роботи студента [1; 6], а детектори AI-згенерованого коду помиляються в обидва боки настільки часто, що покладати на них основу політики наївно. Альтернативи групуються довкола трьох взаємодоповнюваних ідей, кожна з яких відновлює різний аспект автентичності. Зсув на процес означає портфоліо з історією версій репозиторію, де видно еволюцію розв'язку, відкати, експерименти й проміжні гіпотези, без яких готовий код виглядає як артефакт без походження. Повернення до автентичних форматів живого контакту, втрачених у роки масової цифровізації, передбачає усний захист коду перед викладачем, технологію «поясни рядок за рядком», парне код-рев'ю з викладачем, контрольні в аудиторії з обмеженим або відсутнім доступом до AI [24]. Нарешті, метакогнітивні завдання вимагають від студента пояснити, чому обрано саме цей алгоритм, які альтернативи відкидалися та з яких міркувань, і саме така рефлексія дає викладачеві можливість оцінити не результат, а саме мислення, що його породило.

Політика академічної доброчесності не часто зводиться до дилеми «дозволити чи заборонити». Обидва полюси на практиці виявляються неробочими: тотальна заборона стає правилом, яке ніхто послідовно не виконує

і яке неможливо проконтролювати; повна ж відкритість позбавляє оцінювання сенсу – стає незрозуміло, чию саме роботу ми оцінюємо. Після 2023 року низка університетів вийшла з цього протиставлення через структурований дозвіл з обов'язковою декларацією використання AI [11]. Усередині конкретного курсу, проте, цей дозвіл сам потребує подальшого розрізнення: модуль, на якому студент тільки опановує синтаксис і базові алгоритми, не може працювати в тому ж режимі, що й інтегральний проєкт у фіналі семестру. У таблиці 1 такі режими розписано для кожного типу модуля – разом із відповідними контрольними процедурами та прикладами завдань, що реалізують конкретний режим на практиці.

Таблиця 1

Режими доступу до AI за типами модулів у курсі програмування для технічних спеціальностей

№	Модуль	Режим AI	Як перевіряти	Приклад
1.	Фундаментальний – синтаксис мови, базові алгоритми, чисельні методи.	закритий доступ	1) аудиторна контрольна без AI; 2) усне опитування біля дошки; 3) ручний код у Git із позначками часу комітів.	метод бісекції – реалізувати самостійно; на прикладі f(x)=x3−x−2f(x)=x^3-x-2 f(x)=x3−x−2 пояснити, як відстежується збіжність
2.	Прикладний (типові задачі моделювання за фахом).	структурований дозвіл – обов'язкова декларація використання	портфоліо з версіями + рефлексивний коментар про роль моделі у розв'язанні	рівняння теплопровідності методом скінченних різниць; до решта – журнал промптів і критика згенерованих варіантів
3.	Проєктний – комплексні задачі предметної області, як правило, у командах по 2–3 особи.	вільний, з документуванням взаємодії	усний захист; відкритий доступ викладача до репозиторію з повною історією комітів і гілок; презентація архітектурних рішень.	симуляція механічної системи з кількома ступенями свободи: обґрунтувати алгоритми, задокументувати, де і як залучено AI

ISSN: 2786-9458

4.	Контрольний – підсумкове.	закритий / обмежений	аудиторно без інтернету. Альтернатива – захист готового проєкту перед комісією.	окремі функції чисельного інтегрування + пояснення вибору методу під конкретну фізичну задачу

Джерело: розроблено автором

Методично виправдано не вибирати між забороною та відкритістю на рівні курсу загалом, а диференціювати режими всередині курсу за функціональним призначенням модуля: фундаментальні модулі закриваються від AI, прикладні модулі його допускають за умови декларування, проєктні відкривають його повністю з обов'язковим документуванням, а контрольні відтворюють умови автентичного контакту з предметом без посередників. Така архітектура зберігає валідність оцінювання й одночасно готує студента до реальної професійної ситуації, де AI вже є й уникнути його неможливо.

Специфіка студентів технічних спеціальностей становить окрему групу методичних викликів, які виникають із самого профілю інженерного студента: сильніша математична підготовка, орієнтація на прикладні задачі та програмування як засіб обчислювального моделювання створюють асиметричну картину ризиків і можливостей, відмінну від тієї, що складається на спеціальностях розробки програмного забезпечення. Опитування дванадцяти університетів демонструють, що інженерні студенти сприймають LLM передусім як практичних розв'язувачів проблем, а не як репозиторій знань, і використовують їх щільніше за гуманітарні спеціальності [23], так що для інженера AI природним чином стає продовженням калькулятора, MATLAB-скрипту чи бібліотеки чисельних методів. Однак саме у цьому продовженні криється ризик передчасної автоматизації, гостріший для технічної аудиторії, ніж видається на перший погляд: якщо студент-енергетик ще не сформував інтуїції про стійкість методу Рунге–Кутти, а вже делегує його реалізацію Copilot, то він втрачає той досвід прокляття чисельної помилки, без якого неможливо діагностувати збій моделі у реальній задачі [5; 22], і когнітивне розвантаження

перетворюється на когнітивне заміщення, описане в межах теорії когнітивного навантаження як зсув від конструювання ментальних моделей до споживання готових артефактів [22]. Симетричним за структурою є зворотний ризик: у спеціальностях, де комерційна розробка програмного забезпечення не є метою, ігнорування LLM теж шкодить, оскільки студент-фізик, який відмовляється від AI-асистентів, програє у швидкості обробки експериментальних даних і написанні скриптів симуляційного моделювання, де ефективність роботи з кодом критична, а сам код залишається інструментальним, а не продуктовим. Методична відповідь на цю асиметрію оформлюється як двофазний цикл: спершу ручне освоєння ключових патернів предметного коду з повною заблокованістю AI, потім свідома передача рутинних операцій моделі з фокусуванням студента на верифікації, інтерпретації результатів та інтеграції коду у фізичну модель [21].

Українська вища технічна освіта переживає інтеграцію LLM в обставинах післявоєнного відновлення та євроінтеграційної перспективи, і ці обставини накладаються одна на одну, створюючи стек асиметрій, який методика мусить враховувати. Найочевиднішою є нерівність доступу: частина студентів користується платними версіями GPT-4o чи Claude через VPN, частина має лише обмежений безкоштовний рівень, що породжує нерівність освітніх можливостей усередині однієї академічної групи й перетворює AI-компетентність на функцію економічного капіталу родини, а не педагогічної організації курсу. Глибшим за рівнем є розрив цифрової компетентності викладачів, виявлений внутрішніми обстеженнями: між тими, хто інтегрував LLM у власну дослідницьку й педагогічну практику, і тими, для кого технологія залишається маркером порушення доброчесності, лежить дистанція, яка унеможливлює узгоджену курсову політику навіть у межах однієї кафедри [18; 20]. Найзагальнішим виміром є регуляторна невизначеність: МОН України спільно з Мінцифрою у 2024 році оприлюднили рекомендації щодо відповідального використання штучного інтелекту в ЗВО, проте документ має м'який характер і не задає

обов'язкових стандартів для інженерних освітніх програм [18], залишаючи університети сам на сам із політичним вибором. Якщо зіставити українську ситуацію з рамковими документами Європейської Комісії, ЮНЕСКО та ОЕСР, то стає видно, що Україна рухається у спільному з ЄС напрямі, але з суттєвим відставанням у нормативній конкретиці [10; 11; 12], тоді як спільний знаменник цих документів – людиноцентричність, обов'язковість педагогічного посередництва, прозорість використання AI, рівність доступу – задає вектор, який має бути операціоналізований у внутрішніх політиках українських ЗВО. Видається обґрунтованим, що для українських технічних університетів необхідні три взаємопов'язані інституційні рішення: рамкові політики університетів із чіткою типологією допустимих режимів використання LLM у курсах різного рівня, що знімуть з окремих викладачів тягар політичного вибору; систематична перепідготовка викладачів з акцентом не на технічних характеристиках моделей, а на дидактичних моделях їх застосування, без чого жодна рамкова політика не отримає виконавця; оновлення стандартів вищої освіти для інженерних спеціальностей, які мають включити компетентності prompt literacy, верифікації AI-результатів та етичної рефлексії над автоматизацією як обов'язкові, а не факультативні.

Аналіз дидактичних моделей, систем оцінювання, специфіки технічних спеціальностей та інституційних умов інтеграції збігається в одній фокусній точці, яка дає підстави стверджувати: продуктивна інтеграція LLM не може бути одною універсальною моделлю, а оформлюється як диференційований методичний каркас, побудований на двох змінних. Перша змінна – роль програмування у спеціальності, що задає опозицію інструмент проти професії й визначає, наскільки глибокою має бути партнерська взаємодія з AI на пізніх етапах підготовки. Друга змінна – етап освітнього циклу, який задає опозицію формування фундаменту проти поглиблення сформованих навичок і визначає, наскільки доступним має бути AI на ранніх етапах. Їх перетин породжує три режими інтеграції – початковий, проміжний, поглиблений – кожен з яких

відповідає певній комбінації змінних, а узагальнена структура каркаса подана у рисунку 1.

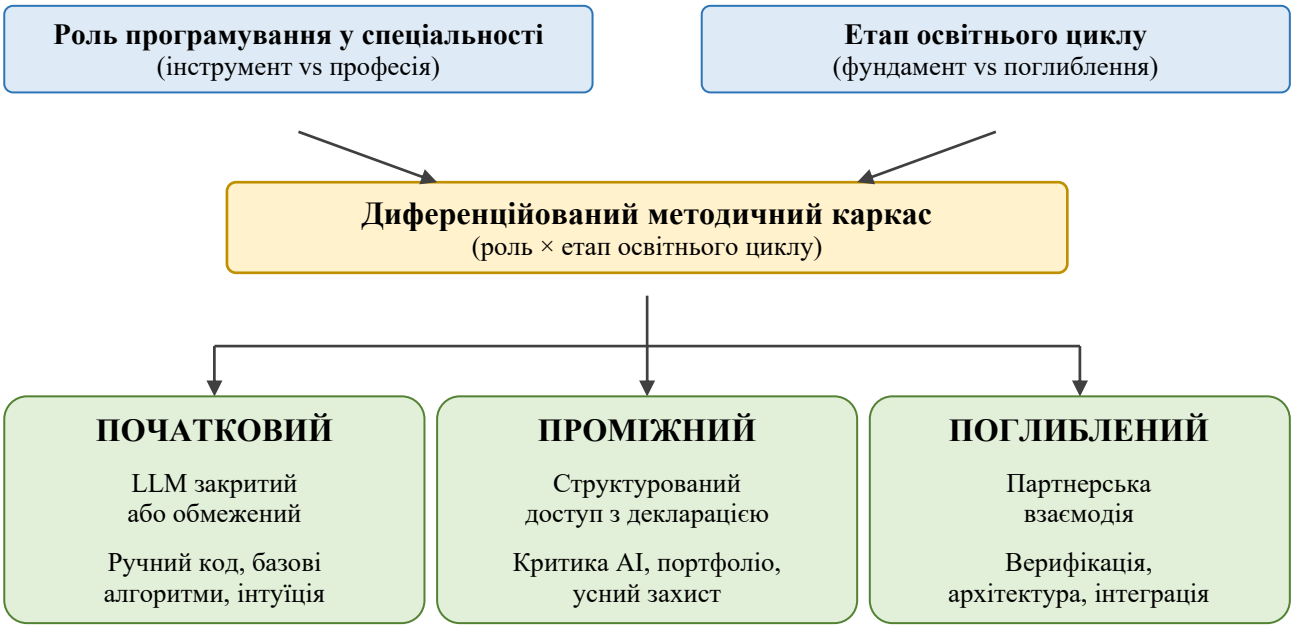

Рис. 1. Триступенева типологія режимів інтеграції LLM у курси програмування для технічних спеціальностей

Джерело: розроблено автором

Подана схема показує, що каркас не пропонує однієї моделі для всіх ситуацій, а оформлюється як простір розв'язків, у якому конкретна навчальна траєкторія читається як шлях через три режими, темп проходження якого визначається роллю програмування у спеціальності та етапом освітнього циклу. Для IT-спеціальностей рух через проміжний до поглибленого режиму відбувається швидше, тоді як для інженерних – повільніше й з довшим утриманням у початковому режимі задля формування алгоритмічної інтуїції перед партнерством з AI.

Висновки

Аналіз дає підстави стверджувати, що інтеграція генеративних мовних моделей у викладання програмування для технічних спеціальностей не може будуватися як одна універсальна модель, а має оформлюватися як диференційований каркас, продуктивна форма якого залежить від двох змінних:

ролі програмування у спеціальності та етапу освітнього циклу. Запропонована типологія включає початковий режим, у якому LLM закритий або жорстко обмежений, акцент покладається на ручний код, мануальне відтворення базових алгоритмів та чисельних методів, формування алгоритмічної інтуїції, і AI з'являється лише як об'єкт критики на демонстраційних прикладах; проміжний режим, який характеризується структурованим доступом з обов'язковим аналізом і критикою AI-результату, рубриками якості промптів, портфоліо з версійною історією, усними захистами, коли студент уже володіє фундаментом і вчиться партнерству з моделлю під педагогічним наглядом; поглиблений режим, що оформлюється як партнерська взаємодія з LLM як норма професійної діяльності, з фокусом на верифікації, інтеграції коду у складніші системи, архітектурних і доменно-специфічних рішеннях, причому для IT-спеціальностей цей режим є режимом старших курсів і магістратури, а для інженерних – режимом спеціалізованих курсів обчислювального моделювання та обробки експериментальних даних.

Філософська рамка каркаса формулюється просто: онтологія навчання у світі LLM перестає бути онтологією індивідуального продукування знання й перетворюється на онтологію критичної співпраці з автоматизованими когнітивними посередниками, а телеологія інженерної освіти зміщується від виховання того, хто пише код, до виховання того, хто розуміє, верифікує та інтегрує код незалежно від його джерела. Йдеться не про зменшення вимог, а про їх перерозподіл, і саме від методичної готовності українських технічних університетів артикулювати цей перерозподіл залежить, чи стане LLM прискорювачем якості інженерної підготовки, чи її непомітним руйнівником.

Перспективи подальших досліджень пов'язані з емпіричною верифікацією запропонованих режимів інтеграції у конкретних курсах обчислювальної математики та програмування для різних інженерних спеціальностей, а також із розробленням методик підготовки викладацького корпусу до системного впровадження LLM у викладання технічних дисциплін.

Список використаних джерел

1. Finnie-Ansley J., Denny P., Becker B. A., Luxton-Reilly A., Prather J. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. Australasian Computing Education Conference (ACE '22). New York : ACM, 2022. P. 10–19. DOI: https://doi.org/10.1145/3511861.3511863

2. Prather J., Reeves B. N., Denny P., Becker B. A., Leinonen J., Luxton-Reilly A., Powell G., Finnie-Ansley J., Santos E. A. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Transactions on Computer-Human Interaction*. 2024. Vol. 31, № 1. Art. 4. DOI: https://doi.org/10.1145/3617367

3. Denny P., Kumar V., Giacaman N. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. Proceedings of the 54th ACM Technical Symposium on Computer Science Education (SIGCSE '23). Toronto : ACM, 2023. P. 1136–1142. DOI: https://doi.org/10.1145/3545945.3569823

4. Kasneci E., Sessler K., Küchemann S. et al. ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education. *Learning and Individual Differences*. 2023. Vol. 103. Art. 102274. DOI: https://doi.org/10.1016/j.lindif.2023.102274

5. Qadir J. Engineering Education in the Era of ChatGPT: Promise and Pitfalls of Generative AI for Education. IEEE Global Engineering Education Conference (EDUCON). Kuwait : IEEE, 2023. P. 1–9. DOI: https://doi.org/10.1109/EDUCON54358.2023.10125121

6. Kosar T., Ostojić D., Liu Y., Mernik M. Computer Science Education in ChatGPT Era: Experiences from an Experiment in a Programming Course for Novice Programmers. *Mathematics*. 2024. Vol. 12, № 5. Art. 629. DOI: https://doi.org/10.3390/math12050629

7. The Effects of GitHub Copilot on Computing Students' Programming Effectiveness, Efficiency, and Processes in Brownfield Programming Tasks. Proceedings of the 21st

ACM Conference on International Computing Education Research (ICER '25). Charlottesville : ACM, 2025. URL: https://arxiv.org/abs/2506.10051

8. Lau S., Guo P. J. From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation. Proceedings of the 2023 ACM Conference on International Computing Education Research. 2023. P. 106–121. DOI: https://doi.org/10.1145/3568813.3600138

9. Becker B. A., Denny P., Finnie-Ansley J., Luxton-Reilly A., Prather J., Santos E. A. Programming Is Hard – Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. Proceedings of the 54th ACM Technical Symposium on Computer Science Education. 2023. P. 500–506. DOI: https://doi.org/10.1145/3545945.3569759

10. Miao F., Holmes W. Guidance for Generative AI in Education and Research. Paris : UNESCO, 2023. 44 p. URL: https://unesdoc.unesco.org/ark:/48223/pf0000386693

11. European Commission, Directorate-General for Education, Youth, Sport and Culture. Ethical Guidelines on the Use of Artificial Intelligence (AI) and Data in Teaching and Learning for Educators. Luxembourg : Publications Office of the European Union, 2022. DOI: https://doi.org/10.2766/153756

12. OECD. Digital Education Outlook 2023: Towards an Effective Digital Education Ecosystem. Paris : OECD Publishing, 2023. DOI: https://doi.org/10.1787/c74f03de-en

13. Woo D. J., Susanto H., Yeung C. H., Guo K., Fung A. K. Y. Effects of a Prompt Engineering Intervention on Undergraduate Students' AI Self-Efficacy, AI Knowledge and Prompt Engineering Ability. *British Educational Research Journal*. 2024. DOI: https://doi.org/10.1002/berj.70087

14. Sarsa S., Denny P., Hellas A., Leinonen J. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. Proceedings of the 2022 ACM Conference on International Computing Education Research. 2022. Vol. 1. P. 27–43. DOI: https://doi.org/10.1145/3501385.3543957

15. Hellas A., Leinonen J., Sarsa S., Koutcheme C., Kujanpää L., Sorva J. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. Proceedings of the 2023 ACM Conference on International Computing Education Research. 2023. DOI: https://doi.org/10.1145/3568813.3600139

16. Yilmaz R., Karaoglan Yilmaz F. G. Augmented Intelligence in Programming Learning: Examining Student Views on the Use of ChatGPT for Programming Learning. *Computers in Human Behavior: Artificial Humans*. 2023. Vol. 1, № 2. Art. 100005. DOI: https://doi.org/10.1016/j.chbah.2023.100005

17. Оніщенко Д. Інструменти штучного інтелекту в навчанні програмування майбутніх учителів інформатики: критерії добору та дидактична систематизація. *Відкрите освітнє е-середовище сучасного університету*. 2026. № 20. С. 112–126. DOI: https://doi.org/10.28925/2414-0325.2026.209

18. Міністерство освіти і науки України, Міністерство цифрової трансформації України. Рекомендації щодо відповідального використання штучного інтелекту в закладах вищої освіти. Київ, 2024. URL: https://mon.gov.ua/news/shtuchnyi-intelekt-u-zakladakh-vyshchoi-osvity-rekomendatsii-dlia-vykladachiv-studentiv-i-pratsivnykiv-zvo

19. Бахмат Н. В. та ін. Використання штучного інтелекту для персоналізації дистанційного навчання студентів спеціальності «Інженерія програмного забезпечення». *Вісник Херсонського національного технічного університету*. 2024. URL: https://journals.kntu.kherson.ua/index.php/visnyk_kntu/article/view/1037

20. Шишкіна М. П. та ін. Використання генеративного штучного інтелекту вчителями для вдосконалення педагогічних практик: світовий та український досвід. *Імідж сучасного педагога*. 2025. URL: https://isp.pano.pl.ua/article/view/340836

21. Yan Y.-M., Chen C.-Q., Hu Y.-B., Ye X.-D. LLM-Based Collaborative Programming: Impact on Students' Computational Thinking and Self-Efficacy. *Humanities and Social Sciences Communications*. 2025. Vol. 12. DOI: https://doi.org/10.1038/s41599-025-04471-1

22. Less Stress, Better Scores, Same Learning: The Dissociation of Performance and Learning in AI-Supported Programming Education. *Computers and Education: Artificial Intelligence*. 2025. URL: https://www.sciencedirect.com/science/article/pii/S2666920X25001778

23. Engineering Students' Use of Large Language Model Tools: An Empirical Study Based on a Survey of Students from 12 Universities. *Education Sciences*. 2025. Vol. 15, № 3. Art. 280. DOI: https://doi.org/10.3390/educsci15030280

24. Ensuring Computer Science Learning in the AI Era: Open Generative AI Policies and Assignment-Driven Written Quizzes. 2025. URL: https://arxiv.org/abs/2601.17024

25. Prather J., Denny P., Leinonen J. et al. The Robots Are Here: Navigating the Generative AI Revolution in Computing Education. Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education. 2023. DOI: https://doi.org/10.1145/3623762.3633499

26. Vadaparty A., Zingaro D., Smith IV D. H. et al. CS1-LLM: Integrating LLMs into CS1 Instruction. Proceedings of the 2024 on Innovation and Technology in Computer Science Education. 2024. Vol. 1. P. 297–303. DOI: https://doi.org/10.1145/3649217.3653584

27. From Programming to Prompting: Developing Computational Thinking through Large Language Model-Based Generative AI. *TechTrends*. 2025. DOI: https://doi.org/10.1007/s11528-025-01052-6