ТРАНСФОРМАЦІЯ НАВЧАЛЬНОГО ПРОЦЕСУ: ІНТЕГРАЦІЯ LLM-АСИСТЕНТІВ ДЛЯ АНАЛІЗУ АРХІТЕКТУРНИХ РІШЕНЬ І РОЗВИТКУ КРИТИЧНОГО МИСЛЕННЯ

Калашник Валерій Юрійович
к.т.н., старший викладач
Київський Національний Університет Технологій та Дизайну

Мельник Геннадій Валерійович
к.т.н., доцент
Київський Національний Університет Технологій та Дизайну

Колиско Оксана Зенонівна
к.т.н., доцент
Київський Національний Університет Технологій та Дизайну

Анотація

У роботі досліджено методику інтеграції технологій **Великих Мовних Моделей (LLM)**, зокрема **GitHub Copilot, Gemini та ChatGPT**, у процес підготовки магістрів комп'ютерних наук. В умовах стрімкої інтеграції Штучного Інтелекту в комерційну розробку, вища освіта стикається з викликом перенесення фокусу навчання з *генерації* коду на його *критичну валідацію*. Представлено **авторську методику «Архітектурного Співпілота»**, де LLM виступає як інструмент для автоматичного формування архітектурних рішень та опису патернів (наприклад, для **Booking Service**). Завдання здобувача полягає у проведенні багатоаспектного **критичного аналізу** згенерованих рішень, що слугує прямим засобом розвитку **критичного мислення (ЗК 6)** та досягнення **Програмних Результатів Навчання (ПРН 10, ПРН 13)**. Обґрунтовано, що така інноваційна методика є ключовим елементом **синергії** між науковими дослідженнями та забезпеченням якості освітніх компонентів.

1.1. Цифрова трансформація та педагогічний виклик

Стрімкий розвиток інструментів на основі Штучного Інтелекту (ШІ) та Великих Мовних Моделей (LLM) — таких як **GitHub Copilot, Gemini та ChatGPT** — трансформував індустріальний процес розробки програмного забезпечення (ПЗ). Для вищої освіти, що готує магістрів комп'ютерних наук, це створює критичний педагогічний виклик. **Пріоритетом стає не здатність до генерації коду, а здатність до верифікації, критичного аналізу та прийняття архітектурних рішень**. Традиційні методи навчання, орієнтовані на написання коду з нуля, вже не відображають реалій індустрії. Проблема полягає у відсутності **систематизованої методики**, яка б гарантовано перетворювала LLM з інструмента для списування на інструмент для **критичного мислення**.

1.2. ПРН як індикатор підтримки якості освітніх компонентів

Швидкість розвитку сучасних технологій тісно пов'язаний з якістю підготовки фахівців. **Програмні Результати Навчання (ПРН)**, визначені Міністерством освіти, є ключовими **стандартами (індикаторами)**, за якими акредитаційні комісії перевіряють, чи освітня програма (ОП) дійсно забезпечує якісний рівень.

Включення ПРН у наше дослідження є **інструментом підтримки якості освітніх матеріалів** (лекцій, практичних робіт). Наша **авторська методика «Архітектурного Співпілота»** слугує **інноваційним інструментом** для цілеспрямованого розвитку цих ПРН: **Проєктування архітектурних рішень (ПРН 10):** Студент вчиться оцінювати LLM-рішення, використовуючи UML/C4 Model. **Оцінка та забезпечення якості ІТ-проєктів (ПРН 13):** Студент валідує LLM-код через Unit/Performance Testing. Цей підхід забезпечує **зв'язок** між інноваційними технологіями та якістю ОП.

Мета статті — обґрунтувати та розробити авторську методику **«Архітектурного Співпілота»**, що інтегрує LLM-асистенти для розвитку критичного мислення при проєктуванні архітектури, забезпечуючи високий рівень відповідності освітніх компонентів індустріальним та академічним стандартам.

2. Теоретичні основи та Авторська Методика «Архітектурного Співпілота»

2.1. Зсув парадигми: Від генерації до критичної валідації в інженерії ПЗ

Інтеграція Великих Мовних Моделей (LLM) у сферу розробки програмного забезпечення (ПЗ) спричинила фундаментальний зсув у парадигмі навчання. Інструменти, такі як **GitHub Copilot, Gemini та ChatGPT**, здатні автоматично генерувати значні обсяги коду, описувати архітектурні патерни (наприклад, Event-Driven Architecture чи Microkernel Architecture) та пропонувати готові конфігурації.

Це трансформує роль інженера-програміста. Як зазначено в класичних працях, архітектура є насамперед про **прийняття рішень**, а не про кодування. Завдання сучасної вищої освіти полягає у розвитку здатності магістрів до **критичної оцінки** рішень LLM. LLM є **статистичним генератором**, який продукує найбільш ймовірні (але не завжди найбільш оптимальні, безпечні чи масштабовані) рішення, ігноруючи специфічний контекст.

Саме тому необхідна методика, яка перетворює LLM з інструменту для автоматизації рутини на **технологічний інструмент для розвитку критичного мислення (ЗК 6)**, що є ключовим індикатором якості освітньої програми.

2.2. ПРН-орієнтований підхід до підтримки якості освітнього компонента

Вища освіта, керується стандартами, визначеними Програмними Результатами Навчання (ПРН). Включення ПРН у наше дослідження є інструментом підтримки якості освітніх компонентів і демонстрацією їхньої актуальності, в таб. 1 наведений приклад їхнього зв'язку.

Таблиця 1

ПРН/Компетенція	Зміст ПРН	Зв'язок із Методикою LLM
ПРН 10	Проєктувати архітектурні рішення інформаційних та комп'ютерних систем.	**Критичний Аудит:** Студент оцінює, наскільки LLM-рішення відповідає вимогам якості (UML, C4 Model).
ПРН 13	Оцінювати та забезпечувати якість IT-проєктів.	**Емпірична Валідація:** Студент перевіряє LLM-код за допомогою модульного та інтеграційного тестування.

Методика «Архітектурного Співпілота» безпосередньо спрямована на досягнення цих ПРН, інтегруючи сучасні технології в систему оцінки якості освіти.

2.3. Авторська Методика «Архітектурного Співпілота»

Методика «Архітектурного Співпілота» розроблена спеціально для курсу «Проєктування архітектури та забезпечення якості програмного забезпечення» і використовує наскрізний кейс-стаді **Booking Service**. Вона складається з трьох ключових, послідовних етапів.

2.3.1. Етап I: Керована Генерація Рішення (LLM Drafting)

На цьому етапі LLM-асистенти (**Gemini, Copilot**) використовуються для швидкої генерації вихідного матеріалу, що економить час, але створює потенційні "пастки" для критичного мислення.

- **Технічна Завдання (Prompt):** Здобувачу ставиться завдання, що вимагає знання архітектурних патернів. Наприклад, "Попросіть **Gemini** згенерувати структуру NestJS-модуля (bookings.module.ts, bookings.controller.ts, bookings.service.ts) для нашої CRUD-логіки Booking Service".
- **Створення Коду з Дефектами:** Оскільки LLM генерує типові, іноді надто спрощені рішення, здобувач отримує код, який, ймовірно, матиме прогалини у валідації або ігноруватиме нефункціональні вимоги. Наприклад, LLM може не інтегрувати кастомний валідатор `isValidBookingDates` або проігнорувати потребу в кластеризації.

2.3.2. Етап II: Багатоаспектний Критичний Аудит (ПРН 10)

Цей етап є центральним для розвитку **критичного мислення (ЗК 6)** і прямо корелює з **ПРН 10 (Проєктування архітектурних рішень)**. Студент повинен провести глибокий аналіз LLM-рішення у трьох вимірах:

1. **Архітектурний Аналіз:** Студент має визначити, який саме архітектурний патерн (Microservices, N-tier, Monolith) закладено LLM, і візуалізувати його за допомогою **C4 Model** (Рівень 2: Діаграма контейнерів) або **UML Deployment Diagram**. Критичний аналіз полягає в обґрунтуванні, чи підходить цей патерн для наскрізного проєкту.

2. **Аналіз Якості (Non-functional Requirements):** Студент ідентифікує, які нефункціональні вимоги (Performance, Security) не були враховані LLM. Наприклад, чи є в рішенні механізми відмовостійкості (Service Discovery, Health

Check).

3. **Аналіз Коду (Clean Code/Design Patterns):** Перевірка коду на відповідність принципам чистого коду та патернам проєктування.

2.3.3. Етап III: Емпірична Валідація Якості (ПРН 13)

Цей етап забезпечує досягнення **ПРН 13 (Оцінка та забезпечення якості IT-проєктів)** шляхом практичного тестування LLM-коду. Це найважливіший крок, що перетворює теоретичний аудит на **інженерну практику**.

1. **Модульне та Інтеграційне Тестування:** Студент використовує знання про Unit Testing для перевірки згенерованого коду. Наприклад, написання тестів для `bookings.service.spec.ts` для верифікації бізнес-логіки, згенерованої LLM. Це виявляє логічні помилки, які LLM часто допускає у складних обчисленнях.

2. **Тестування Продуктивності (Performance Testing):** Студент проводить навантажувальне тестування на базі LLM-рішення, використовуючи інструменти з курсу, щоб кількісно довести, що LLM-рішення має низький RPS або високу затримку (Latency) порівняно з оптимізованими варіантами (наприклад, кластеризованим `main-cluster.ts`). Це змушує студента не просто *вірити* LLM, а *доводити* його недоліки емпірично.

3. Інструментарій та практична імплементація методики

Успішна реалізація методики «Архітектурного Співпілота» вимагає інтеграції двох категорій інструментів: LLM-асистентів як генераторів рішень та індустріальних інструментів для їхньої критичної валідації.

3.1. LLM-асистенти як джерело архітектурних рішень

Для стимулювання критичного мислення здобувачі використовують такі інструменти, як **Gemini, GitHub Copilot та ChatGPT**. Вони виступають як "перший драфт архітектора", який: **Генерує boilerplate-код:** Наприклад, базову структуру NestJS-контролерів та сервісів (`bookings.controller.ts`, `bookings.service.ts`) для Booking Service. **Пропонує архітектурні патерни:** Надає опис та рекомендації щодо використання Microkernel або N-tier архітектури у контексті завдання. **Створює конфігураційні файли:** Наприклад, початковий `docker-compose.yml` для розгортання сервісу. Ключовим моментом є те, що завдання вимагає від студента **цілеспрямовано** шукати "дефекти" у згенерованому рішенні.

3.1. Індустріальне середовище для валідації

Критична перевірка LLM-рішень здійснюється за допомогою інструментів, що вивчаються у курсі, забезпечуючи досягнення **ПРН 13 (Оцінка та забезпечення якості)** наведено в таблиці 2.

Таблиця 2

Засіб Валідації	**Інструменти та Релевантність**	**Мета Валідації**
Контейнеризація	**Docker та Docker Compose**.	Перевірка **розгортання та ізоляції** LLM-рішення. Часто LLM генерує неповну або

		несумісну конфігурацію, що студент має виправити.
Архітектурне Моделювання	**UML (Deployment Diagram) / C4 Model (Container Diagram)**.	Візуалізація та **критичний аудит** LLM-архітектури (ПРН 10). Студент фіксує, чи відповідає LLM-рішення принципам масштабованості.
Тестування Якості	**Jest/Mocha** (Unit Testing), **k6** (Performance Testing).	Емпіричне доведення недоліків LLM. Наприклад, LLM може не врахувати кластеризацію (main-cluster.ts), що призведе до низького RPS навантажувальному тестуванні.

3.3. Практичні аспекти імплементації LLM-методики

Методика інтегрується в практичні та самостійні роботи, перетворюючи їх з рутинного кодування на **інженерний аналіз**. **Створення 'Пасток' від LLM:** Здобувачам дається завдання, у якому LLM часто пропонує помилкове або неоптимальне рішення. Наприклад, LLM може згенерувати Unit Test, який не враховує асинхронність чи edge cases. Завдання студента — **виявити та виправити** ці типові "сліпі зони" ШІ. **Фокус на Валідаторах:** У контексті **Booking Service**, LLM може згенерувати базовий DTO, але ігнорувати складну бізнес-логіку, наприклад, кастомну валідацію дат (`IsValidBookingDates`). Це змушує студента самостійно проєктувати та реалізовувати критично важливі для надійності компоненти (ПРН 13). **Моніторинг:** Студент має обґрунтувати, які метрики (затримка, пропускна здатність) мають бути додані до **Prometheus/Grafana** для моніторингу LLM-рішення в експлуатації. Це розвиває системне мислення, що виходить за межі циклу розробки.

3.3. Результати та освітній вплив

Інтеграція LLM-асистентів у форматі «Архітектурного Співпілота» дозволяє досягти кількох ключових освітніх результатів:

1. **Підвищення ефективності:** Здобувачі витрачають менше часу на синтаксис (який генерує LLM) і більше — на **дизайн, якість та архітектуру**.
2. **Розвиток професійної автономії:** Перехід від кодування за інструкцією до **критичного діалогу** з машиною формує автономного, компетентного фахівця (ЗК 6), що відповідає меті ОП.
3. **Підтвердження якості ОП:** Методика надає **вимірювані результати** для ПРН 10 та ПРН 13, що є вагомим аргументом на користь **якості освітніх компонентів** перед акредитаційними органами.

4. Висновки

Проведене дослідження обґрунтовує необхідність інтеграції технологій **Великих Мовних Моделей (LLM)** у навчальний процес як сучасного інструменту для трансформації вищої освіти.

1. **Парадигмальний Зсув:** Інтеграція LLM-асистентів (**Gemini, Copilot**) у сферу розробки ПЗ спричинила незворотний зсув: фокус інженерної роботи

перемістився з рутинного кодування на **критичний аналіз, валідацію та прийняття рішень**.

2. **Наукова Новація:** Розроблено **авторську методику «Архітектурного Співпілота»**, яка є інноваційною освітньою технологією. Новизна методики полягає у **цілеспрямованому використанні LLM як генератора драфтових рішень**, які студент повинен піддавати систематичному багатоаспектному аудиту.

3. **Розвиток Компетенцій:** Методика прямо слугує інструментом для розвитку **критичного мислення (ЗК 6)** та досягнення ключових **Програмних Результатів Навчання (ПРН)**. Студенти навчаються:

○ **Проєктувати архітектурні рішення (ПРН 10)** через аудит LLM-архітектури (наприклад, оцінка Microkernel чи N-tier патернів).

○ **Оцінювати та забезпечувати якість (ПРН 13)** через емпіричну валідацію LLM-коду за допомогою Unit та Performance Testing.

4. **Підтримка Якості Освітніх Компонентів:** Використання ПРН як індикатора в даній методиці є ключовим механізмом **підтримки якості освітньої програми**. Це демонструє, що навчальні матеріали постійно осучаснюються, забезпечуючи високий рівень підготовки фахівців відповідно до вимог акредитації.

Список літератури

1. The Open Group. (2022). *The Archimate 3.2 Specification*.

2. Lankhorst, M. (2017). *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer.

3. Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

4. Лисенко, В. В. (2024). *Основи розробки та тестування програмного забезпечення*: навчальний посібник. Харків: НТУ «ХПІ».

5. Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice* (3rd ed.). Addison-Wesley.

6. *Проєктування архітектури та забезпечення якості програмного забезпечення* : конспект лекцій / уклад. В. Ю. Калашник ; Київський національний університет технологій та дизайну. – Київ, 2025 – 255 с.

7. *Проєктування архітектури та забезпечення якості програмного забезпечення* : методичні вказівки до практичних робіт / уклад. В. Ю. Калашник ; Київський національний університет технологій та дизайну. – Київ, 2025 – 209 с.

8. Bird, C., et al. (2021). *Machine Learning in Software Engineering*. Morgan & Claypool Publishers.