MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
State University «Kyiv Aviation Institute»
Faculty of Computer Science and Technology

Abstracts of
XX International
conference of higher education students
and young scientists

SEConf.
SOFTWARE ENGINEERING CONFERENCE

SOFTWARE ENGINEERING

Kyiv 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЇ

KAI

Тези доповідей
XX Міжнародної
науково-практичної конференції
здобувачів вищої освіти і молодих вчених

ІПЗ.
ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

Інженерія програмного забезпечення

Київ 2025

UDC 004.4(045)

SEConf. SOFTWARE ENGINEERING CONFERENCE: Abstracts of reports of the XX International Scientific and Practical Conference of Higher Education Graduates and Young Scientists, Kyiv, 2025, State University «Kyiv Aviation Institute», / S. Hnatyuk Editorial Board [etc.]. - K.: SU «KAI», 2025. - 154 p.

The materials of the scientific and practical conference contain a summary of the reports of scientific research works of students of higher education and young scientists in the field of "SOFTWARE ENGINEERING".

Program committee of the conference:

УДК 004.4(045)

ІПЗ. ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: Тези доповідей XX Міжнародної науково-практичної конференції здобувачів вищої освіти і молодих учених, Київ, 2025, Державний університет «Київськиї авіаційний інститут» / Редакційна колегія: С. Гнатюк [та ін.]. – К.: ДУ «КАІ», 2025. – 154 с.

Матеріали науково-практичної конференції містять узагальнення доповідей науково-дослідних робіт здобувачів вищої освіти та молодих учених у галузі «ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ».

Програмний комітет конференції:

Стецюк Петро Іванович, доктор фізико-математичних наук, член-кореспондент НАН України, завідувач відділу Інституту кібернетики імені В.М. Глушкова НАН України, Київ, Україна

Гнатюк Сергій Олександрович, доктор технічних наук, професор, проректор з наукових досліджень та трансферу технологій, Державний університет «Київський авіаційний інститут», Київ, Україна

Савченко Аліна Станіславівна, доктор технічних наук, професор, завідувач кафедри комп'ютерних інформаційних технологій, факультету комп'ютерних наук та технологій, Державний університет «Київський авіаційний інститут», Київ, Україна

Провотар Олександр Іванович, доктор фізико-математичних наук, професор, завідувач кафедри інтелектуальних програмних систем, факультет комп'ютерних наук та кібернетики, Київський національний університет імені Тараса Шевченка, Київ, Україна

Писарчук Олексій Олександрович, доктор технічних наук, професор, професор кафедри обчислювальної техніки, Факультет інформатики та обчислювальної техніки, Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна

Бондарчук Андрій Петрович, доктор технічних наук, професор, професор кафедри штучного інтелекту, Навчально-науковий інститут Інформаційних технологій, Державний університет інформаційно-комунікаційних технологій, Київ, Україна

Чебанюк Олена Вікторівна, доктор технічних наук, доцент, професор кафедри інженерії програмного забезпечення, факультет комп'ютерних наук та технологій, Державний університет «Київський авіаційний інститут», Київ, Україна

Єгорченкова Наталія Юріївна, доктор технічних наук, професор, дослідник департаменту Просторового планування, Інститут менеджменту, Словацький технічний університет в Братиславі, Братислава, Словаччина

Мулеса Оксана Юріївна, доктор технічних наук, професор, професор кафедри фізики, математики і техніки, Пряшівський університет, Пряшів, Словаччина

Кучанський Олександр Юрійович, доктор технічних наук, професор, професор департаменту обчислень та науки про дані, Astana IT University, Астана, Казахстан

Нога Генрик, доктор габілітований, професор, директор інституту технічних наук, завідувач кафедри технічної та комп'ютерної освіти, Університет комісії національної освіти в Кракові, Краків, Польща

Ключевська-Хмелярж Каміла, PhD, заступник директора Інституту технічних наук, Університет комісії національної освіти в Кракові, Краків, Польща

Ковальська Агнєшка, PhD, заступник директора Інституту технічних наук, Університет комісії національної освіти в Кракові, Краків, Польща

Гаєвська Агнєшка, PhD, Університет комісії національної освіти в Кракові, Краків, Польща

Scientific publication

SEConf.
Software Engineering Conference

Abstracts of
XX International
conference of higher education students
and young scientists

Kyiv, 14-16 May 2025
Published in the author's edition

Наукова публікація

ІПЗ.
Інженерія програмного забезпечення

Тези доповідей
XX Міжнародної
науково-практичної конференції здобувачів
вищої освіти і молодих учених

Київ, 14-16 травня 2025
Публікується у авторській редакції

Державний університет «Київський авіаційний інститут» 03058. Київ – 58, проспект Любомира Гузара, 1
Свідоцтво про внесення до Державного реєстру України суб'єктів видавничої справи ДК № 977 від 05.07.2002

ЗМІСТ

УДК 004.421:004.72

МЕТРОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ ПРОГРАМНИХ ПРОЄКТІВ: ОБЄКТНО-ОРІЄНТОВАНИЙ ПІДХІД

Оксана АНДРІЄНКО
Начальник навчально-лабораторного комплексу
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут
Єва Шигимага
Курсант 4 курсу
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут

У роботі розглядається можливість систематизації знань з метрології програмного забезпечення (ПЗ) на основі аналогій з об'єктно-орієнтованим програмуванням (ООП). Визначено основні класи та об'єкти в галузі метрології ПЗ, встановлено відносини між ними з використанням концепцій успадкування, композиції та агрегації, а також описано атрибути та методи кожного класу.

Ключові слова: *метрологія програмного забезпечення, об'єктно-орієнтоване програмування, систематизація знань, успадкування, поліморфізм, інкапсуляція.*

Вступ

Метрологія ПЗ відіграє важливу роль у задачах метрологічного забезпечення проєктів створення програмних продуктів і програмних систем. Однак, на даний момент знання в цій області часто розрізнені та несистематизовані, що ускладнює їх застосування на практиці. Систематизація знань з метрології ПЗ допоможе фахівцям краще розуміти та застосовувати відповідні методи та інструменти, що, своєю чергою, призведе до підвищення якості ПЗ.

Ціль роботи

Метою роботи є систематизація знань з метрології ПЗ на основі аналогій з ООП. Для досягнення цієї мети необхідно:

1. Визначити основні класи та об'єкти в галузі метрології ПЗ.
2. Встановити відносини між цими класами, використовуючи концепції успадкування, композиції та агрегації.
3. Описати атрибути та методи кожного класу, використовуючи відповідну термінологію з метрології ПЗ.

Матеріали та методи

Використання принципів об’єктно-орієнтованого підходу для опису галузі знань «Метрологія програмного забезпечення» базується на формуванні двох класів: «Метрологія» та «Метрологія програмного забезпечення». При цьому клас «Метрологія» є базовим класом, що визначає загальні атрибути та методи для всіх типів вимірювань, а клас «Метрологія програмного забезпечення» - похідний клас, що успадковує атрибути та методи базового класу та додає специфічні для вимірювання характеристик програмного забезпечення.

Для класу «Метрологія ПЗ» можуть бути запропоновані наступні дочірні класи:

1. Метрологія продуктивності ПЗ. Цей клас спеціалізується на вимірюванні та оцінці продуктивності програмного забезпечення, включаючи швидкість виконання, використання ресурсів та інші показники ефективності.

2. Метрологія надійності ПЗ. Цей клас фокусується на вимірюванні та оцінці надійності програмного забезпечення, включаючи такі аспекти, як безвідмовність, стійкість до помилок та здатність до відновлення.
3. Метрологія безпеки ПЗ. Цей клас займається вимірюванням та оцінкою безпеки програмного забезпечення, включаючи аналіз вразливостей, оцінку ризиків та перевірку відповідності стандартам безпеки.
4. Метрологія складності ПЗ. Цей клас спеціалізується на вимірюванні та оцінці складності програмного забезпечення, включаючи аналіз коду, структури та архітектури.
5. Метрологія якості коду ПЗ. Цей клас фокусується на вимірюванні та оцінці якості коду програмного забезпечення, включаючи аналіз стилю кодування, дотримання стандартів та виявлення потенційних проблем.
6. Метрологія людино-машинної взаємодії. Цей клас займається вимірюванням та оцінкою зручності використання програмного забезпечення, включаючи аналіз інтерфейсу користувача, оцінку ефективності та задоволеності користувачів.

Для запропонованого варіанту успадкувань, ієрархія класів виглядатиме так (Рис. 1):

```
classDiagram
    class Метрологія{
        +Об'єкт вимірювання: Фізичні величини
        +Одиниці вимірювання: Стандартизовані одиниці
        +Еталони: Зразки, що відтворюють точні значення одиниці
вимірювання
        +Похибка: Відхилення результату вимірювання від істинного значення
        +Вимірювання()
        +Калібрування()
        +Повірка()
        +Аналіз похибок()
    }
    class Метрологія_програмного_забезпечення{
        +Об'єкт вимірювання: Характеристики програмного забезпечення
        +Одиниці вимірювання: Метрики, специфічні для ПЗ
        +Еталони: Референсні моделі або стандартні зразки коду
        +Похибка: Різниця між очікуваними та фактичними результатами
роботи ПЗ
        +Вимірювання(): Збір та аналіз метрики ПЗ
        +Калібрування(): Налаштування інструментів та середовищ розробки
        +Повірка(): Перевірка відповідності ПЗ стандартам та вимогам
        +Аналіз похибок(): Виявлення, аналіз та усунення дефектів у ПЗ
        +Складність коду: Оцінка складності програмного коду
        +Покриття тестами: Відсоток коду, закритого автоматизованими
тестами
        +Час розробки: Тривалість часу, витраченого на розробку та
тестування ПЗ
        +Статичний аналіз коду(): Аналіз вихідного коду без його виконання
        +Динамічний аналіз коду(): Аналіз роботи ПЗ під час його виконання
        +Оцінка відповідності стандартам(): Перевірка ПЗ відповідності
стандартам
        +Моніторинг продуктивності(): Безперервне відстеження та аналіз
продуктивності ПЗ
    }
    Метрологія <|-- Метрологія_програмного_забезпечення
```

Рис. 1 – Фрагмент коду з описом класів

Цей фрагмент коду демонструє, як можна реалізувати ієрархію класів для метрології, використовуючи принципи об'єктно-орієнтованого підходу.

Таким чином, ієрархія класів матиме такий вигляд (Рис.2):

```
Метрологія
    └── Метрологія_програмного_забезпечення
        ├── Метрологія_продуктивності_ПЗ
        ├── Метрологія_надійності_ПЗ
        ├── Метрологія_безпеки_ПЗ
        ├── Метрологія_складності_ПЗ
        ├── Метрологія_якості_коду_ПЗ
        └── Метрологія_людино_машинної_взаємодії_ПЗ
```

Рис. 2 – Ієрархія класів

Висновки

Систематизація знань з метрології ПЗ є важливим завданням, яке допоможе підвищити ефективність застосування методів цієї галузі на практиці. Використання аналогій з ООП є перспективним підходом до розв'язання цієї проблеми.

Список використаних джерел

1. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

2. Fenton, N. E., & Bieman, J. M. (2014). Software metrics: A rigorous and practical approach. CRC press.

3. Kan, S. H. (2002). Metrics and models in software quality engineering. Addison-Wesley Professional.

Відомості про авторів:

Андрієнко Оксана Володимирівна – начальник навчально-лабораторного комплексу, Військовий інститут телекомунікацій та інформатизації імені Героїв Крут. *Наукові інтереси:* метрологічні аспекти вимірювання ПЗ.

E-mail: oksana020180@gmail.com

Шигимага Єва Русланівна – курсант 4 курсу, Військовий інститут телекомунікацій та інформатизації імені Героїв Крут. *Наукові інтереси:* розробка надійного ПЗ для засобів телекомунікацій.

E-mail: evasigimaga@gmail.com

УДК 004.414.23

ПІДВИЩЕННЯ ЯКОСТІ КОРИСТУВАЦЬКОГО ДОСВІДУ ЗА ДОПОМОГОЮ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ

Юрій БАЖАН
Аспірант кафедри ІПЗ
Державного університету інформаційно-комунікаційних технологій
Науковий консультант к.т.н., доц., доцент кафедри ІПЗ ДУІКТ
Оксана Анатоліївна Золотухіна

У роботі досліджено застосування методів штучного інтелекту для підвищення якості користувацького досвіду (UX) шляхом автоматизації тестування інтерфейсів (UI). Результати показують зменшення часу тестування та підвищення точності виявлення проблем юзабіліті. Також проаналізовано сегментацію користувачів і персоналізацію інтерфейсів. Визначено обмеження, пов'язані з якістю даних та етичними аспектами.

Ключові слова: *штучний інтелект, користувацький досвід, UX, UI, тестування, машинне навчання, юзабіліті, автоматизація, нейронні мережі, адаптивні інтерфейси.*

Вступ

Сучасна інженерія програмного забезпечення зосереджена на створенні програмних продуктів, які не лише виконують функціональні вимоги, а й забезпечують високий рівень користувацького досвіду (UX). Якість UX визначається зручністю, інтуїтивністю та адаптивністю інтерфейсів користувача (UI), що безпосередньо впливає на задоволеність користувачів і конкурентоспроможність продукту. Зі зростанням складності програмних систем традиційні методи тестування UI/UX стають менш ефективними через їх трудомісткість, суб'єктивність оцінок і високі витрати часу [1].

Методи штучного інтелекту (ШІ) відкривають нові перспективи для автоматизації аналізу інтерфейсів, прогнозування поведінки користувачів і персоналізації UX. Такі підходи дозволяють значно скоротити час тестування, зменшити кількість пропущених дефектів і підвищити точність оцінки юзабіліті, що є критичним для сучасних цифрових продуктів [2]. Актуальність цього дослідження зумовлена необхідністю швидкого реагування на зміни в поведінці користувачів, забезпечення високих стандартів якості та конкурентоспроможності програмних продуктів на глобальному ринку.

Ціль роботи

Метою дослідження є розробка підходу до підвищення якості користувацького досвіду через інтеграцію методів штучного інтелекту у процеси тестування та оптимізації UI/UX, аналіз ефективності методів штучного інтелекту у тестуванні UI/UX та оцінка впливу автоматизації на задоволеність користувачів.

Матеріали та методи

Дослідження проводилося на основі тестування UI/UX із застосуванням методів штучного інтелекту та стандартів ISO 9241-210 [3]. Використано набір даних, таких як: час виконання тестування, частоти помилок і суб'єктивні оцінки зручності. Для обробки даних застосовано методи машинного навчання, зокрема кластеризацію (алгоритм K-means) для сегментації користувачів за поведінковими патернами та нейронні мережі для прогнозування проблем юзабіліті. Тестування відбувалося на п'яти вебдодатках.

Результати та обговорення

Отримані результати дослідження підтверджують ефективність використання методів штучного інтелекту для тестування UI/UX. Зафіксовано, що застосування ШІ дозволяє скоротити середній час тестування на 35–45% порівняно з традиційними методами, завдяки автоматизації процесів виявлення дефектів [4]. ШІ виявив 92% проблем юзабіліті, таких як низька контрастність елементів, незручне розташування кнопок та неочевидність функціональних компонентів, тоді як традиційні підходи ідентифікували лише 78% таких проблем.

Додатковий аналіз поведінкових патернів користувачів з використанням ШІ дозволив сегментувати аудиторію на три основні групи: новачки, досвідчені користувачі та експерти, що забезпечило можливість адаптації інтерфейсів до специфічних потреб кожної групи. Це призвело до підвищення загальної задоволеності користувачів на 18%.

Однак, слід відзначити залежність таких моделей від обсягів і якості вхідних даних для навчання, що може стати критичним фактором при роботі з обмеженими вибірками. Крім того, питання етичності обробки персональних даних користувачів залишається відкритим і вимагає подальшого аналізу.

Висновки

Інтеграція методів ШІ у процеси UI/UX інженерії суттєво підвищує якість користувацького досвіду за рахунок автоматизації тестування, персоналізації інтерфейсів та оптимізації ітерацій розробки. Запропонований підхід демонструє значний потенціал для створення адаптивних та конкурентоспроможних програмних продуктів. Перспективні напрями подальших досліджень включають вдосконалення алгоритмів ШІ для ефективної роботи з обмеженими наборами даних та інтеграцію з технологіями доповненої реальності для покращення точності і персоналізації користувацького досвіду.

Список використаних джерел

1. Nielsen, J. (2019). Usability Engineering. Morgan Kaufmann.
2. Shneiderman, B., & Plaisant, C. (2022). Designing the User Interface: Strategies for Effective Human-Computer Interaction. Pearson.
3. ISO 9241-210:2019. Ergonomics of human-system interaction. Geneva: ISO, 2019.
4. Patel, H., & Singh, A. (2020). Artificial Intelligence in User Experience Testing. Journal of Digital Transformation, 6(3), 112-121.

Відомості про автора:

Бажан Юрій Павлович – аспірант кафедри інженерії програмного забезпечення Державному університеті інформаційно-комунікаційних технологій. *Наукові інтереси*: машинне навчання, юзабіліті, автоматизація, нейронні мережі, адаптивні інтерфейси.

E-mail: iurii.bazhan@gmail.com

УДК 004.75

ОГЛЯД СУЧАСНИХ ПРОТОКОЛІВ УПРАВЛІННЯ ТРАНЗАКЦІЯМИ

Андрій БЕХ
Аспірант 1 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий консультант к.ф.-м.н., с.н.с., доцент кафедри ІПЗ ФКНТ
Михайло Вікторович Оленін

У дослідженні розглядаються сучасні протоколи управління транзакціями (двофазний коміт та патерн Saga) для розподілених інформаційних систем транспортного обслуговування з визначенням їх переваг і недоліків.

Ключові слова: *управління транзакціями, двофазний коміт, патерн Saga, розподілені інформаційні системи, транспортне обслуговування.*

Вступ

Актуальність дослідження сучасних протоколів управління транзакціями у розподілених інформаційних системах транспортного обслуговування зумовлена стрімким розвитком цифрових технологій, зростанням обсягів оброблюваних даних та високими вимогами до оперативності, масштабованості й надійності таких систем. Сучасні транспортні інформаційні системи характеризуються складною багатокомпонентною архітектурою, що вимагає ефективних методів управління транзакціями для забезпечення узгодженості, цілісності та швидкодії операцій.

Як свідчать останні дослідження [1 - 3], серед найбільш поширених протоколів можна виділити двофазний коміт (2PC), що забезпечує строгі гарантії узгодженості даних, та патерн Saga, який орієнтований на високу масштабованість завдяки розбиттю транзакцій на послідовність локальних операцій з механізмами компенсації. Водночас кожен із зазначених підходів має свої недоліки: протокол 2PC характеризується високою латентністю та ризиком одиночних точок відмови, а патерн Saga – недостатньою ізоляцією транзакцій, що може впливати на цілісність даних у критичних ситуаціях.

Таким чином, вибір оптимального протоколу управління транзакціями для транспортних інформаційних систем залишається актуальним і потребує подальшого детального аналізу та порівняльного дослідження.

Ціль роботи

Метою дослідження є комплексний аналіз сучасних протоколів управління транзакціями з порівнянням їх характеристик, ефективності та застосовності у розподілених інформаційних системах транспортного обслуговування. Завданням є визначення сильних і слабких сторін кожного протоколу, аналіз методів забезпечення ізоляції та узгодженості даних, а також оцінка їх впливу на загальну надійність систем..

Матеріали та методи

Об'єктом дослідження є сучасні протоколи управління транзакціями, що використовуються в розподілених інформаційних системах транспортного обслуговування, а саме класичний двофазний коміт (Two-Phase Commit, 2PC) та патерн Saga. Вибір цих об’єктів зумовлений їх широким застосуванням у сучасних розподілених системах та

наявністю суттєвих відмінностей у механізмах забезпечення транзакційної цілісності й ізоляції.

Для аналізу використано метод порівняльного аналізу, який полягає у детальному зіставленні протоколів за критеріями продуктивності, відмовостійкості, латентності, ізоляції транзакцій та масштабованості. Порівняльний аналіз дозволяє виявити переваги та недоліки кожного з протоколів та визначити їх доцільність для використання в умовах транспортних інформаційних систем, що характеризуються високими вимогами до оперативності й надійності.

Для наукового обґрунтування та узагальнення результатів дослідження було використано статистичний метод, а саме аналіз емпіричних даних з сучасних наукових джерел [1–3], де наведено кількісні показники ефективності протоколів управління транзакціями. Використання цього методу дало можливість сформулювати узагальнені висновки про продуктивність та застосовність зазначених протоколів у конкретних умовах розподілених інформаційних систем транспортного обслуговування.

Результати та обговорення

Проведений порівняльний аналіз сучасних протоколів управління транзакціями дозволив визначити ключові особливості досліджуваних підходів: класичного двофазного коміту (2PC) та патерну Saga.

Протокол двофазного коміту (2PC) базується на чітко визначеній двофазній процедурі узгодження транзакції між усіма її учасниками. Перша фаза — підготовча, коли координатор очікує готовність усіх учасників до виконання транзакції, друга — коміту, коли транзакція або підтверджується, або скасовується залежно від відповідей учасників. Головною перевагою 2PC є високий рівень узгодженості та ізоляції транзакцій, що робить його придатним для систем, де коректність і цілісність даних є абсолютним пріоритетом. Однак, основними недоліками є висока латентність через синхронізацію учасників, а також ризик виникнення «одиночної точки відмови» через централізовану роль координатора [1].

Патерн Saga демонструє протилежний підхід, розподіляючи складну транзакцію на низку послідовних локальних операцій, кожна з яких супроводжується компенсаційною дією у випадку збою [1, 3]. Основні переваги Saga — висока масштабованість, низька латентність та добра відмовостійкість, оскільки не існує єдиної точки відмови. Проте суттєвим обмеженням класичного патерну Saga є недостатня ізоляція транзакцій, що створює ризики взаємного впливу паралельних транзакцій і часткової втрати узгодженості даних у системі [1]. Основні характеристики протоколів узагальнено у таблиці 1.

Таблиця 1

Порівняльна характеристика протоколів управління транзакціями

Характеристика	Двофазний коміт (2PC)	Патерн Saga
Узгодженість та ізоляція	висока	середня
Латентність	висока	низька
Масштабованість	низька	висока
Відмовостійкість	низька	висока

Таким чином, 2PC доцільно застосовувати у випадках, коли цілісність і узгодженість критично важливі, але можна допустити затримки, тоді як Saga краще підходить для динамічних розподілених середовищ, що вимагають швидкої реакції і масштабованості, але допускають компроміс щодо строгої ізоляції транзакцій. Результати дослідження

демонструють, що вибір протоколу управління транзакціями значною мірою визначається пріоритетами та специфікою задач конкретної системи.

Двофазний коміт є найкращим рішенням для застосунків, які ставлять на перше місце цілісність та строгі вимоги до узгодженості даних. Проте, як вказують Daraghmi et al. [1], висока латентність цього підходу та ризик централізованої точки відмови координатора можуть критично знизити продуктивність, особливо у розподілених системах з великим навантаженням і високою інтенсивністю транзакційних операцій.

У свою чергу, патерн Saga, описаний Talaver та Vakaliuk [3], виграє в аспектах швидкодії та масштабованості. Це робить його кращим для систем, які працюють з великими обсягами даних, де затримки у виконанні транзакцій є неприпустимими. Водночас, недостатня ізоляція транзакційних операцій патерну Saga може негативно позначитися на якості та узгодженості даних у системах з високою взаємозалежністю транзакцій.

Для забезпечення ефективної роботи транспортних інформаційних систем доцільно використовувати патерн Saga з додатковими заходами щодо мінімізації негативних ефектів від недостатньої ізоляції, такими як ретельне моделювання меж між сервісами та обмеження взаємних залежностей [3]. Подальші дослідження доцільно спрямувати на створення комбінованих або адаптивних підходів, які дозволять збалансувати вимоги до швидкості, масштабованості та узгодженості транзакцій у сучасних транспортних ІС.

Висновки

На підставі проведеного аналізу можна зробити висновок, що сучасні протоколи управління транзакціями мають свої сильні та слабкі сторони. Класичний 2PC забезпечує строгий контроль узгодженості, але не підходить для систем із високою динамікою операцій через значні затримки та потенційний ризик відмови одного з учасників. Патерн Saga, особливо у своїй удосконаленій реалізації з використанням in-memory кешування, є більш перспективним для розподілених систем, де пріоритетом є висока продуктивність та гнучкість управління транзакціями. Отже, для систем транспортного обслуговування, де оперативність обробки транзакцій є критичною, варто віддати перевагу протоколам, що дозволяють знизити затримки при збереженні достатнього рівня узгодженості даних.

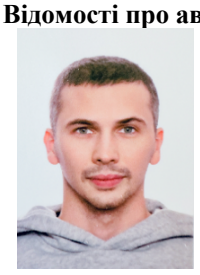
Список використаних джерел

1 Daraghmi, E.; Zhang, C.-P.; Yuan, S.-M. Enhancing Saga Pattern for Distributed Transactions within a Microservices Architecture. Appl. Sci. 2022, 12, 6242. DOI: 10.3390/app12126242.Talaver, O. V. та Vakaliuk, T. A. «Reliable distributed systems: review of modern approaches». Journal of Edge Computing, 2023, 2(1), с. 84–101. DOI:10.55056/jec.586.

2. Zangana, H. M.; Zeebaree, S. R. M. Distributed Systems for Artificial Intelligence in Cloud Computing: A Review of AI-Powered Applications and Services. International Journal of Informatics Information System and Computer Engineering, 2024, 5(1), с.11–30.

3. Talaver, O. V.; Vakaliuk, T. A. Reliable distributed systems: review of modern approaches. Journal of Edge Computing, 2023, 2(1), с.84–101. DOI: 10.55056/jec.586.

Відомості про автора:



Бех Андрій Олександрович – аспірант 1-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення

E-mail: andrii.bekh@gmail.com

УДК 004.415.2:004.8

ДОСЛІДЖЕННЯ ПІДХОДІВ ДО ГЕНЕРАЦІЇ КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ З ТЕКСТОВОГО ОПИСУ ІЗ ЗАСТОСУВАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ТА БІБЛІОТЕК ВІЗУАЛЬНИХ КОМПОНЕНТІВ

Данило БЛІЗНІЧЕНКО
Здобувач вищої освіти 1 курсу магістратури кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник доцент кафедри ІПЗ ФКНТ
Євген Олександрович Татаринов

У роботі досліджено сучасні методи автоматичної генерації графічних користувацьких інтерфейсів на основі текстового опису із залученням великих мовнихмоделей (LLM) та візуальних компонентів.

Ключові слова: *генерація користувацького інтерфейсу; великі мовні моделі; графічний інтерфейс користувача; zero-shot; wireframing; генерація коду; бібліотеки візуальних компонентів.*

Вступ

Автоматична генерація користувацьких інтерфейсів (UI) з текстового опису за допомогою великих мовних моделей (LLM) відкриває нові можливості в UI-дизайні. Такий підхід спрощує перехід від задуму до реалізації інтерфейсу, прискорює розробку та зменшує кількість помилок у комунікації між дизайнерами й розробниками.

Попри значний прогрес у можливостях LLM, створення повноцінного, структурованого та стилістично узгодженого користувацького інтерфейсу з тексту залишається складною задачею. Необхідно забезпечити відповідність дизайну, технологічному стеку та стандартам користувацького досвіду (UX). Особливу роль тут відіграє підготовка інструкцій для мовної моделі і використання бібліотек візуальнихкомпонентів, які дають змогу адаптувати генерацію до реальних практик розробки.

Ціль роботи

Метою роботи є проаналізувати підходи до автоматичної генерації користувацьких інтерфейсів на основі текстових описів із використанням великих мовних моделей і бібліотек візуальних компонентів. Основна увага приділена виявленню особливостей кожного підходу, їх переваг і недоліків.

Матеріали та методи

У роботі проведено аналіз чотирьох сучасних підходів до генерації користувацького інтерфейсу з тексту за участі великих мовних моделей. Вибрані джерела репрезентують різні стратегії: навчання спеціалізованої LLM на даних інтерфейсів, донавчання універсальної моделі на наборі даних інтерфейсів користувача, zero-shot генерацію з інженерією запитів, а також комплексну генерацію коду з макетіві доменних знань.

Результати та обговорення

Модель, навчена для пошуку макетів графічних інтерфейсів [1], забезпечує генерацію низькодеталізованих прототипів користувацького інтерфейсу на основі текстових запитів. Вона працює як інструмент швидкого підбору варіантів компонування, які потім можуть

бути доопрацьовані вручну. Результати оцінювання показали високу релевантність варіантів, але відсутність готового коду та потреба у попередньому навчанні обмежують масштабованість цього підходу.

Донавчене рішення для побудови макетів інтерфейсів (wireframing) [2] демонструє вищу точність завдяки адаптації LLM до структури UI-даних. Згенеровані макети містять реалістичний контент і відповідають загальноприйнятим принципамдизайну. Результати експериментів підтверджують значне зростання якості порівняно з базовими методами.

Zero-shot генерація коду для графічних користувацьких інтерфейсів [3] реалізується через інженерію підказок і стратегії покрокового міркування або самокритики. Вона не потребує донавчання, дозволяє створювати повноцінний HTML/JSX-код і показала найкращі результати за умови застосування self-critique. Водночас якість генерації залишається чутливою до структури запиту.

Фреймворк із генерацією коду на основі макетів і знань [3] поєднує аналіз макетів, візуальні підказки та доменні правила. Це дозволяє LLM створювати багатокомпонентні UI-застосунки на React. Система є найбільш повною за функціональністю, але вимагає складної попередньої підготовки.

Усі розглянуті методи підтверджують ефективність LLM у задачах генерації інтерфейсів. Донавчання покращує якість макетів, zero-shot підходи забезпечують гнучкість, а фреймворки дозволяють створювати повноцінні UI-додатки. Найбільш перспективними є поєднання генеративних моделей зі знаннями про дизайн та компонентні бібліотеки, однак актуальними залишаються виклики якості, узгодженості та валідації результатів.

Висновки

У роботі проаналізовано сучасні підходи до генерації користувацького інтерфейсу з текстового опису із застосуванням LLM. Якість результату залежить не лише від моделі, а й від структури підказок, наявності правил дизайну та використання компонентних бібліотек. Zero-shot стратегії забезпечують гнучкість, а комплексні фреймворки – повноцінну генерацію коду для практичного застосування. Перспективні напрями для подальших досліджень включають автоматичну оцінкуякості згенерованого інтерфейсу, мультимодальну генерацію з макетів, генерацію кодубекенду, оптимізацію інструкцій і залучення користувачів до оцінювання результатів.

Список використаних джерел

1. P. Brie, N. Burny, A. Sluyters, and J. Vanderdonckt, “Evaluating a large languagemodel on searching for gui layouts,” Proceedings of the ACM on Human-Computer Interaction, vol. 7, no. EICS, pp. 1–37, 2023

2. Designing with Language: Wireframing UI Design Intent with Generative Large Language Models / S. Feng et al. ar5iv. URL: https://ar5iv.labs.arxiv.org/html/2312.07755.

3. Zero-Shot Prompting Approaches for LLM-based Graphical User Interface Generation / K. Kolthoff et al. ar5iv. URL: https://ar5iv.labs.arxiv.org/html/2412.11328v1.

Відомості про автора:

Блізніченко Данило Романович – здобувач вищої освіти 1-го курсу магістратури кафедри інженерії програмного забезпечення факультету комп’ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси:* інженерія програмного забезпечення, сучасна розробка, розробка користувацьких інтерфейсів.

E-mail: 6964897@stud.kai.edu.ua

УДК 004.414.2:347.77

ЛІЦЕНЗІЇ З ВІДКРИТИМ КОДОМ І ЇХ ЗАСТОСУВАННЯ У РОЗРОБЦІ КОМЕРЦІЙНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Гліб Бродський
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий консультант к.т.н., доцент кафедри ІПЗ ФКНТ
Яна Андріївна Бєлозьорова

У цьому дослідженні проведено порівняльний аналіз різних ліцензій з відкритим кодом та їх вплив на комерційне програмне забезпечення. Розглянуто основні аспекти використання відкритого коду в комерційних проєктах та проаналізовано переваги та недоліки кожної з ліцензій. Особливу увагу приділено аналізу ліцензій GNU GPL, MIT та Apache, а також їх впливу на бізнес-моделі та стратегії розвитку програмного забезпечення.

Ключові слова: *відкритий код, ліцензії, комерційне програмне забезпечення, GNU GPL, MIT, Apache.*

Вступ

Використання програмного забезпечення з відкритим кодом (Open source) стало важливою частиною сучасного процесу розробки програмного забезпечення. Відкритий код дозволяє розробникам використовувати, модифікувати та поширювати програмне забезпечення без обмежень, що надає значні переваги для комерційних проєктів. Відкрите програмне забезпечення сприяє інноваціям, знижує вартість розробки та підтримки програмних продуктів, а також забезпечує високу якість та безпеку програмного забезпечення завдяки широкому залученню спільноти розробників. Проте, різні ліцензії з відкритим кодом мають різні умови використання, які можуть впливати на комерційне використання програмного забезпечення. Вибір ліцензії може мати значний вплив на бізнес-моделі та стратегії розвитку програмного забезпечення. Тому важливо розуміти особливості та умови кожної з ліцензій, а також їх вплив на комерційне використання програмного забезпечення.

Ціль роботи

Метою дослідження є здійснення порівняльного аналізу ліцензій з відкритим кодом та визначення їх впливу на стратегії комерційного використання програмного забезпечення. У роботі розглядаються популярні ліцензії, такі як GNU GPL, MIT, Apache та аналізуються їх переваги і недоліки в контексті комерційного використання. Також досліджено вплив ліцензій на бізнес-моделі та стратегії розвитку програмного забезпечення.

Матеріали та методи

Для проведення дослідження були проаналізовані документи ліцензій, такі як GNU GPL, MIT та Apache з точки зору їх впливу на комерційне програмне забезпечення. Було проведено порівняльний аналіз умов використання кожної з ліцензій та їх вплив на комерційні проєкти.

Результати та обговорення

У результаті дослідження було виявлено, що кожна з ліцензій з відкритим кодом має свої особливості та умови використання, які можуть впливати на комерційне програмне забезпечення. Наприклад, ліцензія GNU GPL вимагає, щоб будь-які модифікації програмного забезпечення також поширювалися під цією ліцензією, що може обмежувати комерційне використання. З іншого боку, ліцензія MIT дозволяє використовувати програмне забезпечення з відкритим кодом у комерційних проєктах без обмежень, що робить її більш привабливою для комерційних розробників.

1. Ліцензія Apache 2.0 надає користувачам широкі права на використання програмного забезпечення, включаючи право на використання патентів, повє'язаних з програмним забезпеченням. Згідно з офіційним текстом ліцензії:

«3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed» [1].

Це положення забезпечує користувачам додатковий захист від патентних позовів, що робить ліцензію Apache привабливою для комерційних розробників. Google випускає під Apache 2.0 численні проєкти, включаючи TensorFlow, який широко інтегрується комерційними компаніями, такими як DeepMind та NVIDIA у їхні продукти, пов'язані із штучним інтелектом [2]. Так само Amazon Web Services розвиває фреймворк Apache MXNet з використанням Apache 2.0, дозволяючи клієнтам впроваджувати масштабні зміни без патентних ризиків від AWS чи постачальників [3].

2. Ліцензія GNU GPL (General Public License) є однією з найпоширеніших ліцензій з відкритим кодом. Вона гарантує користувачам свободу запускати, вивчати, змінювати та поширювати програмне забезпечення. Проте, GNU GPL вимагає, щоб будь-які модифікації програмного забезпечення також поширювалися під цією ліцензією. Офіційний текст ліцензії зазначає:

«The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users» [4].

Це означає, що якщо компанія використовує програмне забезпечення під ліцензією GNU GPL у своєму комерційному проєкті, вона повинна також надати доступ до вихідного коду своїх модифікацій. Наприклад, MySQL має GPL версію та комерційну ліцензію від Oracle, щоб розробники могли уникнути необхідності розкривати вихідні коди, вони купують комерційну ліцензію [5, 6]. Іншим прикладом є VLC Media Player, ліцензія на який унеможливлює його інтеграцію в закриті комерційні продукти без розкриття вихідного коду.

3. Ліцензія MIT є однією з найліберальніших ліцензій з відкритим кодом. Вона дозволяє використовувати, копіювати, модифікувати, об'єднувати, публікувати, поширювати, субліцензувати та продавати копії програмного забезпечення без обмежень.

Ліцензія MIT вимагає лише збереження інформації про авторські права та текст ліцензії у всіх копіях або суттєвих частинах програмного забезпечення. Офіційний текст ліцензії:

«Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software» [7].

Це робить ліцензію MIT дуже привабливою для комерційних розробників, оскільки вона не накладає жодних обмежень на використання програмного забезпечення в комерційних проєктах. Microsoft випустила під ліцензією від MIT .NET Core та VS Code, дозволяючи іншим розробникам та компаніям без обмежень включати ці інструменти у свої закриті продукти. Також Meta перевела React на MIT після тиску від спільноти, зробивши його універсальною бібліотекою для комерційної розробки, що активно використовується компаніями як Shopify, Uber, Airbnb [8].

Порівняння вище описаних ліцензій можна побачити у Таблиці 1.

Таблиця 1

Порівняння ліцензій

Критерій	Apache 2.0	GNU GPL	MIT
Тип ліцензії	Ліцензія з мінімальними обмеженнями (permissive)	Ліцензія з умовою відкриття коду (copyleft)	Ліцензія з мінімальними обмеженнями (permissive)
Використання в комерційних цілях	Дозволене без обмежень	Лише за умови відкриття змін	Дозволене без обмежень
Умови розповсюдження	Збереження ліцензії + патентний блок	Збереження ліцензії + відкриття похідних робіт	Збереження ліцензії
Патентні права	Надаються явно	Не надаються	Не надаються
Сумісність із закритим ПЗ	Висока	Низька	Висока
Поширеність у бізнесі	Висока (Google, AWS)	Обмежена (MySQL)	Дуже висока (Meta, Microsoft)

Висновки

У цій роботі проведено порівняльний аналіз ліцензій з відкритим кодом та досліджено їх вплив на комерційне програмне забезпечення. Проведений аналіз показав, що вибір ліцензії суттєво визначає можливості інтеграції відкритого коду в комерційні продукти. Зокрема, ліцензії типу copyleft (наприклад, GNU GPL) можуть обмежити використання через вимогу відкритості модифікацій, тоді як ліцензії типу permissive (MIT, Apache) створюють сприятливі умови для бізнесу. Компаніям рекомендовано ретельно аналізувати юридичні наслідки вибраної ліцензії ще на етапі проєктування продукту.

Список використаних джерел:

1. Apache Software Foundation. Apache License, Version 2.0. https://www.apache.org/licenses/LICENSE-2.0.

2. Contributors to Wikimedia projects. TensorFlow - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/TensorFlow.

3. Contributors to Wikimedia projects. Apache MXNet - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Apache_MXNet.

4. GNU Operating System. GNU General Public License. https://www.gnu.org/licenses/gpl-3.0.html.

5. Contributors to Wikimedia projects. MySQL - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/MySQL.

6. Contributors to Wikimedia projects. MySQL Enterprise - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/MySQL_Enterprise.

7. Open Source Initiative. MIT License. https://opensource.org/licenses/MIT.

8. Larson Q. Facebook just changed the license on React. Here's a 2-minute explanation why. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/facebook-just-changed-the-license-on-react-heres-a-2-minute-explanation-why-5878478913b2/

Відомості про автора:

Бродський Гліб Володимирович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: відкритий код, ліцензії, комерційне програмне забезпечення.

E-mail: 7412397@stud.kai.edu.ua

,

УДК 004.738:339.371.4(477:4)(043.2)

ВЕБДОДАТОК ДЛЯ КУПІВЛІ ТОВАРІВ З ЄВРОПИ: ОПТИМІЗАЦІЯ ТА РОЗВИТОК ЕЛЕКТРОННОЇ КОМЕРЦІЇ В УКРАЇНІ

Михайло ВАСИЛЬКОВ
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Вікторія Олексіївна Волкогон

Дослідження присвячене розробці вебдодатка для надання послуг з купівлі товарів з Європи для українських споживачів. Метою роботи є забезпечення швидкого, зручного та безпечного процесу покупки європейських товарів через інтеграцію з міжнародними платіжними та логістичними системами. Використано сучасні технології, зокрема .NET, MS SQL та Blazor, для створення високопродуктивного та масштабованого додатка з трирівневою архітектурою. Додаток має на меті полегшити процес покупки, забезпечити прозорість угод і покращити логістику шляхом автоматизації відстеження доставки замовлень через API міжнародних поштових служб.

Ключові слова: *вебдодаток, електронна комерція, купівля товарів з Європи, логістика, інтеграція, API, платіжні системи, автоматизація, архітектура, .NET, Blazor, e-commerce, конкуренція, український ринок, онлайн-торгівля, міжнародні поштові служби.*

Вступ

Згідно з аналізом McKinsey Global Institute 2024, сегмент роздрібної електронної комерції наразі забезпечує ⅕ світових доходів від роздрібної торгівлі загалом — ця частка може сягнути 27–38 % до 2040 року. В умовах війни, що ускладнює офлайн-продажі, e-commerce України розвивається навіть швидше — це зумовлює необхідність зберігати конкурентоспроможність та впроваджувати сучасні рішення.

В Україні, згідно з даними RetailersUA, в 2024 році обсяг онлайн-торгівлі склав 239 млрд грн, що на 25% більше порівняно з попереднім роком. Це є показником не тільки стабільного розвитку сектору, але й зростання надійності та довіри до онлайн-сервісів, що підтверджує необхідність розробки нових технологічних рішень для задоволення зростаючого попиту.

Розробка вебдодатка для надання послуг із купівлі товарів з Європи дозволяє спростити процес для споживачів, забезпечуючи прозорість, зручність та безпеку угод. Важливими аспектами є побудова ефективної архітектури, автоматизація процесу замовлення та оптимізація логістики.

Ціль роботи

Метою дослідження є розробка та впровадження вебдодатка, що забезпечує користувачів можливістю легкої купівлі товарів з європейських онлайн-магазинів із мінімальними зусиллями та фінансовими ризиками.

Основні завдання включають:

- Аналіз ринку вебсервісів для купівлі товарів з Європи та оцінка конкурентного середовища.
- Проєктування архітектури вебдодатка з урахуванням принципів SOLID і OOP.

- Реалізація інтеграції з платіжними та логістичними системами.
- Дослідження можливостей масштабування та впровадження AI для покращення сервісу.

Матеріали та методи

Дослідження проводилось із використанням таких методів:

- Аналіз ринку: Вивчення статистичних даних про ринок e-commerce в Україні та Європі.
- Методи проєктування програмного забезпечення: Використання багаторівневої архітектури для забезпечення масштабованості.
- Моделювання логістичних процесів: Використання API міжнародних поштових служб для аналізу ефективності доставки.

Результати та обговорення

Попит на європейські товари в Україні зростає через кілька факторів, серед яких:

Висока якість продукції: Європейські виробники здобули репутацію завдяки високим стандартам виробництва, що робить їх товари привабливими для споживачів в Україні.

Різноманіття вибору: Завдяки багатим асортиментам, європейські онлайн-магазини пропонують широкий спектр товарів, які можуть бути важкодоступні на місцевому ринку. Особливо популярні категорії включають:

- Одяг і взуття.
- Електроніку та гаджети.
- Косметику та парфумерію.

Конкурентне середовище

На ринку вже існують сервіси, що надають посередницькі послуги з купівлі товарів з Європи, проте більшість із них стикається з низкою проблем, такими як довгі терміни доставки, високі комісії та обмежений вибір товарів. Це створює значні незручності для користувачів, які прагнуть швидко та вигідно отримувати товари з-за кордону.

Запропонований вебдодаток стане потужним конкурентом завдяки поєднанню сучасних технологій, продуманих алгоритмів та ефективної логістики. Його ключові переваги включають високу швидкість обробки замовлень, оптимізацію процесів завдяки автоматизації та інтеграції з міжнародними сервісами, а також прозору систему розрахунку вартості з урахуванням всіх митних платежів. Крім того, особлива увага приділяється зручності користувачів та високій підтримці, що забезпечить надійність і довіру до сервісу.

Архітектурний підхід

Вебдодаток розроблений у вигляді двох окремих додатків:

Back-end: .NET (C#) + MS SQL, що відповідає за бізнес-логіку та роботу з базою даних.

Front-end: Blazor, що забезпечує інтерактивний користувацький інтерфейс.

На серверній частині реалізовано трирівневу архітектуру, яка включає:

1. Рівень даних (Data Layer): Цей рівень відповідає за зберігання даних у базі даних MS SQL. Він включає в себе всі операції з доступом до даних, такі як створення, читання, оновлення та видалення (CRUD). Окремо виділяється Data Access Layer (DAL), який реалізує репозиторії, що забезпечують підключення до бази даних, а також абстрагують взаємодію з даними. Це дозволяє підтримувати чітке розмежування між бізнес-логікою та даними.

2. Рівень бізнес-логіки (Business Logic Layer - BLL): Цей рівень відповідає за реалізацію основної логіки додатка. Він містить сервіси (Services), які обробляють бізнес-операції, а також моделі (Models), які представляють основні об'єкти системи, такі як замовлення, товари та користувачі. BLL взаємодіє з DAL для отримання та маніпуляції даними, при цьому бізнес-логіка залишається відокремленою від деталей реалізації доступу до даних.

3. Рівень представлення (API Layer): Цей рівень відповідає за взаємодію з клієнтською частиною (фронтендом) та іншими зовнішніми системами через REST API. API обробляє запити від користувачів, викликає відповідні сервіси з BLL для виконання бізнес-логіки та повертає відповіді у форматі, зрозумілому для клієнта.

Вибір технологій обумовлений кількома факторами:

- .NET (C#) був обраний через свою надійність, ефективність та розвинену екосистему, що дозволяє швидко розробляти високопродуктивні серверні додатки. C# має хорошу інтеграцію з базами даних, а також підтримку масштабованості та безпеки.
- MS SQL забезпечує потужну та надійну роботу з даними, а також легкість інтеграції з .NET, що значно спрощує роботу з великими обсягами даних та дозволяє здійснювати складні запити.
- Blazor вибраний для фронтенду, оскільки він дозволяє використовувати C# і .NET для створення інтерактивних вебдодатків замість JavaScript. Це спрощує розробку та підтримку, адже можна працювати в єдиній екосистемі.
- REST API забезпечує стандартизований підхід для комунікації між клієнтською частиною та сервером, а також для інтеграції з іншими системами чи зовнішніми сервісами. Цей підхід дозволяє забезпечити гнучкість у взаємодії з іншими додатками та сервісами, що є важливим для майбутнього розширення.

Висновки

Розробка вебдодатка для надання послуг із купівлі товарів з Європи є важливим кроком у напрямку покращення доступу українських споживачів до якісної європейської продукції. Оскільки попит на європейські товари в Україні постійно зростає, забезпечення швидкого, зручного та безпечного процесу покупки є важливим завданням для бізнесу.

Використання сучасних технологій, а також правильний підхід до архітектури вебдодатка, дозволяє досягти високої продуктивності, безпеки та масштабованості проєкту. Особлива увага до логістики та трекінгу дозволяє значно покращити користувацький досвід. Інтеграція з міжнародними поштовими службами через API дає змогу автоматично відстежувати статус доставки замовлень, що дозволяє користувачам отримувати актуальну інформацію про терміни доставки. Це є важливим елементом для підвищення довіри до сервісу та зниження рівня незручностей, пов'язаних з доставкою.

Загалом, успішна реалізація такого вебдодатка відкриває нові можливості для розвитку електронної комерції в Україні, забезпечуючи більш ефективний доступ до європейських товарів, що в кінцевому підсумку призведе до покращення конкурентоспроможності на ринку онлайн-торгівлі.

Список використаних джерел

1. Chaffey, D. (2020). Digital Business and E-Commerce Management. Pearson Education.

2. URL:https://www.shopify.com/blog/ecommerce-logistics (дата звернення: 22.03.2025)
3. Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley.
4. URL: https://retailers.ua/news/ecommerce-2024 (дата звернення: 22.03.2025)
URL: https://eba.com.ua/yak-zminylasya-elektronna-komertsiya-u-2022-rotsi-doslidzhennya-admitad/ (дата звернення: 22.03.2025)

Відомості про автора:

Васильков Михайло Григорович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, електрона комерція.
E-mail: 7363186@stud.kai.edu.ua

УДК 004. 413 (045)

LOW-CODE/NO-CODE ТЕХНОЛОГІЇ У МАЙБУТНЬОМУ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Юлія ГАБЄЄВА
Здобувачка вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Лариса Валеріївна Дакова

У статті розглядаються основні принципи та переваги використання Low-code та No-code технологій в інженерії програмного забезпечення.

Ключові слова: *Low-code, No-code, платформи, інженерія програмного забезпечення, автоматизація, розробка застосунків, нетехнічні користувачі.*

Вступ

У сучасному світі інженерія програмного забезпечення постійно підлаштовується до нових викликів, серед яких – необхідність прискорення розробки додатків, збільшення кола розробників і зниження витрат на створення програмного забезпечення. Одним з ключових рішень стали Low-code та No-code технології, що пропонують новий підхід до побудови програмних продуктів, мінімізуючи або повністю виключаючи традиційне програмування.

Low-code/No-code платформи надають можливість розробляти повноцінні додатки за допомогою візуальних інструментів, готових шаблонів та модулів, що значно зменшує час та вартість розробки проєктів. Їх використання розширює коло учасників процесу створення програмного забезпечення, дозволяючи залучати не тільки професійних розробників, але й бізнес-аналітиків та користувачів без технічної освіти.

У зв'язку з розвитком цих технологій трансформуються вимоги до навичок розробників, а також з'являються нові методи управління життєвим циклом програмного забезпечення. Разом з цим виникають нові виклики, що стосуються масштабованості, захисту даних та обмежень в кастомізації створених рішень.

Актуальність дослідження обумовлена необхідністю оцінити потенціал Low-code та No-code технологій з точки зору їх впливу на майбутнє інженерії програмного забезпечення, а також визначити переваги, ризики та можливості їхнього застосування в різноманітних сферах.

Ціль роботи

Мета дослідження – проаналізувати значення Low-code та No-code технологій в сучасній розробці. Визначити їх сильні та слабкі стороні, а також вплив на класичні методи інженерії програмного забезпечення.

Матеріали та методи

Low-code та No-code платформи з’явилися як практичне рішення для бізнесу, який потребує швидкої розробки програм без глибоких технічних знань. Вони дозволяють створювати застосунки за допомогою зручних візуальних інструментів, використовуючи готові блоки та шаблони. За допомогою Low-code підходу дозволяється частково використовувати ручне програмування для розширення функціональності, а за допомогою No-code підходу – зовсім не писати код.

Традиційна розробка програмного забезпечення вимагає високого рівня технічних знань, значних часових і фінансових витрат. Натомість, Low-code платформи значно прискорюють процес створення застосунків за рахунок автоматизації рутинних завдань, зберігаючи можливість розробникам впливати на ключові аспекти проєкту. No-code рішення націлені переважно на користувачів без спеціальних знань, які можуть самостійно створювати базові або середньої складності програми.

Відмінності між традиційною розробкою, Low-code і No-code платформами наведено у таблиці 1.

Таблиця 1

Порівняння традиційної розробки, Low-code та No-code платформ

Характеристика	Традиційна розробка	Low-code платформи	No-code платформи
Обсяг коду	100% ручне кодування	Часткове кодування	Кодування відсутнє
Потреба у програмістах	Обов'язкова	Бажана	Не потрібна
Швидкість розробки	Низька	Висока	Дуже висока
Гнучкість рішення	Висока	Середня	Низька
Приклади	Java, C#, Python	OutSystems, Mendix	Bubble, Adalo

Сучасний ринок пропонує різноманітні Low-code та No-code платформи, які вирішують багато різних завдань. Їх розрізняють за функціональністю та ціллю використання. Залежно від призначення їх можна класифікувати на кілька основних типів:

- Платформи для веброзробки (Webflow, Wix, Squarespace). Призначені для створення сайтів, вебзастосунків і односторінкових застосунків без глибокого залучення до кодування
- Платформи для мобільних застосунків (Adalo, Appgyver, Glide). Спрямовані на розробку нативних або гібридних мобільних застосунків для Android та iOS без необхідності писати код
- Платформи для корпоративних систем (OutSystems, Mendix, Microsoft Power Apps). Спрямовані на створення складних бізнес-застосунків, що інтегруються з корпоративними базами даних та IT-інфраструктурою.
- Платформи для автоматизації бізнес-процесів (Zapier, Airtable, Integromat). Допомагають налаштовувати автоматизовані робочі процеси шляхом інтеграції різних сервісів без програмування.
- Платформи для створення електронної комерції (Shopify, BigCommerce). Дозволяють створювати онлайн-магазини без програмування за допомогою візуальних конструкторів.

Платформи для веброзробки, такі як Webflow чи Wix, дозволяють створювати сайти без знання спеціальних технічних знань. Інструменти для мобільних додатків, Adalo, OutSystems, Mendix, пропонують рішення для побудови складних бізнес-застосунків з можливістю інтеграції у внутрішні системи компаній. Платформи для автоматизації бізнес-процесів, Zapier чи Airtable, фокусуються на спрощенні рутинних завдань шляхом налаштування автоматизації робочих процесів.

Результати та обговорення

Результати дослідження показують, що Low-code та No-code технології суттєво спрощують процес розробки програмного забезпечення. Одна з основних переваг це зменшення часу на розробку порівняно з традиційними методами програмування. Платформи надають візуальні інтерфейси, готові компоненти та шаблони, які дозволяють за допомогою мінімальної кількості коду швидко створювати застосунки, що зменшує вимоги до кваліфікації розробників і дозволяє скоротити час реалізації проєктів.

Окрім того, Low-code та No-code платформи мають великий потенціал для залучення нетехнічних користувачів. Такі платформи, орієнтовані на створення вебзастосунків та мобільних застосунків, дозволяють користувачам без програмістських навичок створювати функціональні та естетичні продукти.

Одним із головних недоліків є обмеження у масштабуванні та гнучкості. Технології Low-code та No-code найкраще підходять для створення простої або середньої складності застосунків. Якщо проєкти вимагають високої продуктивності, гнучкості або специфічних функцій, то такі технології можуть бути малоефективними у розробці.

Щодо використання у корпоративному середовищі, важливо відзначити, що платформи, зосереджені на автоматизації бізнес-процесів, мають великий потенціал для полегшення інтеграції різних IT-систем. Вони допомагають скоротити час на налаштування бізнес-процесів та зменшують ймовірність помилок, пов'язаних з ручним введенням даних.

Висновки

Отже, Low-code та No-code технології стають важливими інструментами у сучасній розробці, вони значно прискорюють процес розробки та дозволяють залучати користувачів без технічних знань до створення програмних рішень.

Завдяки використанню даних технологій, знижується бар’єр для входа в програмну інженерію, зменшуються витрати на розробку та збільшується зручність швидкого створення простих або середніх по складності застосунків.

Список використаних джерел

1. Корзаченко О.В. Low-Code та No-Code BPMS: сучасні тренди автоматизації бізнес-процесів підприємства. 2022. С. 114. [Електронний ресурс]. — https://ir.kneu.edu.ua/server/api/core/bitstreams/f237d8da-b9e1-461c-838a-0535bc27401a/content (дата звернення: 26.04.2025)

2. Швидше і дешевше. No-code рішення для мобільних застосунків. [Електронний ресурс]. — https://netpeak.net/uk/blog/shvidshe-i-deshevshe-no-code-rishennya-dlya-mobil-nikh-zastosunkiv/ (дата звернення: 28.04.2025)

3. Худякова І.М. No-Code і Low-Code технології як основа дисципліни «Інтернет технології» для гуманітарних спеціальностей. The 5 th International scientific and practical conference—Modern research in world science (August 7-9, 2022) SPC—Sci-conf. com. ua‖, Lviv, Ukraine. 2022. 1067 p. 2022.

4. Створіть сайт без обмежень. [Електронний ресурс]. — Режим доступу: https://uk.wix.com/ (дата звернення : 26.04.2025)

5. DeSilva D.I., Ranathunga R.A. A. L., Shangavie R. Quality Assurance in Low-Code/No-Code Development. 2023 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES), Chennai, India, 14–15 December 2023. 2023. [Електронний ресурс]. — Режим доступу: https://doi.org/10.1109/icses60034.2023.10465268 (дата звернення: 26.04.2025)

6. Difference Between Low-Code and No-Code Platform. Kissflow. [Електронний ресурс]. — Режим доступу: https://kissflow.com/low-code/low-code-vs-no-code/ (дата звернення: 26.04.2025)

Відомості про автора:

Габєєва Юлія Іванівна – здобувачка вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, сучасна розробка, Low-code and No-code технології.

E-mail: 7363186@stud.kai.edu.ua

УДК 004. 4

ГЕНЕРАЦІЯ КОДУ З ДОПОМОГОЮ MICROSOFT COPILOT

Дмитро ГОЛОЛОБОВ
Кандидат фізико-математичних наук
Доцент кафедри ІПЗ
ДПН «Державний університет «Київський авіаційний інститут»

Розглянуто особливості генерації початкового коду безкоштовним програмним забезпеченням Microsoft Copilot на мові асемблера на основі текстових запитів.

Ключові слова: *генеративний штучний інтелект, чат-бот, Microsoft Copilot, асемблер, GPT-4.*

Вступ

Початок 20 рр. XXI ст. відмітився поверненням штучного інтелекту (ШІ) до трендів сучасного суспільства. Ця «весна штучного інтелекту» пов'язана з розвитком генеративного ШІ, головним чином, чат-ботів, що генерують різноманітний контент (текст, зображення, музику, відео тощо) у відповідь на запит людини. Не стала винятком і генерація початкового коду програм на різноманітних мовах програмування. Помічники на основі ШІ стали впроваджуватися практично до всіх сфер життя людини. Це привело до того, що реальна корисність і якість продуктів на основі генеративного ШІ стала поширеним предметом наукових досліджень.

Microsoft Copilot (початкова назва «Bing Chat») [1] – чат-бот на основі генеративного штучного інтелекту, розроблений Microsoft на основі GPT-4, реліз якого відбувся 2023 року. Більшість функцій доступні безкоштовно, без авторизації, деякі (зокрема, генерація зображень на основі текстових підказок) через обліковий запис Windows, платна підписка дає пріоритетний доступ до нових функцій, включаючи створення власних чат-ботів. Стиль розмовного інтерфейсу Microsoft Copilot нагадує ChatGPT, – чат-бот може цитувати джерела, створювати вірші та пісні.

Цілі та методи дослідження

Мета роботи – дослідити в контексті якості результати роботи чат-бота з генеративним штучним інтелектом Microsoft Copilot для випадку генерації початкового коду на мові асемблера.

Для проведення дослідження використовувався вебсайт Microsoft Copilot [2], браузер Microsoft Edge, була обрана опція поглибленої відповіді «Think Deeper». Формулювання задач до ШІ виконувалося українською мовою.

Результати дослідження

ШІ Microsoft Copilot здатен генерувати код на різних асемблерах (NASM, TASM, FASM, GAS, MASM тощо) для різноманітних платформ та архітектур (Windows, Linux, ARM, FreeBSD, MacOS тощо).

За замовчуванням, якщо вказати тільки завдання з написання програми на асемблері, без вказування додаткових вимог, код зазвичай буде наведений на асемблері NASM для Linux, x86, 32 біт. Якщо задати конкретний (інший) асемблер, то код може бути згенерований для іншої платформи, наприклад, MS DOS. Якщо у подальшому після

конкретизації не вказувати необхідну мову/платформу, то ШІ генеруватиме код відповідно контексту (попередніх вимог).

Для отримання бажаного результату, потрібно вказувати додаткові умови, причому роботи це слід максимально точно і докладно. Це також є справедливим і для отримання оптимального з точки зору продуктивності чи за іншим критерієм результату. Наприклад, якщо поставити задачу додавання двох чисел без додаткових вимог, то результат (NASM) буде містити певний загальний шаблон, який включає, зокрема, область даних (section .data), що не є обов'язковим у загальному випадку, наприклад, якщо нема потреби у використанні змінних. Ці надлишкові дані є навіть у тому з прикладів розв'язання задачі, який ШІ визначає найпростішим.

При постановці умови виконання задачі над числами заданої розрядності, ШІ може використовувати операнди іншої розрядності для допоміжних дій (наприклад, обнулення регістра командою xor), якщо не поставити спеціальну додаткову умову.

Висновки

Microsoft Copilot здатний на основі текстових запитів генерувати код на різних асемблерах для різноманітних платформ та архітектур.

Продуктивність програмних продуктів є актуальною незалежно від використовуваних мов і платформ, і асемблер наразі не є винятком. У випадку використання асемблера, як правило, критичним є використання пам'яті, що власне її є однією з переваг цієї мови програмування. Проведене дослідження показало, що згенерований код, за замовчуванням не відповідає ані критеріям оптимальної продуктивності, ані критеріям оптимального використання пам'яті. Більш того, у супроводжуючих коментарях, ШІ не дає рекомендацій щодо цих, безумовно, ключових для асемблера питань.

Таким чином, для отримання потрібного результату від Microsoft Copilot, необхідно докладно і точно формулювати задачу і всі додаткові уточнення, для чого людина, яка це робить, повинна досить непогано розуміти ключові особливості асемблера, тобто для новачків такий генератор коду точно не буде корисним. Зокрема, використання вказаного продукту з метою освіти може бути корисне лише при використанні додаткових матеріалів, які надають глибокі фундаментальні знання.

Список використаних джерел

1. Copilot — ваш помічник із ШІ. – URL: https://www.microsoft.com/uk-ua/microsoft-copilot/for-individuals (дата звернення: 05.05.2025).

2. Microsoft Copilot: You AI Companion. URL: https://copilot.microsoft.com (дата звернення: 05.05.2025).

Відомості про автора:

Гололобов Дмитро Олександрович – кандидат фізико-математичних наук, доцент кафедри Інженерії програмного забезпечення Факультету комп'ютерних наук та технологій Державного некомерційного підприємства «Державний університет «Київський авіаційний інститут». *Наукові інтереси*: Програмування, Інформаційні технології.

E-mail: dmytro.hololobov@npp.kai.edu.ua

УДК 004.421.2:004.415

ОСНОВНІ МОДЕЛІ ЯКОСТІ ПРОГРАМНИХ СИСТЕМ

Ярослав ДРОВОЛЬСЬКИЙ
Здобувач вищої освіти 1-го курсу
магістратури спеціальності 121
«Інженерія програмного забезпечення»
Факультет комп'ютерних наук та кібернетики,
Київський національний університет імені Тараса Шевченка
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Олена Олександрівна Гріненко

У роботі проаналізовано основні моделі якості програмного забезпечення: Маккола, Боема, ISO/IEC 9126 та ISO/IEC 25010. Розглянуто їхні характеристики та підходи до оцінювання якості ПЗ. Наголошено на важливості якості в умовах широкого використання програмних систем у різних сферах діяльності.

Ключові слова: *якість програмного забезпечення, моделі якості, модель Маккола, модель Боема, ISO/IEC 9126, ISO/IEC 25010, атрибути якості, стандарти ПЗ.*

Вступ

У сучасному світі програмне забезпечення широко використовується людьми як у їхньому повсякденному житті, так і у різноманітних бізнес-процесах. Зараз від ПЗ залежить багато аспектів людської життєдіяльності, наприклад, фінанси, державні послуги, енергетика, медицина, освіта, транспорт, промисловість, бізнес-комунікації тощо. До того ж, помилки у цих програмних системах призводять до серйозних наслідків: витоки персональних даних, фінансові втрати, аварії та збої в критичних системах (медицина, енергетика, транспорт). Саме тому якість ПЗ, її оцінювання і підтримання її високого рівня є надзвичайно важливою частиною розробки програмних систем, особливо в умовах, коли ПЗ глибоко проникло у людську життєдіяльність.

Ціль роботи

Ціллю роботи є ознайомлення з основними наявними моделями якості програмних систем.

Матеріали та методи

Матеріалами роботи є вітчизняна та зарубіжна література з програмного забезпечення, зокрема підручники, наукові публікації, методичні посібники та міжнародні стандарти. Методами для написання роботи є аналіз джерел, узагальнення та синтез, порівняння, системний підхід.

Результати та обговорення

Якість програмного забезпечення (Software Quality) – це ступінь відповідності програмної системи встановленим, передбачуваним або очікуваним вимогам чи потребам причетних сторін. Для визначення якості системи використовують так звані атрибути (характеристики) якості [1] [2].

Основою для комплексного оцінювання якості програмного забезпечення є моделі якості ПЗ. Саме вони і визначають, які характеристики якості програмного продукту

потрібно враховувати при оцінюванні його властивостей. Різні моделі мають різну кількість рівнів та різні характеристики якості [3].

Далі розглянемо основні моделі якості програмного забезпечення.

Першою загальновідомою моделлю стала модель Маккола, запропонована в 1977 році. В ній визначено 11 факторів якості ПЗ: коректність, надійність, ефективність, цілісність, практичність, супроводжуваність, оцінюваність, гнучкість, переносимість, повторне використання, здатність до взаємодії. Фактори поділені на три групи (відповідно до роду роботи людей з ПЗ): функціонування (експлуатація), здатність до змін, адаптивність до нових середовищ [1]. Часто ці фактори та їхній поділ на групи зображають у вигляді т. зв. «трикутника Маккола». Для числового вираження факторів у моделі використовуються 20 метрик якості: audability, accuracy, completeness, consistency тощо. Кожна метрика оцінюється за шкалою від 0 до 10. А значення фактора обчислюється як лінійна комбінація значень метрик, які впливають на даний фактор. Коефіцієнти для лінійної комбінації визначаються залежно від організації, команди розробки та виду програмного забезпечення. В рамках цієї моделі під час розробки встановлюють також критерії якості – цільові значення факторів [2].

Розширенням моделі Маккола є модель Боема, запропонована у 1978 році. У ній визначено 19 атрибутів якості, що включають всі 11 факторів якості з моделі Маккола. Додатково було визначено такі атрибути: ясність, зручність внесення змін (modifiability), документованість, зрозумілість, універсальність, економічна ефективність тощо. Атрибути якості класифікують за способами використання ПЗ [2].

Наразі найбільш широко застосовуваними є дві стандартизовані моделі якості ПЗ, які також є і стандартами де-факто: ISO/IEC 9126, ISO/IEC 25010.

У моделі ISO/IEC 9126, яка не є прямим розширенням описаних раніше моделей, оцінка якості програмного забезпечення ґрунтується на шести основних характеристиках зовнішньої і внутрішньої якості програмної системи (ПС) та чотирьох характеристиках експлуатаційної якості. До характеристик зовнішньої та внутрішньої якості входять наступні: функціональність, надійність (reliability), зручність використання (usability), ефективність, супроводжуваність (maintainability), портативність (portability) [2]. А до характеристик експлуатаційної якості входять результативність, продуктивність, безпечність та задоволеність користувачів. Кожна вищенаведена характеристика уточнюється за допомогою спеціального набору підхарактеристик і метрик – властивостей ПС, вимірних за певною шкалою.

Модель ISO/IEC 25010 є серйозним доопрацюванням попередньої моделі. Для більш точного опису процесу забезпечення якості програмного продукту було уточнено деякі характеристики якості та додано дві нові: сумісність та безпеку. Також було додано 7 нових підхарактеристик. Окрім цього, було здійснено перегрупування характеристик [3].

Висновки

В ході роботи було з’ясовано, що якість є надважливою при розробці програмних систем. Для її оцінювання використовують різні моделі якості, які визначають значущі характеристики якості. Окрім цього, було розглянуто чотири основні моделі якості ПЗ.

Список використаних джерел

1. Pressman R. S., Maxim B. R. Software engineering: a practitioner’s approach. 9th ed. New York: McGraw-Hill Education, 2020. 671 p.

2. Крепич С. Я., Співак І. Я. Якість програмного забезпечення та тестування: базовий курс : навчальний посібник Тернопіль : ФОП Паляниця В.А., 2020. 478 с.

3. Грицюк Ю. І. Система комплексного оцінювання якості програмного забезпечення. Науковий вісник НЛТУ України. 2022. Т. 32, № 2. С. 81–95. URL: https://doi.org/10.36930/40320213 (дата звернення: 03.05.2025).

Відомості про автора:

Дровольський Ярослав Васильович – здобувач вищої освіти 1-го курсу магістратури факультету комп'ютерних наук та кібернетики Київський національний університет імені Тараса Шевченка. *Наукові інтереси*: інженерія програмного забезпечення, якість програмного забезпечення, моделі якості.

E-mail: yar.drovolsky@gmail.com

УДК 027.6-056.265+004.912(043.2)

МАШИННИЙ ПЕРЕКЛАД ЖЕСТОВОЇ МОВИ: СУЧАСНІ ТЕХНОЛОГІЇ ТА ПЕРСПЕКТИВИ

Максим ІВАШКЕВИЧ
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Яна Андріївна Бєлозьорова

У роботі розглядається розвиток технологій, що дозволяють автоматично перекладати жестову мову у текст або голос. Проаналізовано виклики, пов'язані з візуально-кінетичною природою жестової мови, та сучасні підходи до її розпізнавання, зокрема з використанням комп'ютерного зору, глибокого навчання і нейронних мереж. Також підкреслюється соціальне значення таких технологій для забезпечення інклюзивності та доступу до інформації для людей з порушенням слуху.

Ключові слова: *машинний переклад, жестова мова, комп'ютерний зір, глибоке навчання, нейронні мережі, розпізнавання жестів, обробка природної мови (NLP), соціальна інклюзія, доступність інформації.*

Вступ

У сучасному цифровому суспільстві забезпечення рівного доступу до інформації для людей з порушеннями слуху набуло критичного значення. Жестова мова є основним засобом спілкування для мільйонів людей по всьому світу [1]. Проте через візуально-кінетичну природу жестикуляції переклад жестової мови у текст або голос — надзвичайно складне технічне завдання. На відміну від письмових мов, жестові мови не мають загальноприйнятої писемності, а їх передача відбувається через рухи тіла, рук, міміку та пози. Розвиток систем автоматичного перекладу жестової мови є актуальним як з соціальної (інклюзивність), так і з наукової точки зору, адже такі системи можуть суттєво покращити комунікацію між людьми, що чують, і спільнотою глухих та слабкочуючих.

Технічні виклики перекладу жестової мови

Багатовимірність та неоднорідність сигналів. Жестові мови являють собою багатовимірний канал комунікації: значення передається не лише формою рук, а й рухом, орієнтацією долонь, положенням тіла, мімікою обличчя, поглядом та навіть відстанню рук від тіла [2]. Різні компоненти жесту (ручні й неручні) можуть виконуватись асинхронно, одночасно передаючи інформацію. Для комп'ютерної системи вхідні дані є високорозмірними просторово-часовими послідовностями (відео), тож модель повинна розуміти, як виглядає мовник і як він рухається у тривимірному просторі, і що ці рухи означають у поєднанні [3]. Неручні компоненти (наприклад, вираз обличчя) досі рідко враховуються у великих системах, що знижує якість перекладу. Крім того, жестова мова не є прямим відображенням озвученої мови: вона має власну граматику і синтаксис, які часто не мають послівного відповідника у звучанні. Наприклад, порядок слів та способи вираження питань в американській жестовій мові відрізняються від англійської; багато понять передаються через контекст або міміку замість окремих слів. Відтак класичні методи

машинного перекладу, розроблені для послідовностей письмових слів, неможливо безпосередньо застосувати до жестів.

Обмеженість даних та різноманітність мов. Для тренування нейронних мереж потрібні великі корпуси даних «жест → текст». Нині доступні набори даних досить малі: зазвичай це лише десятки тисяч пар відео і відповідних речень, що на порядки менше, ніж паралельні корпуси для письмових мов [4]. Бракує і стандартів анотації жестів: лінгвісти досі не дійшли згоди, як найкраще записувати жестові вислови (існує поняття «глоси» – спрощені текстові позначення жестів, але їх недостатньо для повного відтворення мови). Обмежений обсяг та якість даних стримує навчання глибоких моделей перекладу і знижує їх універсальність. Ще один виклик – велика різноманітність жестових мов у світі: за оцінками, їх кількість відповідає кількості звукових мов. Існують, зокрема, американська (ASL), українська, британська, жестова мова жестів міжнародного спілкування тощо – кожна з власним словником і правилами. Моделі, натреновані на одній жестовій мові, погано працюють для іншої, тому потрібна локалізація і адаптація моделей під конкретні мовні спільноти. Таким чином, створення універсального «перекладача жестів» потребує розв’язання одразу кількох проблем: розпізнавання багатовимірних сигналів, подолання нестачі даних та врахування лінгвістичних відмінностей жестових мов.

Сучасні досягнення і технології

Глибоке навчання та комп’ютерний зір. Протягом останнього десятиліття дослідники досягли значного прогресу у розпізнаванні жестів завдяки появі глибоких нейронних мереж [5]. Класичні підходи базуються на методах комп’ютерного зору: зображення або відео жесту обробляються згортковими нейронними мережами (CNN) для виділення ознак, а часову динаміку руху моделюють рекурентні мережі (RNN/LSTM) або 3D-CNN. Такі системи успішно розпізнають окремі літери або слова жестової абетки, а іноді й безперервні послідовності жестів, але переважно у вузьких доменах (наприклад, цифри, окремі фрази). Сучасні датчики дозволяють отримувати більш багату інформацію про рух: зокрема, алгоритми на зразок MediaPipe Holistic від Google можуть виділяти координати ключових точок скелету людини (суглоби рук, позиція тіла, риси обличчя) з відео в реальному часі. Це відкриває можливості для подальшого аналізу: замість сирих пікселів відео модель оперує векторизованими параметрами рухів, що зменшує складність і робить розпізнавання більш стійким до зовнішніх умов. Наприклад, у 2021 році запропоновано використання просторово-часових графових нейронних мереж, які моделюють взаємодію суглобів рук у часі, досягаючи високої точності розпізнавання жестів. Загалом, поєднання комп’ютерного зору і глибокого навчання вже зараз дозволяє будувати системи, що автоматично розпізнають сотні жестів зі значною точністю.

Трансформери та мультимодальні моделі. Наступним кроком стало застосування трансформерів – сучасних нейромережевих архітектур, що відмінно зарекомендували себе у задачах природної мови. Трансформери дозволяють ефективно моделювати довготривалі залежності у послідовностях і працюють паралельно, на відміну від рекурентних мереж. У 2020 році було представлено модель Sign Language Transformers – першу систему, що навчалася одночасно розпізнавати безперервне жестове мовлення і перекладати його на текст [6]. Ця модель поєднала дві підзадачі (розпізнавання жестів як послідовності глосів і переклад глосів на речення) в єдиному енд-ту-енд процесі. Результати показали, що навіть за відсутності проміжної ручної транскрипції можна досягти зближення жестового і текстового простору, якщо мережа отримує достатньо даних і обчислювальних ресурсів. В подальших роботах для покращення якості перекладу застосовано методи transfer learning – попереднє

навчання компонентів моделі на великих суміжних датасетах (наприклад, на корпусах відео людських дій чи на багатомовних текстах) з подальшим донавчанням на даних жестової мови. Такий підхід дозволив значно підвищити точність перекладу на стандартних наборах даних, продемонструвавши силу мультимодальної інтеграції інформації.

Окрема категорія сучасних рішень – мультимодальні трансформери, що інтегрують візуальні і мовні дані. Приклади включають моделі на кшталт BERT (двонаправлена трансформерна модель для розуміння контексту природної мови), CLIP (модель від OpenAI, що навчається спільному простору «зображення–текст» і може зв'язувати зорові образи зі словами) та Flamingo (модель від DeepMind, здатна аналізувати комбінацію зображень і тексту). Хоча вони не створювалися спеціально для жестової мови, ідеї, закладені в них, активно переносяться в цю сферу. Зокрема, спільне представлення зорових і мовних ознак дозволяє системі «розуміти» значення жесту через зіставлення відео з відповідним текстом. Уже згадана модель Sign Language Transformers фактично є окремим випадком мультимодального трансформера, адаптованого під дані жестового відео і тексту. Очікується, що залучення переднавчених мультимодальних мереж (наприклад, використання CLIP для початкового розпізнавання значення жесту) може прискорити навчання та підвищити точність перекладу жестів у рази [7].

Приклади реалізацій. Хоча дослідницькі прототипи ще далекі від досконалості, у світі вже існують експериментальні системи перекладу жестової мови. Стартап SignAll розробив комплекс камер і програмного забезпечення, що відстежує рухи рук і тіла для перекладу американської жестової мови (ASL) на текст в реальному часі. Компанія Google експериментує з проєктом Sign Language Detection у межах платформи MediaPipe: нещодавно було представлено вебдодаток SignTown, що навчає користувача жестів і одночасно розпізнає їх через вебкамеру на основі моделі TensorFlow.js. Такі проєкти демонструють можливості сучасного штучного інтелекту, проте поки що мають обмеження: більшість підтримує обмежений набір жестів та окремих фраз, і рідко виконує повноцінний синтаксичний аналіз жестового мовлення чи зворотний переклад (тобто генерацію жестів з тексту). Іншими словами, нинішні системи в основному розпізнають жестові вислови і видають підпис або озвучку, тоді як повноцінний двосторонній перекладач (жести ↔ текст/мова) ще знаходиться на стадії розробок і тестування.

Перспективні напрями та міждисциплінарні підходи

Фахівці сходяться на думці, що для створення повноцінної системи двостороннього перекладу між жестовою та звуковою мовами необхідна інтеграція кількох компонентів. Основними складовими такого рішення є:

1. Точне візуальне розпізнавання жестів. Необхідне надійне відстеження та інтерпретація рухів мовця у відеопотоці. Для цього залучаються моделі 3D-позиціонування суглобів і скелетної анімації (motion capture), які перетворюють рухи рук та тіла на послідовності координат (так звані ландмарки). Наприклад, прототип від дослідників із Німеччини дозволяє автоматично отримувати рух скелету з відео жестів, а далі керувати 3D-аватаром на основі цих даних. Сучасні алгоритми глибокого навчання візуально розпізнають навіть дрібні жести пальців, що є критично важливим для мов жестів з розвиненою дактилологією (спелінгом).

2. Семантичне розуміння і переклад. Жестова мова, як і будь-яка інша, потребує розуміння контексту та побудови граматично правильного речення цільовою мовою. Тому до систем перекладу інтегруються модулі обробки природної мови (NLP). Вони виконують роль «мозку», що перетворює послідовність розпізнаних елементів (глосів чи інших

інтермодальних представлень) у зв'язний текст [8]. Використання трансформерів на цьому етапі дозволяє врахувати контекст жестового вислову та вирішити проблему відсутності прямих відповідників між жестами і словами. По суті, ця складова є класичним завданням машинного перекладу, де вхідна «мова» – це послідовність жестових символів, а вихідна – речення природною мовою. Важливою є й зворотна задача: синтез мовлення жестами з тексту, що вимагає генерації такої послідовності жестів, яка б адекватно передала зміст фрази вихідною жестовою мовою.

3. Аватари та зворотний переклад. Для виведення перекладу з тексту назад у жестову форму застосовуються 3D-аватари – анімовані персонажі, здатні виконувати рухи, еквівалентні жестам. Синтез відео з аватаром, який «розмовляє» жестами, дозволяє доносити інформацію до глухих користувачів в їх рідній жестовій мові. Сьогодні існують експериментальні системи, де аватар відображається у реальному середовищі через окуляри доповненої реальності, поруч із мовцем. Так, прототип німецької системи показав, що аватар на AR-окулярах може жестово перекладати мовлення співрозмовника в режимі реального часу, дозволяючи глухій людині одночасно бачити й обличчя мовця, і переклад жестами. Хоча поточна якість такої анімації має недоліки (наприклад, спотворення форми кистей або неприродні рухи пальців, технологія швидко вдосконалюється. Аватари дають змогу кастомізувати переклад: наприклад, відображати емоції через міміку чи змінювати стиль жестів відповідно до побажань користувача. Персоналізовані моделі можуть враховувати індивідуальний стиль жестикулювання конкретної людини або регіональні особливості жестової мови, що підвищить точність та природність перекладу.

4. Доповнена реальність (AR) і носимі пристрої. Доповнена реальність є багатообіцяючим напрямом для підвищення доступності перекладу жестів. Уявімо окуляри або мобільний додаток, які накладають субтитри чи анімованого жестового аватара поверх реального зображення співрозмовника. Такі рішення можуть забезпечити миттєвий переклад: глуха людина бачить переклад того, що говорить співрозмовник, а чуюча людина — субтитри того, що «говорить» жестами глухий співрозмовник. Проєкти на стику машинного навчання та AR вже досліджуються: зокрема, впровадження вищезгаданих 3D-аватарів у смарт-окуляри, інтеграція з голосовими асистентами для автоматичного розпізнавання мовлення і його перекладу на жести в реальному часі. Це міждисциплінарне поле на перетині комп'ютерного зору, мовознавства, дизайну взаємодії та носимих пристроїв потребує тісної співпраці фахівців різних галузей.

5. Етичні та соціальні аспекти. Розробляючи системи перекладу жестової мови, важливо враховувати етичні виклики. По-перше, це приватність: відеодані з жестами містять біометричну інформацію про людину, тому їх передача і зберігання повинні бути захищеними. Один із підходів – здійснювати обробку жестів локально на пристрої користувача (напряму в браузері), без відправлення відео на зовнішні сервери. По-друге, необхідно залучати до розробок саму спільноту глухих. Технології не повинні нав'язувати жестовій мові правила звукових мов або замінювати собою живе спілкування. Натомість, мета – доповнити існуючі засоби комунікації, зробити інформацію більш доступною, зберігаючи при цьому унікальну культуру жестових мов. Також постає питання працевлаштування сурдоперекладачів: впровадження автоматичних систем не має залишити фахівців без роботи, а скоріше зняти навантаження з вирішення рутинних завдань, дозволивши зосередитись на тонкощах перекладу, які поки не під силу машині.

Висновки

Машинний переклад жестової мови – це складна міждисциплінарна задача, що об’єднує методи комп’ютерного зору, глибинного навчання, лінгвістики та знання про Deaf-спільноту. Сучасні технології вже демонструють багатообіцяючі результати у розпізнаванні жестів і навіть початковому перекладі у текст. Проте до появи універсального та надійного перекладача жестових мов ще потрібно подолати чимало викликів. Необхідні більші та різноманітні датасети, нові моделі, здатні врахувати усі аспекти жестової комунікації, а також узгоджені стандарти для представлення жестів у цифровій формі. Попереду – робота над підвищенням точності та швидкодії моделей, адаптацією їх до різних мов і користувачів, вирішенням етичних питань. Успішний розвиток цієї сфери здатен істотно підвищити якість життя мільйонів людей, забезпечивши їм більш доступний і зручний інформаційний простір та нові можливості для взаємодії у суспільстві.

Список використаних джерел

1. Camgoz, N. C., Hadfield, S., Koller, O., & Bowden, R. (2020). Sign Language Transformers: Joint End-to-end Sign Language Recognition and Translation. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 10023–10033.

2. Zhou, B., Wang, Q., Huang, X., & Wang, Z. (2021). Spatial-Temporal Graph Convolutional Network for Sign Language Recognition. Neurocomputing, 442, 253–263.

3. Chen, Y., Wei, F., Sun, X., Wu, Z., & Lin, S. (2022). A Simple Multi-Modality Transfer Learning Baseline for Sign Language Translation. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 5109–5119.

4. Yin, S., & Yang, X. (2022). A Comprehensive Survey of Sign Language Translation with Deep Learning. ACM Computing Surveys, 55(2), 1–36.

5. Koller, O. (2020). Quantitative Survey of the State of the Art in Sign Language Recognition. arXiv preprint arXiv:2008.09918.

6. Nguyen, L. T., Schicktanz, F., Stankowski, A., & Avramidis, E. (2021). Automatic generation of a 3D sign language avatar on AR glasses given 2D videos of human signers. Proceedings of the 1st International Workshop on Automatic Translation for Signed and Spoken Languages (AT4SSL@MT Summit), 71–81.

7. Google Research. (2023). Sign Language Recognition with MediaPipe. Google AI Blog. Retrieved from ai.googleblog.com .

8. World Federation of the Deaf. (2021). Sign Languages. Retrieved from https://wfdeaf.org

Відомості про автора:

Івашкевич Максим Євгенович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп’ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, жестова мова, машинний переклад.

E-mail: 7364262@stud.kai.edu.ua

УДК 004.896:681.5

ЗАСТОСУВАННЯ МАШИННОГО НАВЧАННЯ ДЛЯ СТАБІЛІЗАЦІЇ HEXAPOD-У ПРИ ПОШКОДЖЕННІ АБО ВТРАТІ КІНЦІВОК

Богдан КАВАЦЮК
Здобувач вищої освіти
1-го курсу магістратури кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ
Науковий керівник к.ф.-м.н., доцент кафедри ІПЗ ФКНТ
Олег Васильович Мороз

У роботі розглянуто застосування методів машинного навчання для забезпечення стійкості гексапод-робота при пошкодженні або втраті однієї чи кількох кінцівок. Проаналізовано класичні підходи (PID, rule-based) та сучасні ML-методи, зокрема reinforcement learning, згорткові нейронні мережі (CNN) та self-supervised learning. Показано переваги самонавчальних стратегій у виявленні нестійких станів і адаптації поведінки гексапода до змінених умов середовища. Зроблено порівняння ефективності різних підходів щодо швидкості адаптації та стійкості системи.

Ключові слова: *hexapod, стабілізація, машинне навчання, self-supervised learning, reinforcement learning, згорткові нейронні мережі, втрати кінцівок, адаптивність, балансування, робототехніка.*

Ціль роботи

Метою дослідження є вивчення та порівняння підходів до стабілізації руху гексаподів при пошкодженні кінцівок, а також виявлення ефективності використання методів машинного навчання для забезпечення адаптивності й живучості конструкції у непередбачуваних умовах.

Вступ

При створенні крокуючих роботів має враховуватись ризик розбалансування, зв’язаний з втратою, або пошкодженням однієї, або декількох кнцівок. Наразі стоїть питання розробки систем балансування при виходу з ладу кінцівок гексаподу, оскільки в такій ситуації алгоритм пересування буде кардинально змінюватись – потрібні алгоритми, які зможуть швидко перестроїтись під наявні можливості.

Матеріали та методи

Як і в будь якій сфері розробки ПЗ – зараз ведуться дискусії з ефективності класичних методів (PID, rule-based), які забезпечують стабільність у статичних умовах, проти машинного навчання, яке забезпечує гнучкість і адаптивність [1], хоча вимагає більших обчислювальних ресурсів і даних для тренування.

Основна проблема при пошкодженні кінцівок крокуючого робота – це втрата симетрії і зміна центру маси. Класичні алгоритми, засновані на заданих траєкторіях, ен можуть швидко врахувати зміни і адаптуватись до них. Натомість алгоритми Reinforcement Learning, як-от T-Resilience Algorithm [2], дозволяють гексаподу самостійно виробити новий стиль ходи під будь яку конфігурацію. Такий підхід self-modeling [3], де робот створює внутрішню

модель для перебудови стратегії руху, та нейроеволюційні методи (наприклад, NEAT) [4] є оптимальними для продовження функціонування, при втраті гексаподом 1 або 2 кінцівок.

Візуальна інформація може слугувати основним сенсорним джерелом у разі відмови тактильних або IMU-сенсорів. Зокрема, застосування згорткових нейронних мереж (CNN) у поєднанні з даними гіроскопів дозволяє передбачити втрату рівноваги. Методи vision-based gait adaptation [5], навчання в симуляції з перенесенням у реальний світ (Sim2Real) [6], а також self-supervised learning сприяють формуванню поведінки, здатної адаптуватися до змін. Зокрема, камера може ідентифікувати неефективну роботу кінцівки, після чого ML-модель коригує рух решти кінцівок.

Згорткові нейронні мережі (CNN) використовуються для аналізу візуальної інформації, отриманої з камери гексапода, з метою передбачення втрати рівноваги або неефективної роботи кінцівок. Модель CNN в режимі реального часу обробляє відеопотік і визначає, наприклад, коли одна з ніг волочиться або не має контакту з поверхнею, що сигналізує про втрату стійкості. У відповідь на це система коригує траєкторії інших кінцівок, зміщує центр маси або адаптує гейт відповідно до нових умов. CNN поєднуються з даними з IMU або гіроскопа в системах сенсорного злиття для підвищення точності оцінки стану робота. Такий підхід дозволяє зберегти баланс навіть у разі втрати однієї чи кількох кінцівок, підвищуючи живучість системи [7].

Self-supervised learning застосовується для автономного виявлення нестабільних ситуацій без необхідності зовнішнього маркування даних. Гексапод отримує зображення та сенсорні дані у процесі пересування і самостійно формує мітки, що вказують на успішну або невдалу стратегію руху. Ці мітки використовуються для подальшого донавчання моделі, яка з часом краще розпізнає ситуації втрати рівноваги та адаптує рух. Такий підхід дозволяє формувати інтуїтивне розуміння “стійкості” на основі минулого досвіду, без потреби ручного аналізу чи моделювання кожної можливої ситуації. Це значно знижує потребу в обсягах маркованих даних, роблячи адаптацію швидшою та ефективнішою в польових умовах [8].

Висновки

Порівняння класичних методів і ML-підходів показує, що rule-based алгоритми (наприклад, PID або FSM) забезпечують швидку реакцію та малу обчислювальну складність, але вимагають точного моделювання та не можуть ефективно адаптуватися до несподіваних ситуацій, таких як пошкодження кінцівки. Натомість методи машинного навчання, зокрема reinforcement learning, CNN та self-supervised learning, дозволяють гексаподам вчитись новим стратегіям руху на основі власного досвіду або візуального зворотного зв’язку, забезпечуючи вищу гнучкість і стійкість у непередбачуваних умовах.

Список використаних джерел

1. Şahin, M. (2020). Hexapod Robot Design and Performance Comparison of Fuzzy and PID Control Methods. *Balkan Journal of Electrical and Computer Engineering*, 8(3), 195–201. https://dergipark.org.tr/en/pub/bajece/issue/52149/650784

2. Cully, A., Clune, J., Tarapore, D., & Mouret, J.-B. (2015). Fast Damage Recovery in Robotics with the T-Resilience Algorithm. *Nature*, 521(7553), 503–507. https://arxiv.org/abs/1302.0386

3. Bongard, J., Zykov, V., & Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314(5802), 1118–1121.

4. Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2), 99–127.

5. Han, J., & Zhao, Y. (2021). Design of control system and motion analysis of vision capture for small hexapod robot. *MATEC Web Conf.*, 336, 03004. https://www.matec-conferences.org/articles/matecconf/abs/2021/05/matecconf_cscns20_03004/matecconf_cscns20_03004.html

6. Kasaei, S. H., et al. (2023). Versatile Locomotion Skills for Hexapod Robots. https://arxiv.org/abs/2412.10628

7. Ravi, D., et al. (2020). Deep Learning for Human Activity Recognition: A Resource Efficient Implementation on Low-Power Devices. IEEE Internet of Things Journal, 7(5), 4133–4145.

8. Jatavallabhula, K., et al. (2021). Learning Dexterous Manipulation with Self-Supervised Policy Adaptation from Human Videos. arXiv:2108.11572. https://arxiv.org/abs/2108.11572

Відомості про автора:

Кавацюк Богдан Олегович – здобувач вищої освіти 1-го курсу магістратури кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, згорткові нейронні мережі, машинне навчання.

E-mail: 6880781@stud.kai.edu.ua

УДК 004.75

МЕХАНІЗМИ МІЖПРОЦЕСНОЇ ВЗАЄМОДІЇ ДЛЯ РОЗПОДІЛЕНИХ ЗАСТОСУНКІВ В UNIX ПОДІБНИХ ОПЕРАЦІЙНИХ СИСТЕМАХ

Глєб КАСІМОВ
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник старший викладач кафедри ІПЗ ФКНТ
Марія Давидівна Васильєва

У статті розглядається поняття міжпроцесної взаємодії (IPC) як ключового елементу сучасних обчислювальних систем. Описано значення IPC для узгодженого виконання завдань між окремими процесами та її критичну роль у функціонуванні розподілених систем. Основна увага приділена аналізу механізмів IPC, доступних у UNIX-подібних операційних системах, а також обговоренню потенційних труднощів, що виникають при їх практичному застосуванні.

Ключові слова: *міжпроцесна взаємодія, IPC, socket, message queue, RPC, shared memory.*

Вступ

Міжпроцесна взаємодія (IPC) є фундаментальною концепцією в інформатиці, що представляє собою набір механізмів, які дозволяють декільком процесам або програмним компонентам в обчислювальній системі брати участь в обміні даними, синхронізації їх діяльності та узгодженому виконанні завдань. Ця можливість має важливе значення для побудови складних програмних систем, які вимагають узгодженої роботи різних частин. IPC полегшує обмін інформацією між розрізненими програмами, дозволяючи їм співпрацювати для досягнення спільної мети. У розподілених системах IPC слугує ядром, незамінним елементом, без якого розподілена робота була б неможливою. Вона забезпечує необхідну основу для процесів, потенційно розташованих на географічно різних машинах, для взаємодії та обміну інформацією через мережеві кордони. Здатність передавати повідомлення і синхронізувати дії між цими відокремленими компонентами має першорядне значення для функціональних можливостей розподілених додатків.

Ціль роботи

Мета статті - проаналізувати та описати різні механізми міжпроцесної взаємодії (IPC), доступні в UNIX-подібних операційних системах та потенційні складнощі в їх практичному застосуванні.

Матеріали та методи

UNIX-подібні операційні системи пропонують широкий спектр механізмів міжпроцесної взаємодії, кожен з яких має свої особливості та придатність для різних типів додатків, включаючи ті, що призначені для роботи в розподіленому середовищі.

Найпопулярнішим механізмом є сокети (sockets). Сокети слугують основними кінцевими точками для мережевого зв'язку, дозволяючи обмінюватися даними між процесами, які можуть знаходитися на одному або різних комп'ютерах. У UNIX-подібних системах сокети є універсальними та існують у різних типах, кожен з яких пристосований для конкретних комунікаційних потреб. Потокові сокети (stream sockets) забезпечують

надійний, впорядкований і орієнтований на з'єднання канал зв'язку, зазвичай використовуючи протокол TCP в домені Інтернету. Цей тип сокетів гарантує, що дані прибувають до місця призначення в тій же послідовності, в якій вони були відправлені, що робить його придатним для додатків, які вимагають високої надійності. З іншого боку, дейтаграмні сокети (datagram sockets) пропонують ненадійний метод зв'язку без з'єднання, часто використовуючи UDP в домені Інтернету. Хоча вони не гарантують доставку, дейтаграмні сокети, як правило, швидші і підходять для додатків, де допустима випадкова втрата даних або де швидкість має першорядне значення. Крім них, існують також доменні сокети Unix (Unix domain sockets). Вони забезпечують ефективний механізм для локального міжпроцесного зв'язку на одному хості. Вони використовують шляхи файлової системи як адреси і можуть запропонувати переваги у продуктивності порівняно з мережевими сокетами для локального зв'язку.

Іншим популярним механізмом є черги повідомлень (message queues). Черги повідомлень забезпечують асинхронний механізм зв'язку, за допомогою якого процеси можуть обмінюватися даними у вигляді дискретних повідомлень. UNIX-подібні системи пропонують два основних типи черг повідомлень: черги повідомлень System V та черги повідомлень POSIX. Черги повідомлень System V дозволяють процесам надсилати та отримувати повідомлення, причому кожне повідомлення може бути пов'язане з певним типом або пріоритетом. Ці черги управляються ядром і зберігаються навіть після виходу користувача з системи, що робить їх надійними для програм, які потребують гарантованої доставки повідомлень. Черги повідомлень POSIX пропонують альтернативний API з додатковими можливостями, включаючи ідентифікацію за іменем (рядком), а не за ключем, пріоритети повідомлень і можливість асинхронного сповіщення процесів про надходження повідомлення. У розподілених середовищах черги повідомлень є дуже цінними для забезпечення асинхронного зв'язку між відокремленими сервісами, підвищення масштабованості та надійності, а також забезпечення стійкості до нестабільного трафіку та збоїв компонентів.

Іншим механізмом міжпроцесної взаємодії є спільна пам’ять (shared memory). Спільна пам'ять забезпечує високоефективний засіб міжпроцесної взаємодії, дозволяючи декільком процесам безпосередньо звертатися до спільної області пам'яті. UNIX-подібні системи надають інтерфейси спільної пам'яті System V та POSIX. Ці сегменти зберігаються в ядрі і забезпечують швидкий обмін даними між процесами на одній машині. Однак доступ до спільної пам'яті вимагає явних механізмів синхронізації, таких як семафори, для запобігання гонок і забезпечення узгодженості даних. Хоча спільна пам'ять є переважно локальним механізмом IPC, ця концепція поширюється на розподілені середовища за допомогою розподіленої спільної пам'яті (Distributed Shared Memory, DSM). Системи DSM створюють віртуальний спільний адресний простір між комп'ютерами в мережі, прагнучи поєднати простоту програмування спільної пам'яті з масштабованістю розподілених систем.

Доволі популярним та швидким механізмом є віддалений виклик процедур (RPC). Віддалений виклик процедур (RPC) - це потужна технологія IPC, яка дозволяє програмі виконувати процедуру в іншому адресному просторі, часто на іншому комп'ютері в мережі, так, ніби це локальний виклик процедури. RPC спрощує розробку розподілених додатків, абстрагуючись від складнощів мережевої комунікації. Процес, як правило, включає в себе впорядкування параметрів на стороні клієнта, надсилання запиту до сервера, вилучення параметрів на сервері, виконання процедури, впорядкування результату та надсилання його назад клієнту для вилучення. Поширені реалізації RPC в UNIX-подібних системах

включають Sun RPC (ONC RPC), який став основою для NFS, та сучасні фреймворки, такі як gRPC. RPC широко використовується в розподілених додатках, таких як NFS, для обміну файлами, розподілених базах даних та архітектурах мікросервісів для міжсервісного зв'язку.

Використання механізмів IPC для розподілених додатків в UNIX-подібних операційних системах пов'язане з низкою проблем і вимагає ретельного врахування різних факторів для забезпечення належної функціональності, продуктивності та надійності. Однією з важливих проблем є серіалізація та десеріалізація даних у гетерогенних системах. У розподіленому середовищі додатки можуть працювати на машинах з різною архітектурою, що призводить до відмінностей у представленні даних. Щоб забезпечити безперебійну комунікацію, дані повинні бути перетворені в стандартний, узгоджений формат перед передачею, а потім перетворені назад у рідний формат після отримання. Існують різні інструменти і методи для вирішення цієї проблеми, в тому числі формати обміну даними, такі як JSON, протокольні буфери і XML, кожен з яких пропонує різні компроміси з точки зору ефективності, читабельності і мовної підтримки. Іншим важливим фактором є вплив мережевої затримки на продуктивність зв'язку. Розподілені додатки за своєю суттю передбачають комунікацію через мережу, що вносить затримки (латентність), які можуть суттєво вплинути на загальну продуктивність, особливо для синхронних механізмів IPC, таких як сокети RPC і TCP. Ця латентність стає частиною часу операції вводу/виводу, яку відчуває додаток, що потенційно призводить до повільного часу відгуку і погіршення користувацького досвіду. Щоб пом'якшити вплив мережевої латентності, розробники часто використовують асинхронні моделі зв'язку, такі як, наприклад, ті, що надаються за допомогою черг повідомлень, які дозволяють процесам надсилати повідомлення, не чекаючи на них відразу ж. Забезпечення відмовостійкості та обробка часткових збоїв також є першочерговим завданням у розподілених системах. Для досягнення відмовостійкості розподілені системи часто використовують резервування і реплікацію критично важливих компонентів і даних. Черги повідомлень можуть підвищити відмовостійкість, забезпечуючи надійну доставку повідомлень, навіть якщо деякі частини системи стають тимчасово недоступними. Системи RPC повинні впроваджувати механізми обробки мережевих помилок і збоїв сервера, такі як тайм-аути і повторні спроби.

Результати роботи та обговорення

Аналіз показує, що на вибір механізму IPC в розподілених UNIX-подібних системах сильно впливають конкретні вимоги програми, включаючи такі фактори, як моделі зв'язку (синхронний чи асинхронний), обсяг даних, чутливість до затримок, потреби в надійності та фізичний розподіл процесів. Сокети залишаються фундаментальним і універсальним вибором для мережевої розподіленої комунікації, в той час як черги повідомлень забезпечують надійну асинхронну комунікацію для відокремлених сервісів. Спільна пам'ять у поєднанні з семафорами забезпечує високопродуктивний обмін даними для спільно розміщених процесів. Фреймворки RPC базуються на механізмах IPC нижчого рівня, щоб забезпечити більш абстрактний і зручний для розробників підхід до розподілених обчислень. Тести продуктивності часто показують різні результати залежно від конкретного ядра операційної системи та характеру робочого навантаження, що підкреслює важливість врахування цих факторів під час проєктування та розгортання.

Висновки

Міжпроцесна взаємодія є критично важливим аспектом проєктування та реалізації розподілених додатків в UNIX-подібних операційних системах. Різноманітний набір доступних механізмів IPC пропонує унікальні характеристики та компроміси. Вибір

найбільш підходящого механізму залежить від глибокого розуміння комунікаційних вимог програми, цілей продуктивності та середовища розгортання. В той час як сокети забезпечують базовий рівень мережевої комунікації, абстракції вищого рівня, такі як черги повідомлень і фреймворки RPC, пропонують більш структуровані підходи для конкретних розподілених моделей. Оскільки розподілені системи продовжують розвиватися в складності та масштабі, глибоке розуміння цих базових механізмів IPC залишається важливим для побудови надійних та ефективних рішень.

Список використаних джерел

1. URL: https://www.lenovo.com/us/en/glossary/ipc/ (дата звернення: 24.04.2025)
2. URL: https://en.wikipedia.org/wiki/Network_socket (дата звернення: 24.04.2025)
3. URL: https://medium.com/@ahnafzamil/networking-101-what-are-sockets-7e2d274e153 (дата звернення: 24.04.2025)
4. URL: https://aws.amazon.com/message-queue/ (дата звернення: 24.04.2025)
5. URL: https://www.softprayog.in/programming/interprocess-communication-using-system-v-message-queues-in-linux (дата звернення: 24.04.2025)
6. URL: https://en.wikipedia.org/wiki/Shared_memory (дата звернення: 25.04.2025)
7. URL: https://www.geeksforgeeks.org/what-is-distributed-shared-memory-and-its-advantages/ (дата звернення: 25.04.2025)
8. URL: https://aws.amazon.com/compare/the-difference-between-rpc-and-rest/ (дата звернення: 25.04.2025)

Відомості про автора:

Касімов Глєб Юрійович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, міжпроцесорна взаємодія, розподілени системи.

E-mail: 7493499@stud.kai.edu.ua

UDK 004.42

INNOVATIONS IN SOFTWARE DEVELOPMENT FOR DIFFERENT PROCESSOR ARCHITECTURES: A COMPREHENSIVE COMPARATIVE ANALYSIS OF X86 VS ARM WITH DEVELOPER RECOMMENDATIONS

Maciej Kłoda, Miłosz Zając
B.Sc. Eng., B.Sc. Eng.
University of the National Education Commission, Krakow
Scientific supervisor: PhD Tetiana Konrad

This article presents a comprehensive comparative analysis of the x86 and ARM processor architectures in the context of modern software engineering challenges. Performance, energy efficiency, security, and developer tool availability are assessed. The study is based on real-world application tests and system benchmarks. Market forecasts and practical conclusions for developing multi-architecture applications are also presented. The article concludes with recommendations for engineers and a reflection on the future of CPU architectures up to 2028.

Keywords: *ARM, x86, benchmark, CPI, IPC, Docker, PAC, CET, AI, toolchain, containerization, optimization, multithreading, inference.*

Introduction

Introduction CPU architectures are the foundation of every computing system. For decades, the x86 architecture, developed by Intel and AMD, has dominated personal computers and servers. In recent years, however, ARM processors, initially present almost exclusively in smartphones, have begun expanding into the laptop, server, and even supercomputer markets. This trend has opened new opportunities for innovation but also presented developers with a range of challenges. The differences between x86 and ARM are fundamental. x86 is a CISC (Complex Instruction Set Computing) architecture characterized by a rich instruction set and high microarchitectural complexity. ARM, on the other hand, is RISC (Reduced Instruction Set Computing), focusing on simplicity and energy efficiency.

Theoretical Overview

Processor architectures are broadly categorized as RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing), reflecting distinct design philosophies. x86, a CISC architecture developed since the 1970s by Intel and AMD, aims to maximize instruction functionality through complex, multi-purpose operations. This approach results in advanced but intricate pipelines, extensive instruction decoders, and microcode reliance, offering high performance at the expense of greater power consumption and design complexity. ARM, in contrast, follows the RISC model established in the 1980s by Acorn Computers. It emphasizes streamlined, uniform instructions and simple execution paths, enabling faster execution, shorter pipelines, and efficient branch prediction.

The microarchitectural contrast is striking: x86 pipelines often exceed 20 stages and are supported by sophisticated speculative execution mechanisms, multi-level caches, and SMT (Simultaneous Multithreading). ARM adopts a modular structure, with a shorter pipeline, lower operating voltage, and heterogeneous cores (e.g., big.LITTLE) that balance power and performance. Many modern ARM chips also include integrated AI accelerators and dynamic power management features. ARM’s key advantage lies in its energy efficiency. Its lower TDP (Thermal Design Power)

is due to minimalistic instruction decoding, compact cores, and reduced memory overhead. For example, laptops equipped with Apple's M1 or M2 chips often consume half the power of comparable x86 systems while delivering equivalent performance. In server environments, ARM-based processors like AWS Graviton deliver a superior performance-per-watt ratio, significantly lowering data center operational costs. These architectural differences deeply influence how compilers behave. Tools like GCC, Clang, and LLVM must employ architecture-specific heuristics. ARM's fixed-length instruction encoding and reliance on register-based operations favor faster but sometimes limited optimizations. Meanwhile, x86 allows complex vector operations (e.g., SSE, AVX), but these require meticulous tuning and can increase binary size and compiler complexity. From an ecosystem perspective, x86 remains dominant in traditional desktop and server environments, particularly under Windows and Linux. ARM, originally prevalent in mobile devices, has expanded significantly into laptops and professional workstations, especially through Apple's transition to ARM-based silicon. While both Linux and macOS offer native support for both architectures, migrating applications between them can involve significant work—recompiling, adjusting low-level dependencies, and occasionally relying on emulation tools like Rosetta 2 or QEMU. Overall, ARM and x86 are shaped by fundamentally different goals. ARM emphasizes efficiency, modularity, and energy-conscious scalability, making it ideal for modern mobile, cloud, and embedded scenarios. x86, by contrast, excels in legacy compatibility, raw computational throughput, and mature tooling, making it better suited to traditional high-performance or general-purpose computing. The right choice depends not only on current requirements but also on future scalability and software lifecycle considerations.

Materials and Methods

Hardware configuration. For a comprehensive comparative analysis of ARM and x86 architectures, two distinct test environments were utilized: x86 System - CPU: Intel Core i7-12700K (12 cores: 8P + 4E, up to 5.0 GHz) | RAM: 32 GB DDR4 3200 MHz | OS: Ubuntu 22.04 LTS (kernel 6.2, x86_64) | Storage: SSD NVMe Gen4 (Samsung 980 PRO); System ARM - CPU: Apple M1 Pro (8 cores: 4P + 4E, up to 3.2 GHz) | RAM: 32 GB LPDDR5 | OS: macOS Ventura 13.5 (Darwin 22.5, aarch64) | Storage: Apple Silicon integrated SSD

Both systems were thoroughly cleaned and identically configured to ensure unbiased results: Automatic updates were disabled, Power-saving features and turbo boost were disabled (for power consumption measurements), Cache and tmpfs were cleared before each test, Each test iteration was repeated 10 times.

Measurement tools and methodology

For the measurements, we utilized a combination of open-source tools and built-in operating system utilities. These were carefully selected to ensure maximum consistency in measurement methodologies:

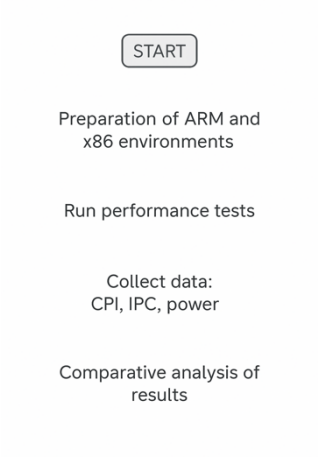
Tab 1

Overview of CPU Monitoring and Profiling Tools for ARM and x86

Category	ARM Tools	x86 Tools
Power Measurement	powermetrics (macOS)	powertop, perf
Timing	time -v, hyperfine	time -v, hyperfine
CPU Statistics	sysctl, top	perf stat, htop
Profiling	Activity Monitor, dtrace	perf, valgrind, gprof

Test Scope. The study encompassed the following test categories:

- Microarchitectural Tests: CPI, IPC, cycle count analysis, cache miss rate
- Power Efficiency Benchmarks: ldle power consumption, load scenarios (compilation, AI workloads, media streaming)
- Application Performance Tests: 7 real-word tools (Octave, FFmpeg, Blender, PostgreSQL, Node.js, Docker, Python)
- Architecture Migration Simulations: Running x86 applications on ARM (Rosetta 2), running ARM applications on x86 (QEMU)
- Compiler Performance: build time analysis for large projects (TensorFlow, GCC, FFmpeg)



Results and discussion

CPI and IPC Analysis

The CPI (Cycles per Instruction) and IPC (Instructions per Cycle) indicators allow for the evaluation of instruction processing efficiency. The ARM M1 achieved a CPI of 1.2 and IPC = 2.4, indicating high pipeline efficiency and good data locality. The x86 achieved a CPI of 1.6 and IPC = 2.1, which is the result of a more complex pipeline and a greater number of non-productive cycles.

ARM achieves higher efficiency under typical workloads – fewer cycles are needed to execute a single instruction.

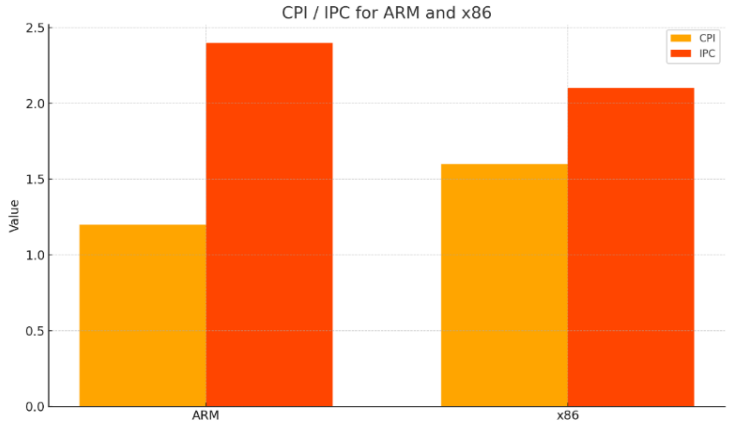


Fig. 1. Comparison of CPI / IPC for ARM and x86

Energy Consumption

Tests were conducted in four scenarios: idle, compilation (CMake + Clang), AI inference (PyTorch), and video streaming (ffplay). ARM consumed 30% to 50% less energy compared to x86. The differences were most significant in compute-intensive tasks.

Tab 2

Power Consumption Comparison Between ARM and x86 in Different Modes

Mode	ARM (W)	x86 (W)
Idle	4	9
Compilation	28	50
AI inference	18	35
Streaming	10	22

ARM demonstrates a significantly better performance-to-power ratio (Perf/Watt), making it more attractive in server environments.

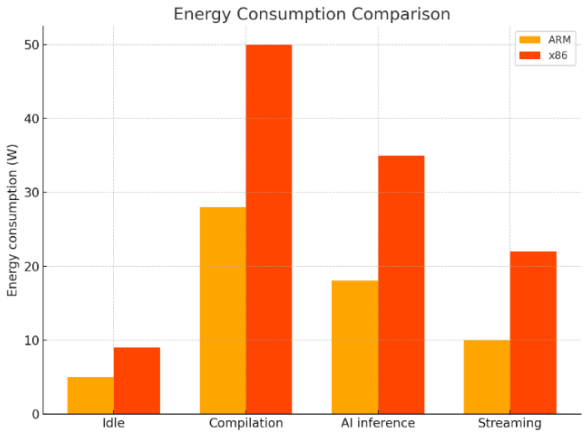


Fig. 2. Comparison of Energy Consumption on ARM and x86 architectures

Performance of actual applications

In comparative application benchmarks, x86 consistently outperformed ARM in compute-heavy tasks. Octave ran 20% faster on x86, FFmpeg benefited from AVX2 extensions, and Blender rendered scenes more quickly due to the higher number of performance cores. For I/O-bound workloads, ARM showed an edge—PostgreSQL performed slightly better on ARM. Node.js showed comparable performance across both architectures. Docker image builds were faster on x86, likely due to faster NVMe SSD access. In Python benchmarks, ARM was marginally slower. Overall, x86 led in raw performance, while ARM remained competitive, especially in I/O-intensive and web-based scenarios.

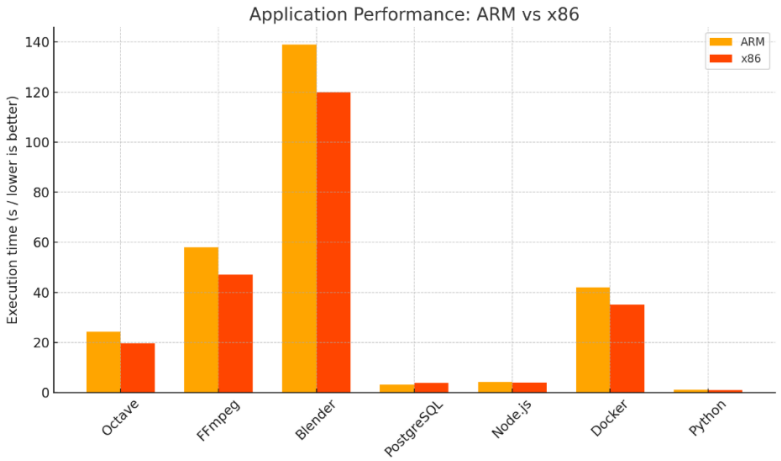


Fig. 3. Comparison of application execution time on ARM and x86 architectures (lower time = better performance)

Migration Tools and security

Running software across different architectures such as ARM and x86 involves challenges related to both compatibility and security. Native compilation delivers the best performance but requires architecture-specific builds and deeper integration efforts. Emulation tools like QEMU offer flexibility by enabling dynamic binary translation between architectures, although at the cost of significant performance overhead, which makes them more suitable for development and testing. Rosetta 2, Apple's real-time x86-to-ARM translation layer, stands out by offering seamless and

highly efficient execution of legacy applications on Apple Silicon devices. Security mechanisms differ significantly between the two platforms. ARM integrates modern, hardware-level protections such as Pointer Authentication Codes (PAC), Memory Tagging Extension (MTE), and Execute-Only Memory (XOM). These features are lightweight, require no software-based overhead, and offer robust defense against common exploits like buffer overflows or pointer tampering. PAC and MTE, in particular, are already widely deployed in systems like macOS and Android, and provide runtime enforcement of pointer integrity and memory access safety. x86, while older in architecture, has evolved with its own set of defenses. SMEP and SMAP prevent unauthorized kernel-to-user memory access, while Intel's Control-flow Enforcement Technology (CET), which includes Shadow Stack and Indirect Branch Tracking, defends against sophisticated control-flow attacks such as ROP and JOP. Compiler-based techniques like ASLR, stack canaries, and PIE/RELRO complement hardware protections but can vary in effectiveness depending on implementation. A simple buffer overflow vulnerability, for instance, is handled differently across these architectures. On ARM with PAC and MTE, execution is terminated immediately due to tag mismatch detection. On x86 with CET, the Shadow Stack intercepts unauthorized changes to return addresses. On legacy x86 systems lacking these protections, the same exploit could lead to full control of the system. In summary, ARM provides a forward-looking, integrated approach to software execution and security, making it ideal for modern, security-sensitive environments such as mobile and client devices. x86 continues to offer broad compatibility and strong control-flow protection, particularly within enterprise and legacy systems. Developers targeting both platforms must understand these differences to effectively deploy secure and portable applications.

Multithreading Performance and Scalability

Multicore performance depends not only on the number of cores, but also on their organization, thread management, interconnect bandwidth, and the operating system’s scheduling strategy. x86 architectures use symmetric cores with SMT (Simultaneous Multithreading), enabling multiple threads per core. ARM often follows a heterogeneous model like big.LITTLE, combining high-performance and energy-efficient cores. While ARM server chips (e.g., Ampere Altra) scale by increasing the number of physical cores without SMT, x86 scales through SMT and deeper cache hierarchies. Empirical tests confirmed that x86 outperforms ARM in multithreaded scaling scenarios—particularly up to 16 threads—due to SMT and higher L3 cache bandwidth.

Tab 3

Compilation task (FFmpeg, make -jN)

Architecture	4 cores	8 cores	16 cores
x86	122 s	69 s	43 s
ARM	130 s	72 s	45 s

In rendering tasks, ARM trails slightly but maintains solid performance with a high numer of physical cores.

Tab 4

Blender rendering (BMW27.blend)

Architecture	4 th	8 th	16 th
x86	185s	104s	61s
ARM	202s	118s	71s

Transactional throughput with PostgreSQL shows similar peak results, though ARM requires more physical threads due to the absence of SMT.

Tab 5

PostgreSQL transactional test (pgbench 1M tx)

Threads	1	4	8	16
TPS (x86)	1720	6540	9480	10300
TPS (ARM)	1600	6890	9550	10120

Operating system scheduling also plays a significant role. On ARM, energy-aware schedulers must properly differentiate between performance and efficiency cores to avoid mismatches in workload distribution. On x86 systems like Alder Lake, Intel Thread Director helps guide the scheduler by identifying core types. Misconfigured policies can lead to inefficient resource usage or latency fluctuations. In terms of thread density, x86 holds an advantage due to Hyper-Threading, which allows it to execute more logical threads on fewer physical cores. For instance, in the Python 3.10 pyperformance benchmark, Apple M1 (ARM) uses 10 physical cores for 10 threads, while Intel i7-12700K (x86) utilizes 8P+4E cores with HT for 20 logical threads. In conclusion, x86 is better suited for tasks that require maximum thread throughput, high single-thread performance, or benefit from mature SMT and cache structures—such as rendering, compilation, or scientific workloads. ARM, on the other hand, excels in highly parallel, scale-out environments like HTTP servers and NoSQL databases, offering energy efficiency and straightforward scaling via physical core count. For developers, optimizing multithreaded performance means understanding not just software logic but also the underlying core architecture and threading model of the target platform.

Containerization and Virtualization

Containerization and virtualization are central to modern software development, enabling applications to run consistently across hardware platforms. Both ARM (aarch64) and x86 (amd64) architectures support Docker and OCI-compliant containers. However, since container images are architecture-specific, developers often rely on Docker Buildx and QEMU to build multi-architecture images—typically from x86 hosts for ARM targets. This introduces a noticeable performance penalty during the build phase.

Tab 6

Container build time comparison (Flask + UWSGI + Gunicorn) CI/CD and development environments

Host architecture	Target architektury	Build time	Overhead
x86	amd64	35 s	–
x86	arm64 (QEMU)	109 s	+211%
ARM	arm64	38 s	+8%

While native ARM builds are fast and efficient, QEMU emulation introduces substantial latency, making it less suitable for production pipelines. CI/CD platforms such as GitHub Actions and GitLab CI support multi-architecture pipelines, but native ARM runners are rare, and emulated environments suffer from lower stability under load. Full virtualization further differentiates the platforms. x86 remains the leader in enterprise-grade solutions (VMware, Hyper-V), supporting a

wide range of guest OSes including Windows. ARM, in contrast, offers efficient lightweight virtualization with QEMU and KVM on Linux and macOS, but lacks official support for Windows VMs due to licensing and architecture limitations.

A real-world test of a containerized Node.js API reinforces this contrast. ARM achieves better energy efficiency, while x86 delivers faster cold starts and response times.

Tab 7

Node.js container test comparison

Architektura	Time cold start	Middle response time	CPU usage
x86 (amd64)	2.3 s	18 ms	14%
ARM (aarch64)	2.6 s	22 ms	11%

In summary, both platforms offer full container support, but x86 remains superior for virtualization-heavy workloads and broad system compatibility. ARM excels in energy-efficient, containerized deployments, particularly in Linux-based environments. Developers targeting multi-architecture systems must account for build strategies, compatibility testing, and CI/CD optimization.

Integration with AI accelerators and ecosystem of tools and libraries

Modern processor ecosystems increasingly rely on AI acceleration and broad tool support. ARM architectures – especially Apple Silicon and Ampere – offer integrated Neural Engines and NPUs that provide high-performance inference capabilities with low power consumption. These platforms support major AI frameworks like CoreML, TensorFlow Lite, Arm NN, and ONNX. In contrast, x86 accelerates AI workloads using CPU-based extensions like Intel DL Boost (AVX512-VNNI) and AMD XDNA, which are effective but less energy-efficient. In terms of development support, ARM now offers comprehensive toolchain compatibility (GCC, Clang, Rust, Go) and works with most contemporary libraries and frameworks, including TensorFlow, PyTorch, OpenCV, and Electron. However, certain legacy and SIMD-heavy libraries may still lack full ARM support. x86 maintains a mature and expansive ecosystem, with strong backward compatibility, precompiled binaries, and optimized support for enterprise and proprietary software. Overall, while ARM is rapidly catching up in both AI capabilities and developer tooling, x86 remains dominant in legacy-heavy environments. The choice depends on whether performance-per-watt and modern toolchain flexibility (ARM) or maximum compatibility and established workflows (x86) are the primary concern.

Market forecast (2024–2028)

Between 2024 and 2028, the processor landscape will undergo a major shift driven by ARM's rapid expansion, the maturation of RISC-V, and x86’s pivot toward hybrid designs. ARM is no longer limited to mobile or embedded use; it now powers high-performance laptops and desktops, with Apple’s M4 SoC and Microsoft's native Windows 11 support on ARM setting a new standard. Meanwhile, Intel and AMD are responding with AI-integrated SoCs and hybrid architectures (e.g., Lunar Lake, Zen 5C), narrowing the energy efficiency gap. By 2026, RISC-V becomes an ISO standard, marking a pivotal moment for open-source hardware. Its adoption in enterprise servers introduces a third player, appealing to countries and companies seeking independence from proprietary IP. The result is a heterogeneous ecosystem where ARM and x86 begin to coexist in production environments – ARM leading in scale-out workloads (microservices, cloud-native apps), x86 remaining strong in scale-up scenarios (databases, legacy systems). By 2028, modular SoCs featuring combinations of ARM and RISC-V cores will dominate new designs.

These chips offer flexibility in assigning roles to different processing units – CPU, GPU, NPU, and FPGA – enabling system-level optimization beyond the constraints of a single architecture. x86, while still relevant in enterprise and workstation contexts, will gradually lose its monopoly on general-purpose computing. Looking ahead, the industry moves toward architecture-neutral software development. Developers will increasingly rely on containers, cross-platform compilers, and adaptive CI/CD pipelines. Instead of a single dominant ISA, the choice of architecture will become a strategic element in optimizing performance, cost, and energy efficiency.

Conclusions

The comparison between ARM and x86 reveals that choosing a processor architecture is not merely a matter of raw performance but a strategic design decision with deep implications for energy efficiency, security, scalability, and long-term software maintenance. ARM excels in environments that prioritize efficiency, such as cloud-native applications, mobile systems, and AI inference at the edge. Its modern features like PAC and MTE offer robust hardware-level security, and its architecture scales well with many physical cores, particularly in server workloads. x86, in contrast, remains dominant in high-performance computing, virtualization-heavy scenarios, and legacy enterprise applications, benefiting from mature SMT implementations, broader OS support, and deep software compatibility. In AI tasks, ARM offers more efficient local inference, while x86 provides flexible server-side acceleration paths. Modern development practices now require software to be architecture-agnostic. Engineers must think beyond a single ISA – writing portable code, adopting multi-architecture container strategies, and configuring CI/CD pipelines that account for varying toolchains and target environments. In an increasingly heterogeneous hardware landscape, the ability to build adaptable, cross-platform systems is no longer optional but essential.

List of references

1. Intel® 64 and IA-32 Architectures Software Developer Manuals
2. Apple Platform Security Documentation
3. Docker Documentation: Multi-Architecture Builds
4. QEMU Documentation - Provides technical details on emulation and cross-architecture compatibility
5. Linux Kernel Documentation: Energy-Aware Scheduling

Information about the authors:

Maciej Kłoda first year student of the second degree of Technical and Computer Education at the University of the National Education Commission in Krakow. Interests: bartending, dancing, electronics and sound acoustics.

E-mail: maciejkloda@interia.pl

Miłosz Zając first year student of the second degree of Technical and Computer Education at the University of the National Education Commission in Krakow. Interests: Gaming, Sports, 3d printers and Electronics.

E-mail: miloszzajac908@gmail.com

УДК 004.8:004.056

ВИКЛИКИ У СФЕРІ ВЕРИФІКАЦІЇ ОРИГІНАЛЬНОСТІ ІНФОРМАЦІЇ ТА ДАНИХ В ЕПОХУ ГЕНЕРАТИВНОГО ШІ

Андрій КОНДРАТЮК
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Володимир Опанасович Талалаєв

У статті досліджено виклики, пов'язані з авторством, академічною доброчесністю, надійністю ШІ-згенерованих даних та ефективністю сучасних методів верифікації. Аналіз охоплює технічні, етичні та соціальні аспекти проблеми, а також окреслює напрями майбутніх досліджень і рішень.

Ключові слова: *генеративний ШІ, академічна доброчесність, верифікація, плагіат, авторство, упередження, дезінформація.*

Вступ

Поява генеративного штучного інтелекту (ШІ) спричинила революцію у створенні контенту в різних сферах, включаючи текст, код, зображення та відео. Цей технологічний стрибок відкриває безпрецедентні можливості, але також створює значні труднощі у встановленні оригінальності та автентичності інформації та даних [1]. Легкість, з якою ШІ може генерувати складний і реалістичний контент у великих масштабах, ускладнює розрізнення між створеним людиною та згенерованим ШІ [2]. Це становить пряму загрозу для усталених норм оригінальності, особливо в академічному та науково-дослідному контекстах [3]. Універсальність генеративного ШІ означає, що проблема верифікації є поширеною для різних типів контенту, роблячи її більш складною та всеосяжною.

Мета - дослідити багатогранність викликів у сфері верифікації оригінальності інформації та даних у цю нову епоху, з особливим акцентом на наслідки для наукових досліджень та академічної доброчесності. Висока якість і масштабованість контенту, згенерованого ШІ, є ключовими факторами, що посилюють проблему верифікації.

Ціль роботи

Основною метою цього звіту є аналіз ключових викликів у сфері верифікації оригінальності інформації та даних в епоху генеративного ШІ [4], з особливим акцентом на:

- вплив на академічну доброчесність та зростання плагіату та фабрикації досліджень за допомогою ШІ [5].
- власні обмеження та потенційні упередження моделей генеративного ШІ, які можуть впливати на надійність згенерованої інформації [1].
- ефективність та обмеження сучасних технологічних рішень, розроблених для виявлення контенту, згенерованого ШІ [3].
- нові загрози, що виникають через дезінформацію та діпфейки, згенеровані ШІ, для ширшої інформаційної екосистеми [6].
- поточні зусилля та майбутні напрямки у розробці надійних методів автентифікації та верифікації [7].

Мета полягає не лише у виявленні викликів, а й у категоризації та розумінні конкретних аспектів цих викликів у різних вимірах (академічному, технологічному, суспільному).

Матеріали та методи

Цей звіт базується на всебічному огляді та синтезі існуючої літератури та результатів досліджень, що стосуються генеративного ШІ та верифікації оригінальності інформації та даних. Основними матеріалами для цього звіту є підібраний набір фрагментів досліджень, надані користувачем, що включають академічні статті, галузеві звіти та новинні статті. Аналіз включав виявлення ключових тем, викликів та запропонованих рішень, пов'язаних з верифікацією оригінальності інформації та даних у контексті генеративного ШІ.

Результати та обговорення

Поширення генеративного штучного інтелекту (ШІ) створює фундаментальні виклики для процесів верифікації оригінальності цифрового контенту. Здатність сучасних моделей ШІ генерувати текст, зображення та інші дані, що важко відрізнити від створених людиною, розмиває традиційні межі авторства та автентичності, вимагаючи перегляду існуючих підходів до перевірки інформації.

Одним із ключових викликів є суттєве ускладнення визначення авторства. Контент, згенерований ШІ, часто демонструє рівень когерентності та стилістичної відповідності, що робить його нерозрізнюваним від людського [3]. Це не лише створює стимули для неправомірного представлення ШІ-згенерованого контенту як власної роботи, особливо в академічному середовищі, але й порушує концептуальні питання щодо самого поняття авторства. Коли ШІ виступає інструментом чи навіть співтворцем контенту, що відповідає формальним вимогам, традиційні уявлення про інтелектуальний внесок та власність потребують переоцінки.

Безпосереднім наслідком цього є підвищений ризик плагіату та академічної нечесності. Доступність та легкість використання інструментів генеративного ШІ значно знижують бар'єри для створення контенту без належного авторського внеску чи атрибуції [3]. Це становить загрозу для академічної доброчесності, потенційно знецінюючи оригінальні зусилля здобувачів освіти та дослідників [8]. Більше того, ця технологія може бути використана для масового виробництва низькоякісних псевдонаукових публікацій [8]. Широке використання ШІ може призвести до нормалізації плагіату, що вимагає адаптації освітніх стратегій та переходу до методів оцінювання, які акцентують на критичному мисленні, застосуванні знань та процесах, важких для імітації ШІ (наприклад, інтерактивні семінари, проєктний захист).

Важливою проблемою є також потенційна наявність неточностей та упереджень у контенті, згенерованому ШІ. Моделі навчаються на великих масивах даних, які неминуче містять помилки та системні упередження (bias), що можуть реплікуватися у вихідних даних [1]. Це особливо критично в наукових дослідженнях та медицині, де точність є першорядною. ШІ також схильний до "галюцинацій" – генерації правдоподібної, але фактично неправдивої інформації, включно зі сфабрикованими посиланнями [3]. Залежність від навчальних даних означає, що якість та об'єктивність результатів ШІ обмежені якістю та репрезентативністю цих даних, що підкреслює імператив ретельної верифікації та критичної оцінки, особливо щодо потенційних соціальних упереджень (гендерних, расових тощо).

Існуючі технології виявлення ШІ-згенерованого контенту мають значні обмеження. Їх точність залишається недостатньою, вони схильні до хибнопозитивних та хибнонегативних спрацьовувань, що робить їх ненадійним інструментом верифікації [3]. Дослідження

вказують на можливу упередженість цих інструментів щодо авторів, для яких мова контенту не є рідною [9]. Ефективність детекторів суттєво знижується при аналізі коротких текстів, і їх можна відносно легко обійти шляхом незначних модифікацій згенерованого тексту [10]. Постійний розвиток генеративних моделей створює динамічну "гонку озброєнь", де можливості генерації випереджають можливості детекції [3]. Надмірне покладання на такі інструменти є недоцільним та може призвести до помилкових звинувачень та підриву довіри [10].

Окремим викликом є верифікація автентичності мультимедійного контенту, зокрема реалістичних відео та аудіо, створених за допомогою ШІ (технологія "deepfake") [6]. Навіть для людини достовірне виявлення діпфейків є складним завданням, а автоматизовані системи детекції також демонструють обмежену ефективність, особливо проти нових або складних методів генерації. Це створює серйозні ризики поширення дезінформації, маніпуляцій громадською думкою, зокрема в політичному контексті, та шахрайства [6]. Протидія вимагає комплексного підходу, що поєднує технологічні рішення з підвищенням медіаграмотності суспільства.

У ширшому контексті, проблеми з верифікацією оригінальності підривають загальну довіру до цифрової інформації та інформаційної екосистеми загалом [6]. Цей феномен, іноді означений як "занепад правди" ("truth decay"), може мати далекосяжні негативні наслідки для суспільних процесів, включаючи демократичні вибори та охорону здоров'я [6]. Ерозія довіри створює парадоксальну ситуацію ("дивіденди брехуна"), коли навіть автентичний контент може сприйматися зі скепсисом, що полегшує зловмисникам дискредитацію правдивої інформації [11]. В академічній сфері потенційна можливість фабрикації досліджень [5] ставить під загрозу цілісність наукового процесу та довіру до наукової спільноти, що вимагає адаптації процедур рецензування для виявлення маніпуляцій, які можуть бути непомітними для стандартних інструментів.

Незважаючи на ці виклики, активно ведуться дослідження та розробка стратегій для покращення верифікації. Це включає створення більш досконалих методів детекції ШІ, що аналізують глибинні лінгвістичні патерни, семантичну когерентність та статистичні властивості тексту [12]. Розглядаються технології автентифікації контенту та відстеження його походження (provenance), такі як цифрове водяне маркування (watermarking) та використання блокчейн-технологій [13]. В академічному середовищі пропонується впровадження стандартів метаданих та систем відстеження для документування використання ШІ при підготовці публікацій [2]. Загальний консенсус схиляється до необхідності багатошарового підходу, що поєднує технологічні інструменти з експертною оцінкою людини та розвитком навичок критичного мислення [12]. Ефективне вирішення проблеми вимагає синергії технічних, освітніх та нормативно-правових заходів [14].

Висновки

Генеративний ШІ радикально ускладнює верифікацію оригінальності інформації, створюючи значні ризики для академічної доброчесності, достовірності даних та довіри до цифрового середовища. Ефективне вирішення цієї проблеми вимагає комплексного підходу, що поєднує розробку досконаліших технологій детекції та автентифікації з посиленням людської експертизи, критичної оцінки та впровадженням відповідних політик.

Список використаних джерел:

1. Concerns about Generative Artificial Intelligence | Alliant International University Center for Teaching Excellence. Alliant International University Center for Teaching Excellence.

URL: https://cte.alliant.edu/concerns-about-generative-artificial-intelligence/ (дата звернення: 15.04.2025).

2. Ethical Challenges and Solutions of Generative AI: An Interdisciplinary Perspective / M. Al-kfairy et al. Informatics. 2024. Vol. 11, no. 3. URL: https://doi.org/10.3390/informatics11030058.

3. Elali FR, Rachid LN: Елалі Ф. Р., Рачід Л. Н. AI-generated research paper fabrication and plagiarism in the scientific community. Patterns (N Y). 2023. Т. 4, № 3. Art. 100706. DOI: 10.1016/j.patter.2023.100706.

4. Майовський М. та ін. Artificial Intelligence Can Generate Fraudulent but Authentic-Looking Scientific Medical Articles: Pandora's Box Has Been Opened. J Med Internet Res. 2023. Т. 25. Art. e46924. DOI: 10.2196/46924. URL: https://www.jmir.org/2023/1/e46924.

5. Кім С. Дж. Research ethics and issues regarding the use of ChatGPT-like artificial intelligence platforms by authors and reviewers: a narrative review. Sci Ed. 2024. Т. 11, № 2. С. 96-106.

6. Аль-Кфайрі М., Мустафа Д., Кшетрі Н., Інсью М., Альфанді О. Ethical Challenges and Solutions of Generative AI: An Interdisciplinary Perspective. Informatics. 2024. Т. 11, № 3. Art. 58. DOI: https://doi.org/10.3390/informatics11030058.

7. Building a Digital Content Authentication Research Ecosystem / Federation of American Scientists. URL: https://fas.org/publication/digital-content-authentication-ecosystem/ (дата звернення: 28.04.2025).

8. Originality.ai blog post: AI for Research Papers: Pros and Cons. Originality.ai blog. 07.02.2024. URL: https://originality.ai/blog/ai-for-research-pros-cons

9. Лі Ф., Ян Ю. Impact of Artificial Intelligence–Generated Content Labels On Perceived Accuracy, Message Credibility, and Sharing Intentions for Misinformation: Web-Based, Randomized, Controlled Experiment. JMIR Form Res. 2024. Т. 8. Art. e60024. DOI: 10.2196/60024. URL: https://formative.jmir.org/2024/1/e60024.

10. ITI Policy Recommendations for Authenticating AI-Generated Content / Information Technology Industry Council. 21.12.2023. URL: https://www.itic.org/policy/ITI_AIContentAuthorizationPolicy_122123.pdf (дата звернення: 01.02.2024).

11. Бхосале У. Measures to Authenticate AI-Generated Scholarly Output. Enago Academy. URL: https://www.enago.com/academy/authentication-of-ai-generated-scholarly-output/ (дата звернення: 10.03.2025).

12. AI Generated Content and Academic Journals. Daily Nous. 24.06.2024. URL: https://dailynous.com/2024/06/24/ai-generated-content-and-academic-journals/ (дата звернення: 30.04.2025).

13. Al writing in journals: Preserving research integrity. Turnitin Blog. URL: https://www.turnitin.com/blog/ai-writing-in-academic-journals-mitigating-its-impact-on-research-integrity (дата звернення: 22.04.2025).

Відомості про автора:

Кондратюк Андрій Олександрович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: програмна інженерія, штучний інтелект (ШІ), SaaS.

E-mail: 7545268@stud.kai.edu.ua

УДК 004.7:004.415

КОНТЕЙНЕРИЗАЦІЯ ЯК ОСНОВА ДЛЯ МАСШТАБОВАНИХ ХМАРНИХ РІШЕНЬ

Станіслав КОРЧЕВСЬКИЙ
Олександра ЛЯШЕНКО
Здобувачі вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник к.т.н., професор кафедри ІПЗ ФКНТ
Андрій Іванович Гізун

У статті розглядаються сучасні виклики, пов'язані з ефективною розробкою, розгортанням та масштабуванням програмного забезпечення в умовах стрімкого розвитку цифрових технологій. Зазначено обмеження традиційних підходів до управління програмними системами, зокрема у контексті складності, неефективного використання ресурсів та проблем сумісності середовищ. Основну увагу приділено контейнеризації як рішенню, що дозволяє забезпечити ізоляцію, портативність, ефективне горизонтальне масштабування та оптимальне використання ресурсів.

Ключові слова: *хмарні рішення, контейнеризація, Docker, масштабованість, Kubernetes, оркестрація.*

В умовах стрімкого розвитку цифрових технологій та зростаючих вимог до програмного забезпечення, питання ефективної розробки, розгортання та масштабування додатків стає критично важливим. Сучасні програмні системи характеризуються складною архітектурою, численними залежностями та потребою у швидкій адаптації до змінних навантажень.

Традиційні підходи до розгортання та масштабування ПС стикаються з низкою суттєвих обмежень та проблем: складністю цих процесів; неефективним використанням ресурсів в умовах пікових навантажень; можливими проблемами через відмінності між середовищами/системами. Це нерідко призводять до підвищення операційних витрат, зниження надійності та доступності сервісів.

Комплексним рішенням цих проблем є контейнеризація, яка забезпечує ефективне використання обчислювальних ресурсів, швидке розгортання в будь-якому середовищі, ізоляцію компонентів та гнучке масштабування.

Метою цього дослідження є аналіз ролі контейнеризації як ключового компонента в побудові масштабованих хмарних рішень, визначення основних переваг та особливостей використання контейнерних технологій для забезпечення ефективного горизонтального масштабування сучасних хмарних інфраструктур.

Контейнер - це стандартизована одиниця програмного забезпечення, яка "пакує" код застосунку разом з усіма необхідними залежностями (системними інструментами, бібліотеками, конфігураційними файлами, тощо.). Зазвичай, контейнери є легкими, портативними та, щонайголовніше – ізольованими [1].

Принцип роботи контейнерів полягає у використанні механізмів віртуалізації на рівні операційної системи, а саме дві ключові технології ядра Linux:

- Namespaces - забезпечують ізоляцію процесів, створюючи для кожного контейнера власний простір для процесів, мережі, файлової системи та користувачів;

- Cgroups - забезпечують обмеження та облік використання ресурсів процесора, пам'яті, мережевого трафіку та операцій з читання і запису з диску.

Віртуальні машини, в свою чергу, ізольовані від операційної системи та повністю емулюють апаратне забезпечення, яке використовується для роботи гостьової операційної системи.

Контейнеризація та віртуальні машини є двома підходами до ізоляції та управління програмними середовищами, схеми роботи яких наведені на рисунку 1, але вони мають суттєві відмінності. Контейнери забезпечують віртуалізацію на рівні операційної системи, використовуючи ядро хост-системи, що робить їх легшими, швидшими для запуску та більш ефективними у використанні ресурсів порівняно з віртуальними машинами, які вимагають повної копії гостьової ОС і працюють через гіпервізор. Завдяки цьому контейнери ідеально підходять для сучасних DevOps-практик, таких як CI/CD, забезпечуючи портативність та консистентність між середовищами [2,3].

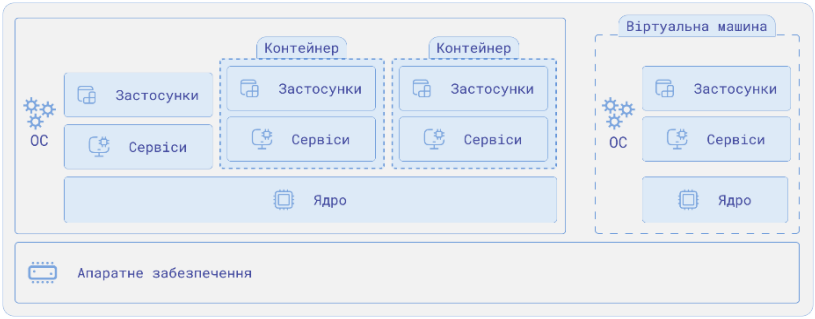


Рис. 1 – Схема роботи контейнерів та віртуальної машини

Контейнери все ж мають певні обмеження, зокрема меншу ізоляцію порівняно з віртуальними машинами та залежність від ядра хост-системи, що може створювати ризики безпеки при некоректній конфігурації. Незважаючи на це, їх переваги, такі як швидкість, ефективність та гнучкість, роблять контейнеризацію ключовою технологією для побудови масштабованих хмарних рішень.

Таблиця 1

Порівняльна характеристика ефективності

Характеристика	Віртуальні машини	Контейнери (Docker + Kubernetes)
Час розгортання	Повільніший (хвилини). Завантаження повної ОС займає певний час.	Швидший (секунди). Запускається як звичайний процес
Використання ресурсів	Високі	Низькі
Щільність ресурсів	Нижча, кожна ВМ потребує окремої ОС	Вища, багато контейнерів на одній ОС
Витрати на інфраструктуру	Вищі, ВМ потребують потужнішого обладнання	Нижчі, 30-40% економії
Операційні витрати	Вищі	Нижчі, 20-30% економії
Портативність	Низька	Висока
Масштабованість	Відносно повільна	Швидша, легко автоматизувати
Доступність сервісів	Висока, проте відновлення може займати хвилини	Вища (HA, auto-healing), адже відновлення відбувається за секунди

Docker став де-факто стандартом контейнеризації завдяки своїй простоті та потужному набору інструментів, тоді як Kubernetes (K8s) доповнює цю екосистему, вирішуючи проблему управління контейнеризованими додатками та їх масштабування. K8s забезпечує гнучкість, надійність і ефективність управління контейнерною інфраструктурою, реалізуючи такі критично важливі функції як балансування навантаження, самовідновлення після збоїв та автоматичне масштабування [4].

Станом на 2025 рік, контейнеризація стала стандартною практикою в IT-індустрії, а Kubernetes утвердився як де-факто стандарт для оркестрації контейнерів. Це підтверджується щорічними опитуваннями фахівців з хмарних технологій, які демонструють зростаючу тенденцію до впровадження контейнеризації та використання Kubernetes для забезпечення масштабованості, гнучкості та надійності хмарних рішень [5].

Список використаних джерел

1. Kane S.P., Matthias K.Docker: Up & Running. Sebastopol: O'ReillyMedia, 2024. 393 p.
2. URL:https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm (дата звернення: 17.04.2025)
3. URL:https://aws.amazon.com/compare/the-difference-between-containers-and-virtual-machines/ (дата звернення: 17.04.2025)
4. Burns B., Beda J., Hightower K., Evenson L. Kubernetes: Up and Running. 3rd ed. Sebastopol: O'Reilly Media, 2022. 309 p.
5. URL:https://www.cncf.io/reports/cncf-annual-survey-2024/ (дата звернення: 17.04.2025)

Відомості про авторів:

Корчевський Станіслав Миколайович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, хмарні технології, контейнерізація.
E-mail: 7299348@stud.kai.edu.ua

Ляшенко Олександра Сергіївна – здобувачка вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, хмарні технології, контейнерізація.
E-mail: 7331491@stud.kai.edu.ua

УДК 004.77:159.98(043.2)

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ IT-ФАХІВЦІВ ПІД ЧАС СПІВБЕСІД: РОЗРОБКА ВЕБДОДАТКУ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ОЦІНЮВАННЯ

Денис КОЧУБЕЙНИК
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Вікторія Олексіївна Волкогон

У роботі представлено розробку вебдодатку для автоматизованого тестування IT-фахівців під час співбесід. Система спрямована на оптимізацію процесу оцінювання технічних навичок кандидатів шляхом стандартизації тестування, автоматичної перевірки відповідей і формування детальних звітів. У розробці використано технології ASP.NET, Blazor, MS SQL та RESTful API. Проведено аналіз існуючих рішень, визначено вимоги до системи та протестовано її працездатність. Отримані результати демонструють підвищення швидкості, об'єктивності та ефективності найму IT-спеціалістів. Також окреслено перспективи подальшого розвитку, включаючи використання адаптивного тестування та технологій штучного інтелекту.

Ключові слова: *автоматизоване тестування, вебдодаток, оцінювання IT-фахівців, ASP.NET, Blazor, RESTful API, база даних MS SQL, технічна співбесіда, найм спеціалістів, адаптивне тестування.*

Вступ

У сучасному світі IT-індустрія є однією з найбільш конкурентних сфер працевлаштування. Компанії стикаються з викликом швидкої та об'єктивної оцінки технічних навичок кандидатів під час співбесід. Традиційні методи тестування, такі як усні запитання або письмові завдання, часто вимагають значних ресурсів, є суб'єктивними та не завжди точно відображають компетенції претендентів.

Автоматизовані системи тестування стають все більш популярними, оскільки вони дозволяють стандартизувати оцінювання, скоротити час найму та підвищити точність відбору спеціалістів. У межах цього дослідження було розроблено вебдодаток, який надає можливість проводити тестування IT-фахівців у режимі онлайн, оцінюючи їх знання з програмування, алгоритмів, баз даних та інших технологій, що відповідають вимогам конкретної вакансії.

Даний вебдодаток включає систему гнучкого налаштування тестів, інтеграцію з базами вакансій та можливість автоматичного аналізу результатів, що допомагає рекрутерам і технічним спеціалістам приймати обґрунтовані рішення. У доповіді буде розглянуто ключові особливості розробки, використані технології, а також переваги та перспективи впровадження автоматизованих тестувань у процес найму IT-фахівців.

Ціль роботи

Метою даного дослідження є розробка вебдодатку для автоматизованого тестування IT-фахівців під час співбесід, що дозволить оптимізувати процес оцінювання знань кандидатів та підвищити об'єктивність прийняття рішень при наймі. Також важливо розглянути можливість розвитку подібного застосунку.

Сучасний ринок праці у сфері інформаційних технологій вимагає швидких і точних методів оцінки компетенцій, адже традиційні підходи, зокрема усні технічні інтерв'ю або ручне перевіряння завдань, є трудомісткими, суб'єктивними та не завжди забезпечують коректну оцінку рівня знань кандидата. Використання автоматизованої системи тестування дозволяє стандартизувати процес перевірки, мінімізувати вплив людського фактору та скоротити час найму.

Основні завдання, які стоять перед розробленим вебдодатком:

- створення гнучкої платформи для проведення тестувань із можливістю налаштування тестів відповідно до вимог вакансій;
- впровадження механізмів автоматичної перевірки відповідей для підвищення швидкості оцінювання;
- забезпечення інтеграції з базами даних про вакансії та кандидатів для ефективного управління процесом відбору;
- розробка аналітичного модуля для генерації детальних звітів про результати тестувань;
- забезпечення зручного інтерфейсу як для кандидатів, так і для рекрутерів та технічних інтерв'юерів.

Очікується, що впровадження розробленого вебдодатку дозволить значно підвищити якість найму IT-спеціалістів, зменшити навантаження на рекрутерів і технічних спеціалістів, а також зробити процес оцінювання більш прозорим і ефективним.

Матеріали та методи

У процесі розробки вебдодатку для автоматизованого тестування IT-фахівців під час співбесід було використано комплексний підхід, що включає аналіз сучасних методів оцінювання технічних навичок, вибір оптимальних технологій та тестування працездатності розробленого рішення.

Дослідження розпочалося з аналізу існуючих платформ для тестування IT-спеціалістів, таких як HackerRank, Codility та TestDome, з метою визначення ключових функціональних можливостей і виявлення їхніх переваг та недоліків. Було розглянуто основні методи технічного тестування, зокрема завдання з вибором відповіді, кодувальні завдання, алгоритмічні задачі та логічні тести.

На основі проведеного аналізу було сформовано вимоги до вебдодатку, які включають швидкість оцінювання, об'єктивність результатів, масштабованість та зручність використання. Архітектура системи спроєктована на основі RESTful API, що забезпечує ефективну взаємодію між клієнтською та серверною частинами. Для реалізації серверної логіки використано ASP.NET, що дозволяє досягти високої продуктивності та безпеки. База даних, створена на основі MS SQL, використовується для зберігання тестових завдань, відповідей кандидатів та історії проходження тестувань.

Інтерфейс розроблено за допомогою Blazor, що дозволяє створити динамічний та інтерактивний вебдодаток із зручною взаємодією для користувачів. Проведено тестування працездатності системи, включаючи модульне та інтеграційне тестування, а також юзабіліті-тестування із залученням тестової групи користувачів.

Розроблений вебдодаток забезпечує ефективну автоматизацію процесу оцінювання IT-фахівців, спрощує роботу рекрутерів і технічних спеціалістів, зменшує витрати часу на перевірку кандидатів та підвищує об'єктивність прийняття рішень під час співбесід.

Результати та обговорення

У результаті проведеного дослідження було розроблено вебдодаток, що дозволяє

автоматизувати процес тестування IT-фахівців під час співбесід, значно підвищуючи швидкість і точність оцінювання кандидатів. Реалізована система забезпечує створення та налаштування тестових завдань різних типів, автоматичну перевірку відповідей та формування детальних звітів про результати. Завдяки використанню RESTful API взаємодія між клієнтською та серверною частинами стала ефективною, а інтеграція MS SQL забезпечила надійне зберігання та управління даними. Використання ASP.NET дозволило досягти високої продуктивності серверної частини, тоді як Blazor значно спростив розробку інтерактивного інтерфейсу, покращуючи взаємодію користувачів із системою.

Разом з тим, у процесі дослідження було виявлено кілька аспектів, які потребують подальшого вдосконалення. Зокрема, подальший розвиток системи може включати адаптивні алгоритми тестування, що дозволять динамічно змінювати складність питань залежно від відповідей кандидата. Також перспективним напрямком є впровадження штучного інтелекту для аналізу відповідей на відкриті питання та виявлення шаблонів у відповіді кандидатів, що допоможе ще точніше визначати рівень їхньої кваліфікації.

Висновки

Розробка вебдодатку для автоматизованого тестування IT-фахівців під час співбесід дозволила значно оптимізувати процес оцінювання кандидатів, зробивши його швидшим, об'єктивнішим та менш ресурсозатратним. Система забезпечує зручність використання, точність перевірки та можливість адаптації до вимог різних вакансій. Використання сучасних технологій, зокрема ASP.NET, Blazor, MS SQL та RESTful API, сприяло створенню продуктивного, масштабованого та безпечного рішення, яке може бути інтегроване в процес найму IT-спеціалістів у компаніях різного масштабу.

Подальші дослідження можуть бути спрямовані на розширення функціоналу системи, включаючи адаптивне тестування, використання машинного навчання для аналізу відповідей та розширення можливостей інтеграції з іншими HR-інструментами. Успішне впровадження таких рішень сприятиме підвищенню ефективності найму та якості оцінки технічних навичок кандидатів, що є важливим завданням для сучасного IT-ринку.

Список використаних джерел

1. HackerRank. About Us. URL: https://www.hackerrank.com/about-us (дата звернення: 20.03.2025).

2. Codility. URL: https://www.codility.com (дата звернення: 20.03.2025).

3. TestDome. URL: https://www.testdome.com/library (дата звернення: 20.03.2025).

4. Microsoft. ASP.NET Overview. URL: https://learn.microsoft.com/en-us/aspnet/overview (дата звернення: 20.03.2025).

5. Microsoft SQL Server. Documentation. URL: https://learn.microsoft.com/en-us/sql/sql-server (дата звернення: 20.03.2025).

6. Blazor. Documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor (дата звернення: 20.03.2025).

7. Nielsen J. Usability Engineering. San Francisco: Morgan Kaufmann, 1993. 362 p.

Відомості про автора:

Кочубейник Денис Миколайович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, міжпроцесорна взаємодія, розподілени системи.

E-mail: 7493499@stud.kai.edu.ua

УДК 004.415.53

ІНТЕГРОВАНИЙ ПІДХІД ДО АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ UI ТА API ВЕБЗАСТОСУНКІВ ЗА ДОПОМОГОЮ PLAYWRIGHT

Анастасія КУДРЯ
Здобувачка вищої освіти 4 курсу кафедри ІПЗ
Наталія ШИБИЦЬКА
к.т.н., доцент кафедри ІПЗ ФКНТ
Державний університет «Київський авіаційний інститут»

У статті досліджено інтегрований підхід до автоматизованого тестування вебзастосунків з одночасною перевіркою інтерфейсу користувача (UI) та серверної логіки (API) за допомогою сучасного інструменту Playwright. Акцент зроблено на перевагах цього підходу, зокрема на можливості перехоплення та модифікації HTTP-запитів, реалізації mocking-сценаріїв, а також тісній інтеграції UI та API тестів в одному середовищі. Наведено приклади реалізації E2E-сценаріїв із одночасною перевіркою відповіді API. Зроблено висновки про доцільність та ефективність застосування Playwright у забезпеченні якості вебдодатків, особливо у проєктах зі складною архітектурою або високими вимогами до стабільності та швидкості тестування.

Ключові слова: *автоматизоване тестування, UI, API, Playwright, E2E, mocking-тестування.*

Вступ

У сучасних розробках програмного забезпечення (ПЗ) важливим етапом є забезпечення високої якості програмних продуктів (ПП), що включає в себе комплексний підхід до тестування. В процесі життєвого циклу розробки ПЗ автоматичне тестування виступає не лише як інструмент контролю якості, але й як фундаментальний елемент, що забезпечує стабільність, надійність та ефективність ПП [1]. Інтеграційне та E2E тестування фокусується на взаємодії між системними модулями і поведінці продукту з точки зору користувача. Тому, важливо використовувати інструменти, які ефективно поєднують UI- та API-тестування, що дозволяє досягти максимального покриття та стабільності в процесі тестування вебзастосунків.

Ціль роботи

Метою даної роботи є дослідження ефективності інтегрованого підходу до автоматизованого тестування вебзастосунків шляхом одночасної перевірки взаємодії інтерфейсу користувача (UI) та серверної логіки (API) за допомогою інструменту Playwright. Особливу увагу приділено аналізу можливостей перехоплення, модифікації та верифікації HTTP-запитів безпосередньо під час UI-взаємодії, а також потенціалу такого підходу в тестуванні.

Матеріали та методи

Забезпечення якості програмного забезпечення (ПЗ) є невід'ємною складовою процесу розроблення і використовуються в ІТ-індустрії фахівцями з тестування і фахівцями щодо забезпечення якості [2]. Окреме тестування UI та API може спричинити проблеми з синхронізацією, коли UI працює коректно, але сервер не відповідає очікувано. Хоча зазвичай UI- та API-тести реалізуються окремо, у складніших сценаріях виникає потреба в

інтегрованому підході до тестування. Для цього часто використовуються додаткові бібліотеки (Axios, Supertest, Postman/Newman тощо), але інструменти нового покоління, як Playwright, дозволяють проводити тестування UI та API в одному середовищі, спрощуючи написання та підтримку E2E-сценаріїв. Актуальність дослідження полягає в потребі гнучкого та ефективного підходу до тестування, що дає змогу перевіряти як поведінку інтерфейсу, так і правильність обробки серверних запитів.

Об'єктом дослідження є інтегрований підхід до перевірки взаємодії UI та API вебзастосунків в автоматизованому тестуванні за допомогою інструменту Playwright. Для досягнення мети використано методи аналізу та методи вивчення впливу цього підходу на процес тестування.

Результати та обговорення

Вибір інструменту для автоматизації тестування впливає на ефективність та якість продукту, враховуючи функціональність і здатність до інтеграції, зокрема для UI та API тестування. Selenium та Playwright є двома найбільш популярними інструментами для кінцевого тестування вебзастосунків, однак вони мають певні відмінності в підходах.

Selenium, зокрема, використовує WebDriver для взаємодії з браузерами через HTTP-протокол, що забезпечує хорошу підтримку різних браузерів, але потребує додаткових бібліотек для інтеграції API-тестів з UI, що ускладнює налаштування та займає більше часу. Натомість, Playwright використовує сучасну архітектуру, безпосередньо взаємодіючи з браузерами через їхні нативні API, такі як Chrome DevTools Protocol (CDP), що забезпечує значно вищу швидкість тестування та здійснювати глибший контроль над браузером. Наприклад, методи перехоплення мережевих запитів, маніпулювання cookie, локальним сховищем та іншими аспектами браузерного середовища, що підтримуються Playwright, дозволяють виконувати API-тести в рамках UI-тестування.

Використання Playwright для інтеграції тестів API та UI надає кілька важливих переваг і можливостей, що значно покращують процес тестування вебзастосунків:

- Зручність в інтеграції тестування UI та API. Завдяки вбудованим можливостям перехоплення запитів і відповідей API, Playwright дає змогу здійснювати тестування API в контексті UI без потреби в окремих інструментах. Це дозволяє автоматично поєднувати API-тести з UI-тестами, забезпечуючи таким чином покращену інтеграцію, коли тестування даних серверу безпосередньо впливає на те, як ці дані відображаються на клієнті.
- Миттєва візуалізація результатів тестів. Playwright дозволяє протестувати API та UI в одному тестовому циклі. Це дає можливість побачити, як API взаємодіє з UI в реальному часі, допомагаючи зменшити час між виявленням проблеми і її виправленням. Тестування API та UI в одному сценарії дозволяє не тільки перевіряти правильність відображення даних на UI, але й контролювати і маніпулювати даними, що повертаються з серверу, перевіряючи їхню правильність у контексті запитів, які були ініційовані через UI.
- Поліпшення стабільності тестування. Завдяки можливості перехоплення і маніпуляції мережевими запитами (наприклад, за допомогою page.on('request') та page.on('response')), Playwright дозволяє виявляти помилки на етапі запитів ще до того, як дані потраплять до UI. Це дозволяє вчасно виявляти помилки і робити коригування на рівні серверу або в кінцевому застосунку.

- Інтеграція тестів з реальними даними. Playwright дає можливість автоматично генерувати дані, перевіряти їх відображення на UI та одночасно перевіряти правильність обробки на сервері. Це дозволяє автоматично обробляти дані, усуваючи потребу в ручній підготовці тестових даних або очищення бази після кожного тесту, що є часозатратним процесом.
- Імітація API запитів для оптимізації тестування. У ситуаціях, де взаємодія з зовнішніми API або сервісами є часозатратною, складною або недоступною, застосування mocking API запитів за допомогою page.route() може значно знизити час виконання тестів. Це дозволяє симулювати реальні сценарії, наприклад, з генеруванням помилок сервера або поверненням некоректних даних, не з'єднуючись із реальними сервісами. Mocking API не повинно замінювати всі типи тестів, особливо у випадках, коли необхідно перевіряти реальні інтеграції між сервісами.

Згідно з описаними перевагами інтеграції тестування UI та API за допомогою Playwright, наведений нижче фрагмент коду демонструє приклад такого підходу (рис.1), як через UI створити задачу в task-менеджері з одночасною перевіркою API-відповіді та очищенням даних для забезпечення ізольованості.

```
test('Verify task creation via UI and validate API response', async ({ page, context }) => {

  await loginAndNavigateToTasks(page, context);
  const tasksPage = new TasksPage(page);
  const [response] = await Promise.all([
    page.waitForResponse(res =>
      res.url().includes('/api/tasks') && res.request().method() === 'POST'
    ),
    tasksPage.createTask({
      title: 'Test Task',
      description: 'Task Description',
      deadline: '2025-05-01T10:00',
      status: 'todo',
      priority: 'medium'
    })
  ]);
  const responseBody = await response.json();
  expect(response.status()).toBe(201);
  expect(responseBody.title).toBe('Test Task');
  page.on('response', async (apiResponse) => {
    if (apiResponse.url().includes('/api/tasks?page=1&limit=10') && apiResponse.status() === 200) {
      const apiResponseBody = await apiResponse.json();
      const task = apiResponseBody.tasks[apiResponseBody.tasks.length - 1];
      expect(task.title).toBe('Test Task');
      expect(task.description).toBe('Task Description');
      expect(task.status).toBe('todo');
      expect(task.priority).toBe('medium');
    }
  });
  await page.reload();
  await expect(page.getByText('Test Task', {exact: true})).toBeVisible();
  const deleteResponse = await page.request.delete(`http://localhost:3000/api/tasks/${responseBody.id}`);
  expect(deleteResponse.status()).toBe(200);
```

Рис. 1 – Фрагмент коду створення задачі в task-менеджері з одночасною перевіркою API-відповіді

Наступний приклад показує, як можна симулювати перевірку пагінації, де кнопка для переходу з'являється при наявності більше 10 записів у таблиці, замінюючи серверні дані на mocked (рис. 2).

```
test('Verify table pagination', async ({ page }) => {
  await page.route('**/api/tasks**', route => {
    const url = new URL(route.request().url());
    const pageParam = url.searchParams.get('page') || '1';

    const allTasks = Array.from({ length: 20 }).map((_, i) => ({
      id: i + 1,
      user_id: 2,
      title: `Task ${i + 1}`,
      description: `Description for task ${i + 1}`,
      deadline: '2025-05-01T17:42:00.000Z',
      status: i % 2 === 0 ? 'completed' : 'in-progress',
      priority: 'medium',
      created_at: '2025-04-22T17:42:54.000Z'
    }));

    const limit = 10;
    const pageNumber = parseInt(pageParam);
    const offset = (pageNumber - 1) * limit;
    const pagedTasks = allTasks.slice(offset, offset + limit);

    route.fulfill({
      status: 200,
      contentType: 'application/json',
      body: JSON.stringify({
        message: 'Tasks retrieved successfully',
        tasks: pagedTasks,
        total: allTasks.length,
        page: pageNumber,
        limit,
        totalPages: Math.ceil(allTasks.length / limit)
      })
    });
  });

  await loginAndNavigateToTasks(page, page.context());
  await expect(page.getByText('Task 1', { exact: true })).toBeVisible();
  await expect(page.getByText('Task 10')).toBeVisible();
  await expect(page.getByText('Task 11')).not.toBeVisible();
  await page.getByRole('button', { name: 'Next' }).click();
  await expect(page.getByText('Task 11')).toBeVisible();
  await expect(page.getByText('Task 20')).toBeVisible();
});
```

Рис. 2 – Фрагмент коду mock-тестування за допомогою Playwright

Показані приклади коду для створення задач і тестування пагінації з одночасною валідацією API підтверджують прикладну користь підходу.

Аналогічний підхід може бути критично важливим у складних системах, де наявність third-party інтеграцій, часові обмеження, нестабільні чи динамічні дані можуть ускладнювати перевірку стандартними методами.

Таким чином, отримані результати демонструють, що використанням інструменту Playwright забезпечує ефективне, стабільне та гнучке середовище для одночасного тестування UI та API у сучасних вебзастосунках.

Висновки

Інтегрований підхід до автоматизованого тестування UI та API за допомогою Playwright, завдяки доступу до нативних API браузера, перехопленню та mocking запитів, забезпечує високу ефективність і стабільність тестування вебзастосунків, дозволяючи швидко виявляти помилки, зменшувати зовнішні залежності та моделювати поведінку системи в складних умовах сучасних вебдодатків. Використання даного підходу покращує синхронізацію між різними компонентами системи, що дозволяє реалістичніше тестувати поведінку системи.

Список використаних джерел

1. Шибицька Н.М., Кочеткова О.В. Інтеграційне тестування та контроль якості програмного забезпечення// Proceedings 1st international scientific and practical conference «Information Systems and Technology: Results and Prospects» (IST 2024)", March 6, 2024 - K.: FIT TSNUK, 2024.- С.366-369

2. Ушакова І. О. Підходи до забезпечення якості програмного забезпечення. Сучасні інформаційні технології і системи : монографія. Харків : «Стильіздат», 2021. С. 125–140.

Відомості про авторів:

Шибицька Наталія Миколаївна – доцент кафедри ІПЗ факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут», (Київ, Україна). Наукові інтереси: інженерія програмного забезпечення, інтелектуальні системи навчання та експертні методи оцінювання, методи та моделі штучного інтелекту, нечітка математика та нейронні мережі.

E-mail: shibnatnik@ukr.net

Кудря Анастасія Ігорівна – здобувачка вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». Наукові інтереси: інженерія та тестування программного забезпечення.

E-mail: kudryastasy@gmail.com

УДК 004.415(045)

ЯКІСТЬ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Дмитро МАЗУРЕНКО
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Олена Олегівна Колганова

У роботі досліджується роль автоматизованого тестування та штучного інтелекту (ШІ) в забезпеченні якості програмного забезпечення. Розглядаються переваги і виклики застосування цих підходів у швидко змінюваних умовах розробки ПЗ, зокрема в контексті Agile та DevOps. Порівнюються традиційні методи тестування з автоматизованим та інтелектуальним тестуванням, наводяться критерії ефективності таких підходів, включаючи час виконання тестів, точність виявлення дефектів та здатність до адаптації.

***Ключові слова**: автоматизоване тестування, штучний інтелект, якість ПЗ, тестування, машинне навчання, адаптивність.*

Вступ

Сучасне програмне забезпечення характеризується стрімким зростанням складності, динамічними змінами вимог замовників та високими очікуваннями кінцевих користувачів щодо якості та стабільності функціонування. У таких умовах традиційні підходи до тестування часто не забезпечують належної швидкості, гнучкості та ефективності. Ручне тестування є обмеженим при високому рівні масштабованості та повторюваності, особливо в контексті Agile- та DevOps-орієнтованої розробки. Це зумовлює необхідність пошуку нових підходів до забезпечення якості, серед яких автоматизоване тестування та застосування штучного інтелекту (ШІ) набувають ключового значення.

Ціль роботи

Метою цього теоретичного дослідження є всебічне обґрунтування доцільності впровадження автоматизованого тестування та технологій штучного інтелекту в процеси перевірки якості програмного забезпечення. Особлива увага приділяється аналізу переваг, викликів і перспектив використання цих підходів у реальному виробничому середовищі.

Матеріали та методи

У роботі застосовано методи системного аналізу, огляду наукової та технічної літератури, а також порівняльного аналізу результатів досліджень, що висвітлюють практику впровадження автоматизації та ШІ в тестування ПЗ.

Основним джерелом інформації стали наукові статті та технічні звіти провідних IT-компаній, що містять порівняльну статистику ефективності інструментів та підходів. Окрім того, враховано результати практичних досліджень із відкритими даними, які дозволяють оцінити точність та продуктивність ШІ-моделей у задачах передбачення помилок.

Застосовано порівняльний аналіз трьох основних підходів до тестування: ручного, автоматизованого та інтелектуального. Порівняння здійснювалося за такими критеріями:

- витрати часу на створення та виконання тестів;

- рівень покриття коду тестами;
- точність виявлення дефектів;
- здатність до адаптації до змін у програмному середовищі;
- необхідні ресурси для підтримки тестової інфраструктури.

Таким чином, комплексне використання теоретичного аналізу, огляду літератури, вивчення практик індустрії та статистичних звітів забезпечило широке й глибоке підґрунтя для формування висновків щодо доцільності впровадження автоматизованого та ШІ тестування.

Результати та обговорення

Автоматизоване тестування давно зарекомендувало себе як інструмент, що дозволяє зменшити час на виконання тестів та забезпечити стабільне виконання перевірок при кожній зміні коду. Його основною перевагою є здатність до багаторазового відтворення тестів із мінімальними витратами людських ресурсів. Завдяки цьому автоматизація особливо ефективна у випадках великої кількості регресійних тестів, тестів продуктивності, UI- та інтеграційного тестування.

Сучасні інструменти, такі як Selenium, Appium, TestNG та інші, надають широкі можливості для автоматизації, проте потребують значних зусиль на етапі налаштування та підтримки тестів. У свою чергу, використання підходів на основі ШІ дозволяє значно підвищити гнучкість і адаптивність автоматизованих рішень. Зокрема, завдяки застосуванню методів машинного навчання (ML) та обробки природної мови (NLP), ШІ може брати участь не лише у виконанні тестів, але й у їхньому створенні, оптимізації та пріоритизації.

ШІ здатен автоматично генерувати тест-кейси на основі історії комітів, змін у функціональності та попередніх дефектів, що значно скорочує час на підготовку тестової документації. Крім того, ML моделі можуть виявляти області коду з високою ймовірністю помилок, дозволяючи зосередити ресурси тестування саме там, де вони найбільш необхідні. Приклади таких систем включають прогнозуючі модулі в GitHub Copilot, DeepCode та інші.

Окремий напрямок становить "розумне тестування" (smart testing), яке комбінує автоматизацію із самонавчальними алгоритмами, що адаптуються до змін середовища виконання та поведінки користувачів. Завдяки цьому можлива побудова систем тестування, які еволюціонують разом із ПЗ і здатні самостійно виявляти критичні зони ризику.

Разом з тим, використання ШІ у тестуванні має певні обмеження. По-перше, навчання моделей вимагає великої кількості якісних даних, що не завжди доступні. По-друге, результати роботи таких систем часто складно інтерпретувати через обмежену прозорість рішень. По-третє, розробка та підтримка таких рішень потребує високої кваліфікації спеціалістів, що може обмежувати впровадження в малих командах.

Незважаючи на ці виклики, потенціал інтеграції ШІ в тестування є надзвичайно високим. Вже сьогодні спостерігається зростання кількості стартапів і R&D-проєктів, спрямованих на створення автономних тестових агентів, систем самовідновлення тестів при змінах у DOM-структурі, а також аналітичних платформ для виявлення аномалій у поведінці користувачів.

Нижче представлено таблицю, що порівнює три основні підходи до тестування: ручне, автоматизоване та інтелектуальне (на базі ШІ), за такими критеріями: витрати часу, рівень покриття коду, точність виявлення дефектів, адаптивність до змін та потреба у підтримці інфраструктури.

Таблиця 1

Порівняння підходів до тестування

Критерій	Ручне тестування	Автоматизоване тестування	Інтелектуальне (ШІ) тестування
Час на створення й виконання тестів	Високий — вимагає значного часу фахівців на підготовку та прогін сценаріїв. Наприклад, типове ручне регресійне тестування може займати десятки годин.	Середній — автоматизовані тести після початкового налаштування запускаються багаторазово з мінімальними затратами часу.	Низький — завдяки можливостям ШІ тести оновлюються й оптимізуються автоматично, що дозволяє зменшити загальний час виконання до мінімуму.
Рівень покриття коду тестами	Залежить від досвіду тестувальника; зазвичай не перевищує 40–60 %.	Досягає 70% завдяки систематичному підходу в автоматичних скриптах.	Може досягати 90% за рахунок генерації тестів на основі логів, змін у коді та історичних даних.
Точність виявлення дефектів	Середня — суб'єктивність і втома тестувальника впливають на якість виявлення помилок.	Висока — скрипти знаходять більше помилок у порівнянні з ручною перевіркою.	Дуже висока — моделі прогнозування дефектів дозволяють виявляти аномалії до їх прояву в системі.
Адаптація до змін у ПЗ	Висока, але повністю залежна від людини — необхідно вручну змінювати сценарії.	Середня — при змінах у UI автоматичні тести часто "ламаються".	Висока — самооновлення тестів знижує кількість flaky-тестів і забезпечує гнучкість при змінах.
Ресурси підтримки інфраструктури	Високі людські ресурси, низькі технічні.	Високі технічні ресурси (сервери, CI/CD, фреймворки, підтримка інструментів).	Дуже високі — необхідні ресурси для обробки ШІ-моделей, хмарних рішень, та підтримки складної екосистеми.

Згідно зі звітом Capgemini World Quality Report (2023)[1], автоматизоване тестування дозволяє зменшити час регресійного тестування до 70% у порівнянні з ручним. Інструменти на базі ШІ, як-от Mabl, додатково скорочують час виконання тестів ще на 10–15%, завдяки динамічному оновленню тест-кейсів на основі змін у коді. У сукупності це забезпечує скорочення до 80% у порівнянні з традиційними ручними підходами.

Дослідження, проведені з використанням бенчмарку Defects4J, а також експерименти з порівнянням традиційних і ШІ-заснованих інструментів автоматизованого тестування, демонструють істотну різницю в рівні покриття коду залежно від підходу. Ручне тестування забезпечує в середньому 40–60% покриття, головним чином через людські обмеження в масштабі перевірок. Автоматизовані фреймворки, дозволяють досягати до 70% покриття завдяки повторюваності й систематичному підходу [2]. При використанні ШІ-інструментів, які генерують тест-кейси з історії змін та логів, цей показник може досягати 90%.

Інструменти, які використовують машинне навчання для виявлення змін у поведінці системи, демонструють до 20% більше виявлених критичних дефектів у порівнянні зі звичайною автоматизацією[3]. Згідно з IEEE Software [4], моделі JIT (just-in-time) дефект-прогнозування дозволяють передбачити (recall) близько 65% дефектів з точністю (precision) до 70%, що значно підвищує ефективність процесу контролю якості.

Автоматизовані скрипти часто вимагають ручного втручання при зміні UI або логіки програми. У свою чергу, інструменти з елементами ШІ адаптуються до змін без втрати стабільності тестів, зменшуючи кількість нестабільних «flaky» тестів на 30–50%. Наприклад,

Testim та Functionize застосовують механізми самооновлення тестів на основі аналізу DOM і поведінки користувачів [5].

Використання автоматизованого тестування зменшує кількість помилок, що проходять до продакшну, на 40–60%, а інструменти з підтримкою ШІ допомагають скоротити час виявлення дефекту (defect detection time) у середньому на 25%[6]. Найбільший вплив мають показники точності виявлення дефектів, покриття коду та стабільність тестів, які прямо корелюють із зниженням витрат і підвищенням ROI.

Висновки

Проведене теоретичне дослідження підтверджує, що автоматизоване тестування у поєднанні з інтелектуальними підходами на базі ШІ є ефективним шляхом до підвищення якості програмного забезпечення. Автоматизація забезпечує швидкість і повторюваність, а ШІ – інтелектуальну підтримку процесу прийняття рішень у тестуванні. Важливо, що сучасні інструменти надають змогу не лише автоматизувати виконання тестів, а й підвищити точність виявлення дефектів, виявити ризики та зробити тестування більш проактивним. Інвестиції в ці технології є виправданими і стратегічно важливими для провідних компаній. Тому в перспективі можна очікувати появу систем, які з мінімальним втручанням людини будуть забезпечувати повний життєвий цикл тестування.

Список використаних джерел:

1. Capgemini, Sogeti, Micro Focus. World Quality Report 2023–24. [Електронний ресурс] — Режим доступу: https://www.capgemini.com/insights/research-library/world-quality-report-2023-24/ (дата звернення: 05.05.2025)

2. Gkikopouli, M., & Bataa, B. Empirical Comparison Between Conventional and AI-based Automated Unit Test Generation Tools in Java. In: Bachelor Degree Project, Blekinge Institute of Technology, 2023. [Електронний ресурс]. — Режим доступу: https://www.diva-portal.org/smash/get/diva2:1764443/FULLTEXT01.pdf (дата звернення: 05.05.2025)

3. Mabl: AI-Powered Test Automation. [Електронний ресурс] — Режим доступу: https://www.mabl.com/ (дата звернення: 05.05.2025)

4. KAMEI, Y., SHIHAB, E., ADAMS, B., et al. A Large-Scale Empirical Study of Just-In-Time Quality Assurance [Electronic resource] // IEEE Transactions on Software Engineering. – 2013. – Vol. 39, No. 6. – P. 757–773. – DOI: https://doi.org/10.1109/TSE.2012.70.

5. Functionize. Self-Healing Test Automation. [Електронний ресурс] — Режим доступу: https://www.functionize.com/self-healing (дата звернення: 05.05.2025)

6. DevOps.com. The Cost of Software Quality. [Електронний ресурс] — Режим доступу: https://devops.com/the-cost-of-software-quality/ (дата звернення: 05.05.2025)

Відомості про автора:

Мазуренко Дмитро Іванович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, автоматизоване тестування, штучний інтелект, якість ПЗ.

E-mail: dmazurenko55@gmail.com

УДК 004. 413 (045)

ОПТИМІЗАЦІЯ ПРОЦЕСУ CODE REVIEW ЗА ДОПОМГОЮ ШТУЧНОГО ІНТЕЛЕКТУ: ПЕРСПЕКТИВИ ТА ОБМЕЖЕННЯ

Ярослав МАНЬКІВСЬКИЙ
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Лариса Валеріївна Дакова

У статті розглядаються методи автоматизації огляду коду за допомогою технологій штучного інтелекту.

Ключові слова: *штучний інтелект, Code Review, автоматизація процесів людино-машинна взаємодія.*

Вступ

У сучасному світі індустрія програмного забезпечення стикається з багатовимірними викликами: зростанням обсягів кодових баз, прискоренням циклів випуску та розширенням команд розробників у глобальному масштабі. Традиційний ручний Code Review, незважаючи на свою важливість, стає вузьким місцем у процесі CI/CD, адже поглиблений аналіз коду потребує значних часових витрат і людських ресурсів. За даними міжнародного опитування, до 40% часу рев'ю може йти на виявлення дрібних синтаксичних та стилістичних помилок, що створює затримки у поставці нових функцій і знижує мотивацію розробників [1].

Поява AI-асистентів у перевірці коду відкриває нові горизонти автоматизації: машинне навчання та глибокі нейронні мережі дозволяють здійснювати статичний і семантичний аналіз програмних артефактів у масштабі великих репозиторіїв. Інструменти на кшталт GitHub Copilot, Amazon CodeWhisperer і DeepCode вже сьогодні демонструють здатність виявляти складні патерни помилок, пропонувати альтернативні реалізації та покращувати архітектурну цілісність проєкту в режимі реального часу [2, 6].

Крім суто технічних переваг, застосування ШІ у Code Review має значний соціально-освітній вплив: здобувачі вищої освіти та молоді спеціалісти отримують миттєвий зворотний зв'язок, що сприяє ефективнішому засвоєнню кращих практик програмування та зменшує криву навчання. Водночас автоматизація породжує низку нових питань: як гарантувати прозорість алгоритмічних рішень, які етичні межі використання тренувальних даних, та як збалансувати участь ШІ й людини для мінімізації ризиків надмірної залежності від машинних рекомендацій.

Таким чином, дана тема окреслює не лише технічні передумови та мотивацію для інтеграції AI-інструментів у процес перевірки коду, але й підкреслює необхідність системного підходу, який поєднує сильні сторони автоматизації та експертної людської оцінки для досягнення оптимального балансу між швидкістю, якістю та етикою розробки.

Ціль роботи

Мета роботи – аналіз можливостей, переваг та обмежень використання штучного інтелекту в процесі перевірки коду. Робота передбачає оцінку ефективності ШІ-асистентів, а

також виявлення потенційних ризиків і етичних питань, пов’язаних із їх впровадженням у реальні проєкти.

Матеріали та методи

Перегляди коду часто сприяють знаходженню та виправленню загальних вразливостей, таких як вразливості форматних рядків, помилки некоректної послідовності виконання частин коду, витоки пам'яті та переповнення буферу, таким чином покращуючи безпеку програмного забезпечення. Онлайн-репозиторії на основі Subversion, Mercurial, Git або інших, дозволяють групам користувачів спільно робити перегляд коду. Крім того, спеціальні інструменти для спільного перегляду коду допомагають полегшити цей процес.

Автоматизоване ПЗ для перегляду коду дозволяє зменшити завдання по перегляду великих відрізків коду завдяки автоматичній перевірці вихідного коду на відомі вразливості.

При перегляді коду рекомендують перевіряти 200 – 400 рядків за годину. Інспектування та перегляд більш ніж декілька сотень рядків коду за годину для критичного ПЗ (наприклад, критичного в плані безпеки вбудованого ПЗ) може бути занадто швидким для того, щоб знайти помилки. Дані по галузі свідчать, що, при перегляді коду, можна досягти виявлення до 85 % помилок, при середньому значенні цього показника 65 %.

Як показали емпіричні дослідження, до 75 % дефектів, виявлених при перегляді коду, більш стосуються розширюваності програмного забезпечення, аніж його функціональності, що робить перегляд коду відмінним інструментом для компаній, які працюють над продуктами або системами з довгим циклом розробки.

Перегляд коду також дає час аби запропонувати рефакторинг [5].

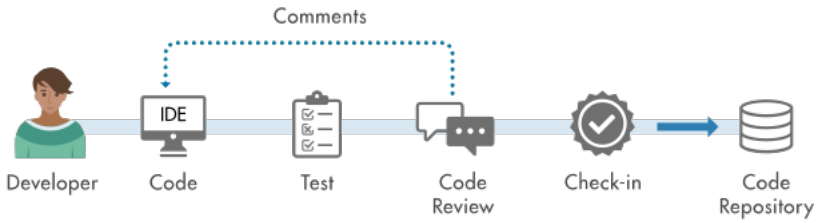


Рис. 1 «Типовий процес розробки ПЗ»[8]

Щоб оцінити ефективність Code Review використовуються наступні метрики:

- Час Code Review
- Швидкість обробки рядків коду
- Коефіцієнт дефектності
- Рівень false positive/false negative

Результати та обговорення

Аналіз показує, що впровадження ШІ-систем у процес рев’ю коду значно підвищує його ефективність. По-перше, ШІ здатен миттєво обробляти великі обсяги коду – на відміну від ручного аналізу, автоматичні алгоритми сканують код за секунди і виділяють потенційні проблеми [4], [6]. Наприклад, дослідження GitHub засвідчило, що з Copilot Chat рев’ю коду відбуваються на 15% швидше, а розробники рідше втрачають фокус під час перевірки коду. По-друге, ШІ-інструменти гарантують стабільну якість перевірки: на відміну від людей, котрі втомлюються чи мають упередження, алгоритми працюють послідовно й виявляють більше помилок навіть у великих і різнорідних кодових базах. Наприклад, рутинні помилки (від пропущених дужок до невідповідностей стильових правил) автоматично фіксуються безперебійно, що звільняє програмістів для складніших завдань. Як показують дані, 48%

компаній уже визнають користь ШІ саме у рев'ю коду, відзначаючи підвищення продуктивності та якості розробки. В цілому автоматизація рутинних перевірок дає змогу зменшити часові затрати на рецензування й фокусуватися на архітектурі та складній логіці проєкту.

Крім того, ШІ-системи ефективно виявляють складні для виявлення помилки, що часто пропускаються вручну. Так, подібні алгоритми можуть проаналізувати код у різних контекстах і сценаріях виконання, виявивши рідкісні баги чи умовні вразливості, які важко підсвітити простим оглядом. Синергія зворотного зв'язку від ШІ та людського аналізу дозволяє скоротити «технічний борг» – так, моделі, навчені на великих репозиторіях, виділяють code smells і небезпеки, що могли залишитися непоміченими . В сумі результати говорять про те, що автоматизоване рев'ю коду зі штучним інтелектом забезпечує вищу стабільність і повноту перевірки при значній економії часу.

Переваги застосування інструментів (GitHub Copilot, Amazon CodeWhisperer, DeepCode)

GitHub Copilot пропонує контекстно-залежні автозаповнення коду та генерацію цілих функцій на основі введеного тексту чи коментарів. Завдяки навчанню на великому корпусі відкритого коду, Copilot підтримує багато мов програмування і може генерувати шаблонний код і навіть складні алгоритми. Наше дослідження показаааало, що з Copilot розробники відчувають більше впевненості в якості свого коду: 85% учасників експерименту повідомили про підвищення впевненості, а 88% змогли зберігати продуктивний «flow» під час кодування[9]. Amazon CodeWhisperer інтегрується з AWS і пропонує в реальному часі підказки з урахуванням контексту AWS-сервісів, включаючи рекомендації щодо безпеки коду. Завдяки цьому розробники, що працюють в екосистемі AWS, отримують спеціалізований інструмент, який враховує вимоги хмарної платформи.

DeepCode (Snyk Code) – це інструмент статичного аналізу коду, орієнтований на безпеку і якість. На відміну від генеративних помічників, він виконує глибокий аналіз існуючого коду, шукаючи вразливості та кодові «смердючі плями». Система DeepCode сканує код в реальному часі, підсвічуючи проблемні місця та радячи виправлення. У результаті команда отримує миттєвий зворотний зв'язок з безпеки й стилю коду, що підвищує загальну якість проєкту.

Узагальнено, ці інструменти підвищують якість коду та продуктивність розробки. ШІ-рецензенти виявляють помилки (від синтаксичних до складних уразливостей), зменшують кількість людських пропусків і збільшують здатність команди дотримуватися стандартів. Наприклад, аналітична стаття Snyk [11] наголошує, що автоматизація рев'ю дозволяє виявляти більше уразливостей і «невидимих» багів, що знижує технічний борг і дає змогу розробникам зосередитися на творчих задачах. Крім того, автоматична генерація пропозицій зберігає єдність стилю коду в команді: відповідно до звіту Intetics [12], Copilot підтримує консистентні практики кодування й виступає каталізатором обговорень.

Обмеження і потенційні ризики використання ШІ

ШІ не розуміє глибокого контексту бізнес-логіки – це може призводити до хибних або невідповідних підказок. Існує також небезпека надмірного покладання на ШІ – розробники можуть втрачати навички критичного аналізу. ШІ-інструменти схильні до помилок типу false-positive та false-negative – або сприймають коректний фрагмент як помилковий, або пропускають реальні проблеми. Наприклад, Copilot генерує коректний код лише у ~ 46% випадків, а CodeWhisperer – у 31%. Також є ризики конфіденційності: інструменти

потребують доступу до приватного коду, що несе потенційну загрозу витоку даних. Тому доцільно зберігати участь людини в рев'ю-процесі

Висновки

Інтеграція штучного інтелекту в процес Code Review забезпечує значне підвищення його ефективності за рахунок автоматизації рутинних перевірок, прискореного виявлення помилок та збереження єдності стилю коду. AI-інструменти, такі як GitHub Copilot, Amazon CodeWhisperer та DeepCode, доповнюють експертний аналіз, знижуючи навантаження на розробників і дозволяючи їм зосередитися на архітектурних та творчих завданнях. Разом із тим, існують ризики хибних спрацьовувань, втрати критичного мислення та потенційних загроз конфіденційності, що вимагає збереження людської верифікації кожної рекомендації. Найефективнішим є гібридний підхід: AI проводить первинне сканування та підсвічує проблемні місця, а фінальну оцінку й прийняття рішення виконує людина. Це забезпечує оптимальний баланс між швидкістю, якістю та безпекою розробки.

Список використаних джерел

1. Verdi, S. «Ефективність та обмеження AI у рев'ю коду». Graphite, 2023. https://graphite.dev/guides/effectiveness-and-limitations-of-ai-code-review

2. GitHub Docs. «Використання GitHub Copilot для рев'ю коду». https://docs.github.com/en/copilot/using-github-copilot/code-review/using-copilot-code-review

3. Mission Cloud. «Amazon CodeWhisperer проти Copilot: що обрати?». 12.01.2024. https://www.missioncloud.com/blog/github-copilot-vs-amazon-codewhisperer

4. Dhaliwal, P. «6 обмежень AI-асистентів коду та чому розробникам варто бути обережними». All Things Open, 2023. https://allthingsopen.org/articles/ai-code-assistants-limitations

5. Wikipedia, «Перегляд коду». https://shorturl.at/0YZlr

6. Graphite. «Огляд використання DeepCode AI для рев'ю коду». https://graphite.dev/guides/using-deepcode-ai-for-code-review

7. Vanek, B. «Microsoft Copilot: відповідність та етичні аспекти використання AI-інструменту». University of Utah, 03.09.2024. https://attheu.utah.edu/facultystaff/microsoft-copilot-compliance-and-ethical-considerations-for-the-ai-tool/

8. MathWorks, «Code Review». https://de.mathworks.com/discovery/code-review.html

9. Tutorialspoint. «GitHub Copilot — рев'ю коду». https://www.tutorialspoint.com/github-copilot/github_copilot_code_review.htm

10. OpenAI. «Про обмеження Codex і Copilot». https://openai.com/blog/openai-codex

11. Snyk, «4 Переваги використання ШІ для перевірки коду». https://snyk.io/blog/4-advantages-of-ai-code-review/

12. Intetics, «Чому GitHub Copilot — це геймченджер для розробників програмного забезпечення». https://intetics.com/blog/why-github-copilot-is-the-ultimate-game-changer-for-software-developers/

Відомості про автора:

Маньківський Ярослав Андрійович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення

E-mail: y.mankivskyi@gmail.com

УДК 004. 4

ЗАСТОСУВАННЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ АНАЛІЗУ ВІДПОВІДНОСТІ ПРОГРАМНОГО КОДУ ПРИНЦИПАМ SOLID

Тарас МЕЛЬНИК
Здобувач вищої освіти 1-го курсу магістратури кафедри ІПЗ
Євген ТАТАРИНОВ
к.ф.-м.н., доцент кафедри ІПЗ ФКНТ
Державний університет «Київський авіаційний інститут»

У статті досліджується застосування великих мовних моделей для автоматизованої перевірки дотримання принципів SOLID у програмному коді на прикладі C#. Проаналізовано результати роботи різних моделей (GPT-4o, GPT-4.5, GPT-4.1, o4-mini-high) та окреслено шляхи підвищення їхньої ефективності в задачах контролю якості коду.

Ключові слова: *SOLID, принцип, якість, велика мовна модель, LLM, GPT, перевірка коду.*

Вступ

У сучасній розробці програмного забезпечення все більше уваги приділяється якості програмних систем та дотриманню архітектурних принципів. Автоматизація процесу перевірки коду за допомогою великих мовних моделей (LLM) виглядає перспективним шляхом у покращенні чистоти коду та вдосконаленні процесу розробки [1].

На сьогоднішній день існує ряд принципів розробки програмного забезпечення, що мають на меті задати правильний вектор у побудові масштабованої, підтримуваної та якісної системи, серед яких найефективнішими та найпопулярнішими є принципи SOLID [2].

Метою даної роботи є представлення результатів дослідження того, наскільки якісно сучасні великі мовні моделі здатні виявляти порушення або дотримання принципів SOLID у вихідному коді програми.

Матеріали і методи

Для проведення дослідження було підібрано 20 екземплярів коду, написаного мовою програмування C#, що дотримують або порушують принципи SOLID. Серед них є як елементарні прості фрагменти коду, так і архіви повноцінних еталонних проєктів .NET. Для кожного прикладу вручну визначено його відповідність принципам SOLID.

Для проведення дослідження залучені такі моделі, як GPT-4o, GPT-4.5, GPT-4.1 та o4-mini-high від OpenAI [3]. Для виконання запитів сформовано два промпти: для фрагменту коду та для архіву із повним програмним проєктом. Усі запити до моделей виконувались в анонімному, тимчасовому чатах задля того, щоб не зберігався контекст відповідей. У разі неоднозначної відповіді («Частково») результат визначався якістю обґрунтування. Визначення результатів відбувалось шляхом порівняння відповідей моделей із попередньо визначеними еталонними значеннями. Всього є 4 типи результату:

1. True Positive (TP) – LLM правильно вказала, що принцип дотримано;
2. True Negative (TN) – LLM правильно вказала, що принцип порушено;
3. False Positive (FP) – LLM вказала «дотримано», хоча принцип порушено;
4. False Negative (FN) – LLM вказала «порушено», хоча принцип дотримано.

З цих значень обчислюються наступні метрики:

- Accuracy (загальна точність) – частка правильних відповідей моделі серед усіх випадків;
- Precision (точність позитивного прогнозу) – з-поміж усіх випадків, коли LLM сказала «дотримано», скільки з них були правильними;
- Recall (повнота, чутливість) – з-поміж усіх випадків, коли принцип насправді дотримано, скільки LLM правильно виявила;
- F1-Score (баланс між точністю і повнотою) – середнє гармонійне Precision і Recall, дає зважену оцінку, яка балансує між FP і FN.

Результати

У таблиці 1 надано отримані значення метрик для кожної піддослідної моделі.

Таблиця 1.

Кількісні результати дослідження

Модель	Accuracy	Precision	Recall	F1-score
GPT-4o	87%	97%	85%	91%
GPT-4.5	83%	98%	82%	89%
GPT-4.1	91%	93%	95%	94%
o4-mini-high	67%	59%	94%	72%

Модель GPT-4o досить швидко надавала відповіді, але вона розглядала дуже маленьку частину вхідних даних. GPT-4.5 здійснює набагато довший аналіз та надає набагато ґрунтовніший огляд у порівнянні з GPT-4o. Дана модель опрацьовує значно більший обсяг інформації та бере до уваги значно більший контекст. GPT-4.1 дуже схожа до GPT-4o – так само швидко й так само поверхнево надає відповіді. З точки зору аргументації та аналізу модель o4-mini-high перебуває на рівні із GPT-4.5, якщо не краще, але в той же час робить це набагато швидше. Проте вона здається «перенавченою» – там, де не варто доопрацьовувати код, модель намагається довести його «до ідеалу», що не є доцільним. Через це для даної моделі були отримані такі низькі показники метрик.

Найвищі формальні метрики були досягнуті моделлю GPT-4.1, тоді як GPT-4.5 та o4-mini-high продемонстрували кращу глибину аналізу та здатність до аргументованих висновків. Це дозволяє стверджувати, що вибір моделі залежить від цілей: якщо важлива швидкість і точність, перевагу варто надати GPT-4o або GPT-4.1; якщо ж пріоритетом є якісний глибинний аудит коду, тоді доцільніше використовувати GPT-4.5 або o4-mini-high.

Висновки

На основі отриманих результатів можна запропонувати декілька напрямків, як підвищити ефективність використання LLM для контролю SOLID та якості коду в цілому:

1. Вдосконалення промптів

Можна експериментувати з більш структурованими промптами. Наприклад, замість одного загального питання про SOLID, поставити серію питань: окремо про кожен принцип.

2. Архітектура LLM + RAG

Перспективним напрямком є побудова окремого застосунку або системи, що поєднує LLM з методами генерації з доповненням через пошук (Retrieval-Augmented Generation, RAG) [4]. Суть даного підходу полягає в тому, щоб надати моделі доступ до зовнішнього знання в процесі відповіді.

3. Вузька спеціалізація моделей

Ще один шлях – це створення або донавчання моделей, спеціально спрямованих на аналіз якості коду. Така модель може бути менш універсальною, але краще розпізнаватиме знайомі анти-патерни.

4. Виявлення Code Smells

Також досить практичним є підхід, зорієнтований на пошук конкретних симптомів поганого дизайну (code smells). Багато «запахів» безпосередньо пов'язані із SOLID. Такий спосіб може бути більш надійним, адже він оперує конкретними характеристиками коду (розмір, дублювання, глибина ієрархії, кількість залежностей тощо), які легше виявити автоматично.

Підсумовуючи, дослідження підтвердило, що LLM можуть стати ефективним інструментом для автоматизації контролю архітектурних принципів, але наразі їх слід застосовувати обережно, у поєднанні з традиційними методами. Майбутні роботи в цьому напрямі мають зосередитися на розширенні можливостей моделей.

Список використаних джерел

1. The Use of Large Language Model in Code Review Automation: An Examination of Enforcing SOLID Principles. URL: https://link.springer.com/chapter/10.1007/978-3-031-60615-1_6.

2. SOLID (object-oriented design). URL: https://simple.wikipedia.org/wiki/SOLID_(object-oriented_design)

3. OpenAI ChatGPT. URL: https://chatgpt.com.

4. What is RAG (Retrieval-Augmented Generation)? URL: https://aws.amazon.com/what-is/retrieval-augmented-generation

Відомості про авторів:

Мельник Тарас Васильович – здобувач вищої освіти 1-го курсу магістратури кафедри інженерії програмного забезпечення Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, мовні моделі, якість програмного забезпечення.

E-mail: 5386862@stud.kai.edu.ua

Татаринов Євген Олександрович – доцент кафедри інженерії програмного забезпечення Державного університету «Київський авіаційний інститут». *Наукові інтереси*: розробка додатків та бах данних C# .NET, MS SQL, застосування AI/ML у вирішенні практичних задач бізнесу.

E-mail: yevhen.tatarynov@npp.kai.edu.ua

УДК 004.8(043.2)

КОНЦЕПТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ІНТЕРАКТИВНОГО ПРОЄКТУВАННЯ MSA З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

Артем МОРГУН
Аспірант 1 року навчання кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий консультант к.т.н., професор кафедри ІПЗ ФКНТ
Андрій Іванович Гізун

У роботі розглядається підхід до проєктування мікросервісної архітектури (MSA) із використанням штучного інтелекту. Аналізуються недоліки традиційного методу побудови архітектури за допомогою блок-схем, зокрема відсутність автоматизації та зв'язності між компонентами. Запропоновано концепт програмного забезпечення та архітектуру основних модулів, які забезпечує реконструкцію та генерацію MSA-моделі з урахуванням вимог і підтримкою інтерактивної взаємодії між інженером та ШІ.

Ключові слова: *мікросервісна архітектура, штучний інтелект, моделювання, реконструкція, генерація, система, автоматизація.*

Вступ

Проєктування мікросервісної архітектури (MSA) програмного забезпечення (ПЗ) є складним і комплексним процесом. Щоб реалізувати переваги MSA – зокрема, незалежність розробки, функціонування та розгортання [1], – необхідно приймати виважені рішення щодо створення нових компонентів або модифікації вже існуючих з урахуванням наявних доменних областей (bounded context) та нових чи змінених вимог. Такі рішення мають мінімізувати ризики виникнення антипатернів у розподілених системах.

Найпростішим і, відповідно, найпоширенішим підходом є використання блок-схем та діаграм [2], у яких компоненти MSA-системи представлені у вигляді геометричних фігур із довільним описом їхніх характеристик. Прикладами ПЗ для побудови таких діаграм є Miro, Lucidcharts, Draw.io. Попри простоту такого підходу, він має суттєві недоліки: через відсутність зв'язності між компонентами будь-які зміни вимагають ручного оновлення діаграми та подальшої перевірки на коректність і повноту. Це ускладнює процес розробки, особливо в умовах постійних змін та розширень системи, характерних для життєвого циклу Agile.

Використання штучного інтелекту (ШІ) для проєктування MSA може допомогти інженерам подолати зазначені виклики. Завдяки ШІ є можливість відтворити поточний стан системи з подальшим внесенням змін відповідно до поставлених завдань із збереженням зв'язків між компонентами. Архітектура MSA може бути описана однією з мов моделювання (з використанням або на основі Komponents [3]), і на основі такої моделі інженер може за допомогою ШІ ухвалювати рішення щодо кількості необхідних сервісів, вибору шаблонів проєктування та способів комунікації між сервісами.

Існуюче програмне забезпечення з використанням ШІ, яке може застосовуватись для реконструкції MSA-систем (один із прикладів – MicroART [4]) або генерації компонентів MSA на основі неструктурованих вимог чи розкладання моноліту [5], здебільшого є

прототипами. Їхній результат – це зазвичай графічний або текстовий опис, що не є формальною моделлю. Виникає потреба в створенні цілісного підходу до проєктування MSA, в основі якого лежить модель, що може бути сформована з існуючої системи та модифікована відповідно до нових вимог із збереженням зв'язків і властивостей компонентів. Такий підхід дозволяє проводити подальший аналіз архітектури, застосовувати рекомендації щодо шаблонів проєктування, а також трансформувати модель у артефакти для розгортання в межах DevOps-процесів. Водночас подібне ПЗ має бути зручним у використанні та забезпечувати графічне представлення системи.

Ціль роботи

Розробити концепт ПЗ, основними завданнями якого є:

- реконструкція існуючої MSA-системи у вигляді моделі, придатної для опису архітектури за допомогою засобів ШІ;
- генерація моделі MSA з урахуванням функціональних і нефункціональних вимог із використанням ШІ;
- забезпечення інтерактивної взаємодії між ПЗ та інженером, що дає змогу вносити зміни на кожному етапі проєктування та надавати зворотний зв'язок моделі ШІ для її подальшого вдосконалення.

Матеріали та методи

У роботі використовуються блок-схеми та діаграми для представлення архітектурної складової запропонованого ПЗ, а також діаграми послідовності UML для відображення взаємодії користувача з системою.

Результати та обговорення

Концепт основних модулів ПЗ представлено на рис. 1. Інтерфейс користувача забезпечує прийняття запитів та надсилання їх до API (Application Programming Interface) з подальшим відображенням результатів у графічному форматі. API виконує обробку та валідацію запитів користувача, адаптує та доповнює запити до формату, необхідного для відповідного модуля ШІ, зберігає результати роботи та трансформує дані у формат, зручний для графічного представлення. Результатом роботи модуля ШІ, що відповідає за реконструкцію MSA, є модель системи, описана за допомогою спеціалізованої мови моделювання. Вхідними даними для модуля ШІ, який здійснює генерацію MSA, зокрема, є вже існуюча модель. Відповідно, результатом цього модуля також є модель.

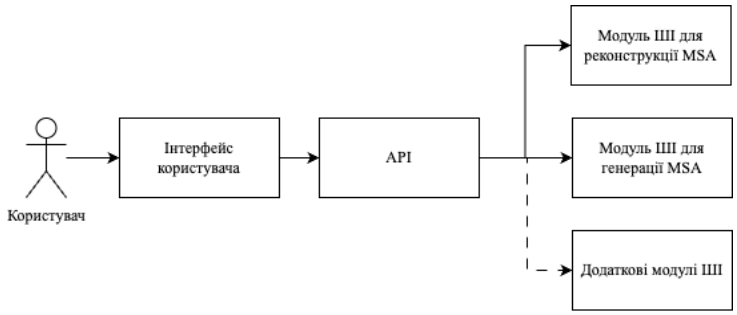


Рис. 1 – Основні модулі ПЗ

Взаємодія з програмним забезпеченням відбувається у два етапи. Перший етап, зображений на рис. 2, полягає у реконструкції моделі наявної MSA-системи. До переходу на другий етап інженер має можливість перевірити отриманий результат і, за потреби, внести корективи. Після затвердження результату (з урахуванням змін або без них), модуль ШІ отримує зворотний зв'язок для подальшого навчання. Другий етап, представлений на рис. 3,

полягає у генерації нової моделі на основі затвердженої реконструйованої архітектури, яка повинна відповідати заданим вимогам. Результати цього етапу також перевіряються користувачем, і за необхідності можуть бути внесені додаткові зміни. Після остаточного затвердження модуль ШІ знову отримує зворотний зв’язок для вдосконалення. Отримана фінальна модель може бути використана користувачем для документування рішень у форматі Architecture Decision Record (ADR), а також для подальшої імплементації.

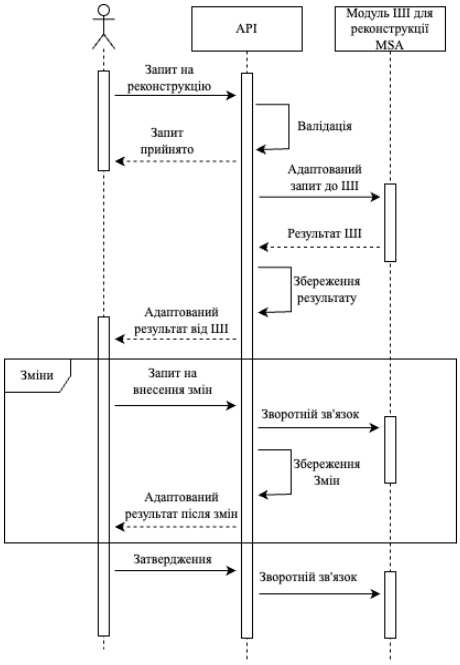


Рис. 2 – Діаграма послідовності взаємодії з ШІ модулем реконструкції MSA

Описаний функціонал є ядром системи. У подальшому до нього можуть бути додані додаткові модулі на основі ШІ для аналізу, надання рекомендацій і генерації шаблонів проєктування, а також для трансформації фінальної моделі в артефакти, які є придатними для розгортання у конкретному хмарному середовищі (AWS, GCP, Azure).

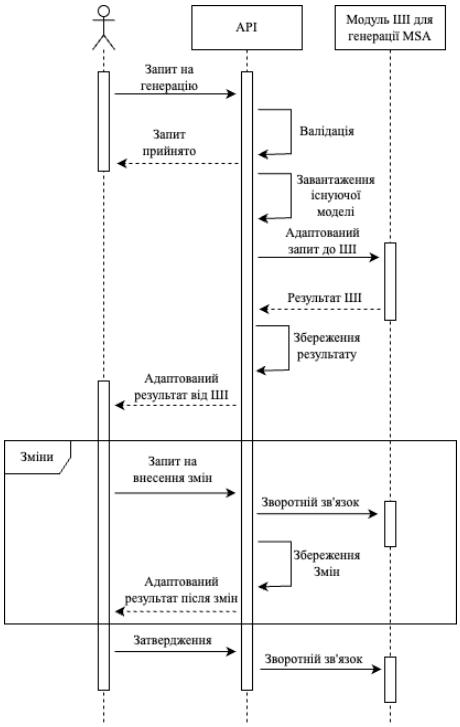


Рис. 3 – Діаграма послідовності взаємодії з ШІ модулем генерації MSA

Висновки

Запропонований концепт ПЗ передбачає використання ШІ для реконструкції MSA у вигляді моделі, описаної спеціалізованою мовою, з подальшим проєктуванням нової архітектури відповідно до заданих функціональних і нефункціональних вимог. Такий підхід дозволяє реалізувати цілісний процес проєктування, який охоплює аналіз наявної системи, інтерактивну взаємодію з інженером, можливість внесення змін та надання зворотного зв'язку для навчання ШІ. Модель є центром, що забезпечує узгодженість компонентів, підтримку графічного представлення, автоматичну генерацію проєктних рішень і документації, зокрема Architecture Decision Records (ADR). Результатом є модель, яка може бути трансформована в артефакти для розгортання в межах DevOps-процесів. Крім того, концепція підтримує масштабування шляхом додавання нових ШІ-модулів для аналізу архітектури, рекомендацій щодо шаблонів проєктування та адаптації до конкретних хмарних середовищ (AWS, GCP, Azure).

Список використаних джерел

1. Newman, S. Building microservices: Designing fine-grained systems. Sebastopol: O'Reilly, 2021. 612 p.

2. State of software architecture report 2024 [Електронний ресурс] // Medium. URL: https://icepanel.medium.com/state-of-software-architecture-report-2024-31eab5fe2c88 (дата звернення: 04.05.2025).

3. Esparza-Peidro, J., Muñoz-Escoí, F. D., Bernabéu-Aubán, J. M. Modeling microservice architectures // Journal of Systems and Software. 2024. Article ID 112041.

4. Granchelli, G., Cardarelli, M., Di Francesco, P., Malavolta, I., Iovino, L., Di Salle, A. MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems // 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). Gothenburg, Sweden, 2017. P. 298–302.

5. Narváez, D., Battaglia, N., Fernández, A., Rossi, G. Designing Microservices Using AI: A Systematic Literature Review // Software. 2025. Vol. 4, No. 1. Article No. 6.

Відомості про автора:

Моргун Артем Вікторович – аспірант 1-го року навчання кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: мікросервісна архітектура, штучний інтелект, моделювання програмних систем.

E-mail: 2175566@stud.kai.edu.ua

УДК 004.853:004.622:004.89(043.2)

ОПИС РЕАЛІЗАЦІЇ ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ OG-RAG

Олена ЧЕБАНЮК
Професор кафедри ІПЗ
Даниїл МОСКАЛЕНКО
Аспірант 1-го року навчання кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ

У роботі представлено підхід OG-RAG – модифікацію Retrieval-Augmented Generation (RAG), що використовує доменно-специфічну онтологію у вигляді гіперграфа для формування релевантного контексту до запитів користувача. Запропоновано спрощену архітектуру з використанням семантичного пошуку, ембедінгів та графових структур, яка забезпечує ефективну інтеграцію знань у відповідь LLM за умов обмежених ресурсів.

Ключові слова: *OG-RAG, LLM, RAG, гіперграф, онтологія, семантичний пошук, ембедінги.*

Вступ

Застосування великих мовник моделей (в подальшому LLM) охоплює все більше сфер суспільства. Однак через обмеженість навчальних даних вони можуть надавати недостовірну або неповну інформацію. Перенавчання моделі для охоплення нових сфер є досить трудомістким, тому для вирішення проблеми було розроблено технологію “Генерація з доповненням через пошук” (RAG).

RAG використовує здатність LLM розуміти людську мову та доповнює їхні знання за допомогою інформації з заданого набору документів з використанням сематичним або іншим методом пошуку [1,2]. Знайдена інформація формує контекст, який доповнює запит користувача, надаючи LLM додаткові дані для формування відповіді [2].

Проте RAG є обмеженим у формуванні точного контексту в сферах зі складними взаємозв'язками та специфічною термінологією [3]. Для вирішення цих задач було розроблено підходи з використанням онтологій та графів, які описують залежності між об'єктами та поняттями. Однією з таких технологій є OG-RAG.

Результати та обговорення

OG-RAG - це підхід, що використовує гіперграф, сформований на основі доменно-специфічної онтології, для здійснення пошуку з врахуванням вузлів і гіперребр, де гіперребра представляють кластери взаємопов'язаних фактів, що дозволяє здійснювати точний пошук контексту для LLM, [3].

Відповідно на підготовчому етапі передбачається формування доменно-специфічної онтології групою експертів на основі, якої формує гіперграф, [3].

Реалізація застосування технології описує процес використання технології OG-RAG після формування гіперграфу. При чому пошук контексту виконується за допомогою семантичного пошуку, який потребує попередніх перетворень даних у векторні представлення (в подальшому ембедінги). В залежності від методу пошуку і складності гіперграфа – його зав’язків підхід до реалізації може потребувати додаткових процесів і сутностей.

Реалізацію методу представлено за допомогою Діаграми послідовності (Рис. 1). Вона складається з таких сутностей, як:

«User» - користувач, який створює запити в застосунку..

«OG-RAG Module» - модуль реалізації OG-RAG..

«SentenceTransformer» - бібліотека для перетворення тексту в векторні представлення (ембедінги) для семантичного пошуку.

«FAIIS» – бібліотека семантичного пошуку даних.

«Anthropic API» – інтерфейс взаємодії (в подальшому API) з LLM «Clauda».

«NetworkX Graph» – бібліотека роботи з графами, для спрощеного представлення гіперграфу.

На етапі ініціалізації (Рис. 1) створюються ембедінги з документів (джерел) та вузлів гіперграфу. Для прискорення пошуку ембедінги індексуються в бібліотеці «FAISS».

При отриманні запиту користувача викликається метод «generate_response(query)», який створює ембедінг із запиту і викликає метод для формування контексту на основі гіперграфу, якщо ж нам не вдалося сформувати контекст пошук здійснюємо в джерелах даних.

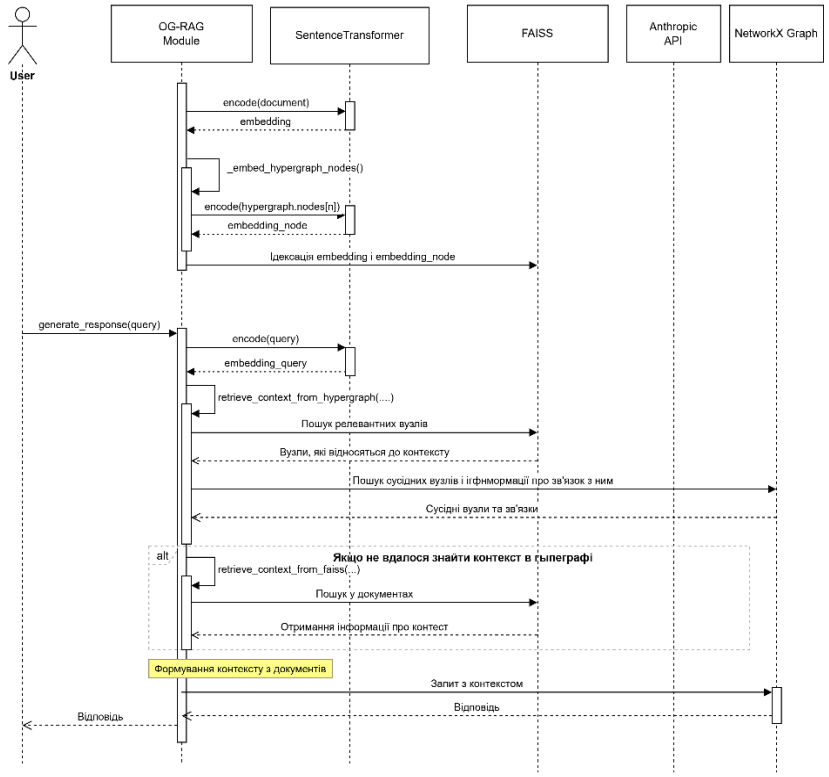


Рис. 1 - Діаграма послідовності застосування технології OG-RAG

Для формування контексту здійснюється семантичний пошук вузлів гіперграфу, який допевнюється жадібним пошуком їх гіперребер і пов’язаних вузлів. На основі результату пошуку формується контекст для запити.

Далі контекст об’єднується з запитом користувача, надсилається до LLM і результат повертається користувачу.

Висновки

OG-RAG дозволяє формувати контекст з використанням гіперребер і вузлів гіперграфу, що надає інформацію про взаємозв'язок понять, що своєю чергу збільшує точність відповіді. У даній роботі було представлено спрощений модуль застосування

технології OG-RAG з обмеженою кількістю ребер. Більш складна реалізація може не тільки включати більш складні зв'язки між компонентами, а й зв'язки компонентів з джерелами, що може використовуватись для додаткового доповнення контексту. Але в разі обмеженої кількості джерел і зв'язків з кожного компонента спрощена модель є більш вигідною в швидкості її впровадження. Загалом було продемонстровано реалізацію застосування OG-RAG, яка описує основні моменти його використання.

Список використаних джерел

1. What is Retrieval-Augmented Generation (RAG)? AWS. URL: https://aws.amazon.com/what-is/retrieval-augmented-generation/ (дата звернення: 20.04.2025)

2. Retrieval augmented generation (RAG). *Prompt Engineering Guide*. URL: https://www.promptingguide.ai/techniques/rag (дата звернення: 20.04.2025).

3. Sharma K., Kumar P., Li Y. OG-RAG: ontology-grounded retrieval-augmented generation for large language models. 2024. 22 с. (Препринт. Microsoft Research; arXiv:2412.15235). URL: https://arxiv.org/abs/2412.15235 (дата звернення: 20.04.2024).

Відомості про авторів:

Олена Вікторівна Чебанюк – професор кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технології Державного університету «Київський авіаційний інститут». *Наукові інтереси*: програмна архітектура, мобільна розробка, , MDA, MDD.

E-mail: olena.chebaniuk@npp.kai.edu.ua

Даниїл Олегович Москаленко – аспірант 1-го року навчання кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технології Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, семантичний пошук, онтологія.

E-mail: 4066207@stud.kai.edu.ua

УДК 004.415.2:004.8

ПРОГНОЗУВАННЯ ЗАХВОРЮВАНЬ НА ОСНОВІ ІСТОРИЧНИХ МЕДИЧНИХ ДАНИХ ПАЦІЄНТІВ У КОНТЕКСТІ ЇХ ПОДАЛЬШОГО ВИКОРИСТАННЯ У ВЕБ-СИСТЕМАХ МЕДИЧНОГО ПРИЗНАЧЕННЯ

Олег РАЗНО
Здобувач вищої освіти 1 курсу магістратури кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник доцент кафедри ІПЗ ФКНТ
Євген Олександрович Татаринов

Робота присвячена аналізу сучасних підходів до прогнозування захворювань на базі історії пацієнтів. Розглянуто та класифіковано основні методи прогнозування на статичні, машинного і глибокого навчання, та гібридні. Сформовано критерії оцінювання та здійснено порівняльний аналіз даних методів. Визначено найкращі підходи для створення модулю прогнозування у медичній системі відповідно до потреб.

Ключові слова: *прогнозування захворювань, методи прогнозування, машинне навчання, нейронні мережі.*

Вступ

Прогнозування захворювань на основі історичних медичних даних набуває особливого значення в умовах зростання обсягів клінічної інформації та потреби в індивідуалізованому підході до лікування пацієнтів. Завдяки впровадженню електронних медичних систем, телемедицини та мобільних застосунків у сфері охорони здоров’я створено технічне підґрунтя для збору, обробки та аналізу великої кількості даних, що дозволяє здійснювати аналітичні операції прогнозного характеру.

З медичної точки зору, актуальність задач прогнозування зумовлена необхідністю раннього виявлення потенційних загроз здоров’ю, своєчасного попередження розвитку хронічних та гострих станів, а також оптимізації ресурсів у системі охорони здоров’я. Зі сторони інформаційних технологій, задачі прогнозування є частиною ширшого напряму – застосування штучного інтелекту (ШІ) у медицині. Методи машинного навчання та аналізу даних дозволяють будувати моделі, які на основі історичних медичних записів здатні оцінювати ймовірність виникнення захворювання, розвитку ускладнень або ефективності лікування.

Особливу актуальність ця тема має в умовах дефіциту медичних кадрів, зростання кількості пацієнтів похилого віку та обмеженості бюджетних ресурсів. Тож, можна стверджувати, що дослідження сучасних підходів до прогнозування захворювань є не лише теоретично обґрунтованим, але й практично необхідним, особливо у складі медичних веб-систем, що надають доступ до цифрових медичних послуг.

Ціль роботи

Метою роботи є дослідження сучасних підходів до прогнозування захворювань, їх класифікація, порівняльний аналіз та формування найкращих підходів для створення модулю прогнозування у медичній системі відповідно до потреб.

Матеріали та методи

У процесі дослідження використовувались методи аналізу та синтезу інформації, компаративний аналіз підходів до машинного навчання, методи формалізації вимог, системного проєктування, а також елементи бібліографічного аналізу наукових джерел.

Результати та обговорення

Методи прогнозування, що застосовуються в медичних інформаційних системах, є центральним елементом сучасної цифрової медицини. Вони дозволяють на основі історичних даних робити висновки про ймовірний розвиток захворювань, оцінювати ризики ускладнень або прогнозувати ефективність лікування. Різноманіття доступних підходів дозволяє адаптувати вибір методу до особливостей конкретного медичного завдання, типу даних і потреб користувачів системи.

Залежно від принципів побудови та рівня складності, методи прогнозування можна умовно класифікувати на такі групи:

- Статистичні методи: це класичні математичні моделі, що засновані на припущеннях про розподіли даних та використовують формалізовані математичні залежності.
- Методи машинного навчання (ML): ці методи базуються на побудові моделей, здатних “навчатися” з даних без явного програмування логіки.
- Глибоке навчання (Deep Learning): це підрозділ машинного навчання, який застосовує штучні нейронні мережі для обробки складних типів даних: зображень, аудіо, тексту.
- Гібридні та спеціалізовані методи: до цієї групи входять поєднання декількох підходів (наприклад, ML + статистика, або логістична регресія + дерева рішень).

Алгоритми машинного навчання

1) Дерева рішень та ансамблеві методи:
   - Decision Tree (DT) – проста інтерпретована модель, яка будує прогноз на основі послідовного розгалуження умов.
   - Random Forest (RF) – комбінація великої кількості дерев, що підвищує стабільність і точність.
   - Gradient Boosting Machines (GBM, XGBoost, LightGBM) – потужні моделі, які поступово покращують прогноз, часто використовуються у змаганнях по машинному навчанню.
2) Алгоритми класифікації:
   - Support Vector Machines (SVM) – ефективні при великій кількості ознак і складному поділі класів. Часто застосовуються для діагностики за результатами обстежень.
   - k-Nearest Neighbors (k-NN) – базується на відстані до найближчих сусідів у навчальній вибірці, підходить для невеликих обсягів даних. Просто і зрозуміло, але працює краще з невеликими наборами даних.
   - Naive Bayes – використовує ймовірнісний підхід, добре працює з текстовими даними (наприклад, електронними записами лікаря).
3) Нейронні мережі та глибоке навчання:
   - Multilayer Perceptron (MLP) – це базова форма штучних нейронних мереж, яка складається з кількох шарів нейронів. Вона працює з табличними даними, такими як клінічні показники пацієнта.

- Convolutional Neural Networks (CNN) – спеціалізуються на обробці зображень і використовують фільтри, які виявляють важливі ознаки на знімках, такі як контури або текстури.
- Recurrent Neural Networks (RNN), зокрема вдосконалена версія LSTM – призначені для аналізу послідовностей даних, які змінюються з часом.
- Transformers, наприклад BERT і спеціалізований Med-BERT, є сучасними моделями для обробки тексту, які враховують контекст слів у реченні.

4) Автоматизоване машинне навчання (AutoML): це підхід, що дозволяє автоматизувати процес підбору алгоритму, налаштування параметрів і оцінки моделей.

Порівняльний аналіз методів прогнозування

Під час розробки медичних інформаційних систем з функціоналом прогнозування особливої уваги потребує обґрунтований вибір методів аналізу даних. Кожен із розглянутих попередніх підходів – статистичний, машинного навчання, або гібридний – має свої переваги та обмеження, що зумовлює доцільність їх використання залежно від конкретних цілей, типу медичних даних і вимог до точності, інтерпретації та обчислювальних ресурсів.

Критерії для порівняння методів:

- Точність прогнозу – рівень правильності передбачення майбутнього стану пацієнта.
- Інтерпретованість – можливість зрозуміти логіку прийняття рішення (важливо у клінічному середовищі).
- Складність реалізації – трудомісткість реалізації моделі в медичній інформаційній системі.
- Необхідність обробки великих даних – наскільки модель ефективно працює з великими та неструктурованими вибірками.
- Стійкість до пропущених/шумових даних – адаптивність до реальних медичних записів.

Таблиця 1.

Порівняльна таблиця методів

Метод	Точність	Інтерпре-тованість	Дані (обсяг)	Реалізація	Приклад використання
Логістична регресія	Середня	Висока	Малий	Легка	Ризик серцевих захворювань
Random Forest	Висока	Середня	Середній	Середня	Сортування пацієнтів
SVM	Висока	Низька	Невеликий	Середня	Виявлення онкозахворювань
k-NN	Середня	Висока	Малий	Легка	Схожість пацієнтів за симптомами
CNN	Висока	Низька	Великий	Складна	Розпізнавання МРТ
LSTM	Висока	Низька	Великий	Складна	Прогнозування стану в динаміці
AutoML	Залежить від конфігурації	Висока (часткова)	Середній	Легка	Різні задачі, без глибоких знань у ML

Тож, статистичні методи добре підходять для невеликих наборів даних, коли важлива інтерпретація. Вони можуть виступати базовими інструментами для початкового аналізу та перевірки гіпотез. Алгоритми машинного навчання демонструють кращу точність і стійкість

до складних даних, проте вимагають більших ресурсів і потребують відповідного досвіду. Глибокі нейронні мережі є незамінними для задач, що пов'язані з медичними зображеннями або часовими рядами, проте їх важко впровадити у звичайну медичну практику без спеціального інтерфейсу. AutoML-рішення можуть зменшити бар'єр входу для медичних закладів і допомогти впроваджувати ІС навіть без залучення фахівців з data science.

Висновки

Отже, при створенні модулю прогнозування у медичній системі доцільно розглядати комбіновані підходи: використовувати статистичні методи на ранніх етапах, а також моделі машинного навчання – для побудови фінального прогнозу. Остаточний вибір має базуватись на співвідношенні точності, складності впровадження, прозорості та клінічної доцільності.

Список використаних джерел

1. Miotto R., Wang F., Wang S., Jiang X., Dudley J. T. Deep learning for healthcare: review, opportunities and challenges // Briefings in Bioinformatics. – 2018. – Vol. 19, No. 6. – P. 1236–1246.

2. Mahmood T., Solomonides T. The role of big data and machine learning in clinical decision support systems: a survey // Journal of Biomedical Informatics. – 2022. – Vol. 128.

3. Kononenko I. Machine learning for medical diagnosis: history, state of the art and perspective // Artificial Intelligence in Medicine. – 2001. – Vol. 23, No. 1. – P. 89–109.

Відомості про автора:

Олег Сергійович Разно – здобувач вищої освіти 1 курсу магістратури кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технології Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, машинне навчання, веб-розробка.

E-mail: 6868062@stud.kai.edu.ua

УДК 004.78

ВИКОРИСТАННЯ ПРИНЦИПІВ МОДУЛЬНОСТІ У СУЧАСНИХ СИСТЕМАХ УПРАВЛІННЯ НАВЧАННЯМ

Катерина РІЗНІЧЕНКО
Здобувачка вищої освіти 4 курсу кафедри ІПЗ,
Державний університет «Київський авіаційний інститут»
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Олена Олегівна Колганова

У статті розглядаються особливості та передумови використання принципів модульності у сучасних спеціалізованих системах управління навчанням. Проведено аналіз застосування принципів модульності при проєктуванні застосунку з управління школою англійської.

Ключові слова: *спеціалізована система управління навчанням, управління школою, принцип модульності, проєктування спеціалізованої системи управління навчанням, проєктування архітектури.*

Вступ

У 21 столітті внаслідок активних світових процесів економічної, політичної та культурної інтеграції, що сприяє постійному обміну інформацією, знаннями, їх більшій доступності, а також стрімкому розвитку інформаційних технологій, постає актуальне питання цифровізації освіти. Можливим рішенням є впровадження спеціалізованих систем управління навчанням у освітньому процесі.

Система управління навчанням (СУН) – це інтегрований програмний засіб, який автоматизує обробку навчального контенту, облік та звітність щодо онлайн-курсів і програм, взаємодію з користувачами у вигляді здобувачів вищої освіти, забезпечуючи централізовану організацію навчального процесу, зокрема планування занять, реєстрацію учасників та оцінювання їхніх результатів та прогресу [1].

Проте постійний розвиток ІТ-технологій породжує зростання кількості нових вимог від користувачів, з’являється потреба інтеграції нових навчальних інструментів та функцій для ефективного конкурування на ринку. Це є ключовим викликом при проєктування систем управління навчання на сьогодні. Оптимальним підходом до подолання цього виклику є використання принципів модульності на етапі проєктування архітектури та власне її реалізації. Модульність передбачає поділ розроблюваної системи на окремі функціональні компоненти, що можуть розроблятися та підтримуватися незалежно один від одного [2].

Ціль роботи

Мета роботи – проаналізувати наявні теоретичні засади принципів модульності та практичний досвід їх використання у системах управління навчанням, визначити основні переваги та недоліки.

Матеріали та методи

За сучасних умов у сфері освіти та інформаційних технологій серед основних вимог до систем управління навчанням спостерігаються не лише базові функції адміністрування навчального процесу та його реалізації, а ще й адаптивність, масштабованість,

інтегрованість. Ефективно задовільнити зазначені вимоги можливо із використанням принципів модульності.

Модульність забезпечує максимальну гнучкість системи, адже функціональні компоненти, тобто модулі, можливо реалізовувати, оновлювати, змінювати, уникаючи фундаментальних змін у всій системі. Розроблювану систему управління навчанням можна легко та зручно розширити не порушуючи вже існуючий устрій та функціонал новими елементами за допомогою модулів. Подібний розподіл складного на менші, простіші компоненти полегшує розробку, процес тестування, супроводження. До того ж створені модулі можуть бути повторно використані, що є вкрай ефективно.

На практиці цей підхід успішно застосовується і популярних сучасних системах управління навчанням. Одними з прикладів використання принципів модульності є:

1. Moodle - популярна система управління навчанням, яка має відкритий код. У ній принципи модульності реалізуються через плагіни. Є окремі модулі для опитування та оцінювання здобувачів освіти, модулі управління контентом.
2. Canvas LMS є ще однією платформою, яка використовує модульний підхід через прикладний програмний інтерфейс, тобто API. У ній система оцінювання, чат, управління курсами реалізовані як окремі модулі. Це дозволяє легко адаптуватися під потреби специфічного навчального закладу.
3. Blackboard також реалізовує модульну архітектуру за допомогою використання плагінів, проте в більш закритому середовищі порівняно з Moodle або Canvas, і є в цілому менш гнучкою.

У контексті проєктування систем управлінням навчання, ключовими та найбільш значущими перевагами використання принципів модульності є підвищені показники гнучкості та масштабованості, адже конкретний навчальний заклад може мати свої потреби, необхідність в інтеграції та використанні специфічних інструментів і функціоналу, освітніх ресурсів. Подібну систему зручно оновлювати та підтримувати, особливо це актуально через плинність трендів та вимог у предметній області. Також варто зазначити, що ізоляція функціональних компонентів сприяє підвищеним показникам безпеки і ефективнішому тестуванню.

Однак у даного підходу є і власні недоліки, які треба висвітлити. По-перше, існує суттєва проблема при підтримці таких систем: необхідно слідкувати за актуальністю версій функціональних компонентів, оновленнями, їх сумісністю. Інша проблема, з якою може стикатися система, що використовує принцип модульності, це зниження продуктивності порівняно з монолітним підходом при комунікації модулів через API.

Результати та обговорення

У результаті проведеного аналізу теоретичних засад принципів модульності та практичного досвіду їх використання в системах управління навчанням, було визначено, що застосування цих принципів явно підвищує гнучкість та ефективність розроблюваної системи. За результатом дослідження можна зазначити, що сучасні системи активно користуються перевагами такого підходу при поділу на модулі для оцінювання, адміністрування, контенту та навчання.

Як наслідок використання модульності, система легше адаптується до стрімко змінних вимог та потреб ринку і освіти, розширює свій функціонал, масштабується та має потенціал до оперативних змін у собі. Проте існують і свої недоліки у вигляді складнощів при підтримці у вигляді проблеми сумісності та актуальністю версій, взаємодії модулів.

Висновки

Використання принципів модульності у спеціалізованих системах управління навчанням дозволяє ефективно відповідати на поставлені виклики у вигляді потреби у підвищеній гнучкості, ширших можливостях інтеграції, масштабування. Система, що реалізовує модульність швидше реагуватиме на зміни у вимогах, дозволятиме максимально ефективно реалізувати нові інструменти й функції без загроз до стабільності до всієї системи, проте жертвуючи показниками продуктивності при взаємодії модулів та ресурсами під час етапу підтримки.

Список використаних джерел

1. Мохд Касім Н. Н., Халід Ф. Choosing the Right Learning Management System (LMS) for the Higher Education Institution Context: A Systematic Review // International Journal of Emerging Technologies in Learning (iJET). – 2016. – Т. 11, № 06. – С. 55–61. – DOI: https://doi.org/10.3991/ijet.v11i06.5644 (дата звернення: 29.04.2025).

2. Шарма А. What is Modularity in Software Engineering | Institute of Data [Електронний ресурс]. – Institute of Data. – Режим доступу: https://www.institutedata.com/blog/modularity-in-software-engineering/ (дата звернення: 29.04.2025).

Відомості про автора:

Різніченко Катерина Віталіївна – здобувачка вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, модульність, системи управління навчанням.

E-mail: 7366109@stud.kai.edu.ua

УДК 004.8:004.42

ІНТЕЛЕКТУАЛЬНИЙ АГЕНТ ДЛЯ ПОДОРОЖЕЙ: КОНЦЕПЦІЯ, АРХІТЕКТУРА, РЕАЛІЗАЦІЯ.

Артем РОЗУМ
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Яна Андріївна Бєлозьорова

У статті розглянуто розвиток інтелектуальних агентів та окреслено можливості створення власного ШІ-агента для подорожей. Детально описано принцип його роботи, архітектуру, використані інструменти та зовнішні сервіси. Також проаналізовано доступність реалізації навіть для користувачів без досвіду та окреслено основні технічні виклики.

Ключові слова: *штучний інтелект, ШІ-агент, подорожі, цифровий помічник, маршрут, API, мовна модель, Telegram-бот.*

Вступ

Станом на початок 2025 року штучний інтелект (ШІ) перестав бути технологією суто академічного або лабораторного характеру. Він впевнено закріпився в повсякденному житті, знайшовши практичне застосування у безлічі сфер – від автоматизації бізнесу до персональних цифрових помічників. Завдяки таким продуктам, як ChatGPT, Midjourney, Claude та іншим, мільйони людей щодня взаємодіють із інтелектуальними системами, не заглиблюючись у технічні деталі їхньої роботи. Проте на зміну пасивному використанню нейромереж приходить новий рівень технологічної взаємодії – створення власних ШІ-агентів, здатних до автономного мислення, цілеспрямованої діяльності та адаптації до нових умов.

Поняття штучного інтелектуального агента (ШІ-агента) відрізняється від простої мовної або візуальної моделі. Це програмна система, яка функціонує автономно, сприймає середовище через сенсорні канали (вхідні дані), аналізує отриману інформацію, приймає рішення на основі заданих або здобутих знань і впливає на середовище за допомогою виконавчих механізмів. Ключовими ознаками ШІ-агента є: автономність, проактивність, здатність до навчання, наявність цілі та здатність адаптуватися до змін.

Залежно від завдання, агент може працювати у сфері юриспруденції, медицини, освіти, маркетингу або творчості. Зокрема, дедалі більшої актуальності набуває розробка інтелектуальних агентів-помічників для мандрівників. В час, коли люди подорожують частіше, стикаються з мовними бар’єрами, нестачею інформації або непередбачуваними ситуаціями, виникає потреба в персоналізованому цифровому супутнику. Такий агент здатен значно спростити навігацію в новому місті, підібрати оптимальний маршрут, забронювати житло, знайти заклади харчування відповідно до уподобань, а також попередити про погодні умови чи інші загрози.

Ціль роботи

Метою написання статті є комплексне дослідження концепції штучного інтелектуального агента та можливостей його застосування у сфері подорожей. У межах

дослідження передбачено розгляд теоретичних основ побудови ШІ-агентів, їхніх структурних компонентів, принципів функціонування та класифікації за типами взаємодії із середовищем.

Окрему увагу приділено практичним аспектам розробки таких систем, включаючи вибір інструментів програмування, джерел даних, інтерфейсів взаємодії з користувачем. У ході дослідження також ідентифіковано потенційні ризики, пов'язані з використанням ШІ-агентів, зокрема – питання безпеки, конфіденційності та достовірності отриманої інформації.

Історія розвитку ШІ-агентів

Історія штучних інтелектуальних агентів розпочалася значно раніше, ніж може здаватися. Перші віртуальні агенти існували як прості текстові програми, здатні відповідати лише на базові запитання. Хоча вони не мали справжнього інтелекту, ці рішення започаткували нову еру в людсько-машинній взаємодії.

Одним із перших голосових пристроїв стала іграшка Radio Rex (1916), яка реагувала на своє ім'я. У 1952 році Bell Labs створила систему Audrey, що розпізнавала цифри з голосу – важливий крок у розвитку мовного розпізнавання. У 1960-х роках з'явилася відома програма ELIZA, яка імітувала спілкування з психотерапевтом і продемонструвала, наскільки люди схильні приписувати машині людські риси.

У 1970-х роках в Університеті Карнегі-Меллона створили систему HARPY, яка розуміла близько 1000 слів. У 1980-х IBM розробила машинку Tangora, яка вже могла розпізнавати 20 000 слів. Подальший розвиток відбувався завдяки поширенню персональних комп'ютерів та збільшенню обчислювальних можливостей.

У 1990-х з'явилася перша комерційна система розпізнавання мовлення – Dragon Naturally Speaking, яка в реальному часі перетворювала голос у текст. А у 1994 році вийшов IBM Simon – перший смартфон з елементами цифрового помічника.

Справжній прорив настав у 2011 році з появою Siri від Apple – першого повноцінного голосового асистента для смартфонів. У 2014 році компанія Amazon представила Alexa, що стала ядром розумного дому і розширила межі взаємодії з технікою.

Сьогодні віртуальні агенти еволюціонували у складні ШІ-системи, здатні вести природну розмову, розуміти контекст, адаптуватися до користувача та навіть передбачати його потреби. Вони вже інтегровані у повсякденне життя – від служби підтримки й мобільних додатків до автономних корпоративних рішень.

Великим проривом стала поява мовних моделей великого масштабу (LLM), зокрема GPT-серії від OpenAI. У 2015 році було засновано OpenAI, у 2018 з'явилася модель GPT-1, а у 2022 – ChatGPT, який відкрив можливість масового доступу до мовного ШІ. У 2023 вийшла GPT-4, що суттєво розширила функціональність: контекстуальність, точність, мультимодальність.

Сучасні агенти стали не просто реактивними системами, а проактивними цифровими співрозмовниками, що навчаються, адаптуються, ініціюють взаємодію й інтегруються з іншими сервісами. Це вже не просто інструменти, а повноцінні цифрові партнери, що формують новий рівень взаємодії між людиною та ШІ.

Технічні аспекти функціонування ШІ-агентів

Розуміння принципів функціонування агентів штучного інтелекту потребує ознайомлення з основними концепціями, структурними компонентами, типами агентів та механізмами їх навчання.

а. Основні компоненти ШІ-агента. Будь-який агент штучного інтелекту виконує цикл: сприйняття → міркування → дія → навчання, що забезпечує його адаптивну поведінку:

Сприйняття. Агент сприймає навколишній світ за допомогою сенсорів або механізмів введення. Це можуть бути фізичні сенсори (камери, мікрофони, GPS) або цифрові джерела даних. Наприклад, в автономних транспортних засобах використовуються LIDAR, камери та радари для аналізу простору навколо автомобіля.

Міркування. Після збору даних агент здійснює логічну або статистичну обробку інформації, аналізує контекст і приймає рішення. У системах рекомендацій, наприклад, агент враховує попередню поведінку користувача, щоб запропонувати релевантний контент або продукт.

Дія. На основі результатів міркування агент вживає відповідних дій для досягнення поставленої мети. Це можуть бути фізичні дії (наприклад, переміщення робота) або цифрові дії (відправка повідомлення, оновлення даних).

Навчання. Агенти можуть удосконалювати свою поведінку завдяки накопиченню досвіду. Залежно від підходу, вони можуть навчатися у контрольованому чи неконтрольованому режимі або шляхом підкріплення.

б. Типи ШІ-агентів. Різноманітність агентів визначається їхньою здатністю до адаптації, автономності та складністю логіки поведінки. Серед основних типів виділяють:

Прості рефлекторні агенти — реагують безпосередньо на подразники з оточення відповідно до жорстко заданих правил. Прикладом є звичайний термостат, який змінює температуру залежно від виміряного значення.

Рефлекторні агенти на основі моделі – зберігають у пам’яті внутрішню модель середовища, що дозволяє враховувати приховану або частково недоступну інформацію. Такі агенти, наприклад, використовуються в навігаційних системах.

Цілеспрямовані агенти – приймають рішення, орієнтуючись на поставлену мету, і обирають дії, які наближають їх до досягнення бажаного результату. Приклад – агент планування задач із урахуванням дедлайнів.

Агенти на основі корисності – оцінюють не лише досягнення мети, а й оптимальність шляху до неї, обираючи варіант із максимальною вигодою. Наприклад, у фінансових системах такі агенти аналізують ризики й прибутковість перед ухваленням угоди.

Агенти, що навчаються – самостійно покращують свою поведінку з часом, накопичуючи досвід та адаптуючись до нових умов. Такі агенти використовуються в системах рекомендацій, де відгуки користувачів допомагають краще налаштовувати майбутні пропозиції.

в. Механізми навчання ШІ-агентів. Існує кілька підходів до навчання агентів, кожен із яких застосовується залежно від характеру завдань та доступних даних:

Навчання під наглядом (supervised learning). Агент тренується на основі пар “вхід–вихід”, які задаються вручну. Це дозволяє моделі навчитися прогнозувати правильний результат. Такий метод широко використовується для розпізнавання зображень, голосу, класифікації тощо.

Навчання без нагляду (unsupervised learning). Застосовується для роботи з непозначеними даними. Агент самостійно шукає закономірності та структури (наприклад, кластери або тематичні групи). Це корисно у сегментації клієнтів, виявленні аномалій та візуалізації даних.

Навчання з підкріпленням (reinforcement learning, RL). Агент навчається через взаємодію зі середовищем, отримуючи винагороди за правильні дії та штрафи за помилки. Це дає змогу вивчати ефективну послідовність дій у динамічному середовищі, як-от у відеоіграх, робототехніці або фінансових симуляціях.

Таким чином, ШІ-агенти є результатом інтеграції кількох ключових технологій – обробки даних, логічного виведення, адаптивного навчання та прийняття рішень у реальному часі. Залежно від цілей і контексту застосування, їхня архітектура та поведінкові алгоритми можуть суттєво відрізнятися, що відкриває широкі можливості для розробки індивідуальних рішень, зокрема в таких динамічних сферах, як туризм і подорожі.

Створення власного ШІ-агента для подорожей. Принцип роботи

Інтелектуальний агент для подорожей виконує функцію персонального цифрового асистента, здатного автономно взаємодіяти з користувачем у реальному часі, враховуючи контекстуальні фактори. Взаємодія починається із запиту користувача – текстового або голосового, через мобільний застосунок, вебінтерфейс чи чат-бот (наприклад, Telegram). Запити можуть бути як простими, так і складними: «Порадь маршрут на 3 дні у Вільнюсі», «Де недорого поїсти у центрі Львова?».

Після отримання запиту агент використовує модуль обробки природної мови (NLP), щоб розпізнати інтенцію, витягнути ключові параметри (локацію, час, ціль) і перейти до аналізу контексту. У цей момент враховуються геолокація, час доби, погода, події або транспортна доступність (через Google Maps API чи локальні джерела), а також історія попередніх взаємодій.

Рис. 1 - Принцип роботи та взаємодія користувача з ШІ-агентом для подорожей

Далі агент формує релевантну відповідь: добирає маршрути, події чи місця, використовуючи логіку на основі правил і персональних налаштувань. Відповідь може включати карту, рейтинг, опис, кнопки дій та візуальні елементи. Користувач має змогу уточнити запит (наприклад, «додай музеї» або «лише пішохідні маршрути»), і агент оновлює відповідь без потреби у повторному формулюванні.

З часом агент вивчає вподобання користувача, фіксує реакції, формує персоналізований профіль і може сам ініціювати взаємодію – наприклад, попередити про дощ, запізнення транспорту чи цікаву подію поруч. Таким чином, функціонування агента стає не лише реактивним, а й проактивним, що підвищує комфорт та ефективність взаємодії. Для кращого розуміння принципу роботи нижче подано блок-схему (рис. 1).

Створення власного ШІ-агента для подорожей: інструменти, структура та реалізація

Розробка інтелектуального агента для подорожей – це завдання, яке вже сьогодні може реалізувати окрема особа або невелика команда, використовуючи відкриті програмні інструменти та доступні API. Такий агент може стати персональним цифровим помічником, здатним відповідати на запити користувача в режимі реального часу, будувати маршрути, надавати поради, враховувати вподобання й контекст подорожі.

Передусім потрібно визначити, як саме працюватиме агент. Основна ідея – надати користувачеві можливість ставити природні запити (наприклад: «що подивитися у Києві протягом дня?» або «де поїсти поруч?»), які агент обробляє, визначає ключову інформацію (локацію, тип запиту, інтереси) і видає адаптовану відповідь на основі зовнішніх джерел – наприклад, Google Maps. Суть полягає в тому, щоб поєднати розуміння мови (через мовну модель) з можливістю отримати реальні дані та сформувати корисну, контекстну відповідь.

У центрі такої системи знаходиться мовна модель, яка відповідає за обробку запитів. Найпростіший спосіб реалізації – використання OpenAI API (ChatGPT). Це дозволяє інтегрувати потужну модель обробки природної мови без потреби тренувати власну. Якщо пріоритет – open-source рішення, можна звернутися до бібліотек Hugging Face Transformers і моделей на кшталт Mistral або LLaMA, які підтримують розпізнавання інтенцій та генерацію тексту.

Для взаємодії з користувачем найзручніше створити бота в Telegram, оскільки ця платформа підтримує швидку розробку, просту інтеграцію через Telegram Bot API, і не вимагає створення складного інтерфейсу. Бот приймає повідомлення, передає їх у мовну модель, отримує оброблений запит, підключає зовнішні джерела даних і повертає відповідь.
Для роботи з геоданими, маршрутами та точками інтересу використовуються зовнішні API-сервіси:

- Google Maps API – для пошуку закладів, побудови маршрутів, перевірки заторів;
- OpenWeatherMap – для актуальної погоди в заданій точці;
- Booking.com API або TripAdvisor API – для пошуку готелів, ресторанів та відгуків;
- Transport API, Easy Way – якщо потрібна інтеграція з громадським транспортом (наприклад, у містах ЄС).

Щоб ці сервіси працювали, достатньо мати акаунт розробника, отримати ключі доступу (API keys), після чого можна надсилати HTTP-запити до них із бекенду. Всі дані, які повертає API, зазвичай надходять у форматі JSON, що легко обробляється в Python.

Логіку обробки запитів найкраще реалізувати за допомогою мови Python, яка має все необхідне для інтеграції зі сторонніми сервісами, роботи з API, реалізації чат-ботів і використання моделей ШІ. Для обробки запитів використовується або Flask, або FastAPI — це легкі вебфреймворки, які дозволяють запускати сервер, що приймає повідомлення від користувача, обробляє їх і повертає відповіді. FastAPI є більш сучасним і дозволяє швидко створювати асинхронні API-сервіси.

Також знадобиться база даних, у якій зберігатиметься профіль користувача, історія взаємодії, збережені маршрути чи уподобання. Для цього можна використовувати SQLite на перших етапах або Firebase для мобільного інтегрованого рішення.

Для побудови логіки агента (ланцюга дій: розпізнай → проаналізуй → отримай дані → сформуй відповідь → адаптуйся) зручно використовувати фреймворк LangChain, який дозволяє об'єднувати запити до LLM, зовнішні API, базу даних і логіку виклику функцій. Він особливо корисний тоді, коли потрібно формувати багатокрокові відповіді, зберігати контекст діалогу або автоматизовано підключати зовнішні функції (наприклад, отримати прогноз погоди, якщо користувач хоче піти в похід).

Ключова перевага створення такого агента – гнучкість і персоналізація. Можна створити систему, яка працює лише в межах одного міста, орієнтована на один тип подорожей (наприклад, бюджетні поїздки, еко-маршрути), або ж розширити її до повноцінного міжнародного асистента.

Проте важливо враховувати і виклики. До основних належать:

- обмеження у безкоштовному доступі до API (наприклад, Google Maps має ліміт на запити без оплати);
- затримки у відповіді – якщо кожен запит проходить через кілька API, відповідь може формуватись повільно;
- мовні труднощі – якщо агент має працювати багатомовно, слід інтегрувати перекладач або обрати модель з підтримкою кількох мов;
- захист персональних даних – якщо агент зберігає профілі користувачів, важливо дотримуватись вимог конфіденційності;
- надійність – іноді зовнішні сервіси можуть бути тимчасово недоступними, тому слід передбачити систему повідомлень про помилки.

Підсумовуючи, можна стверджувати: створення власного ШІ-агента для подорожей – цілком реалістичне завдання, яке не потребує складного обладнання або роками напрацьованих знань. Завдяки хмарним API, готовим моделям і відкритим бібліотекам, сьогодні кожен охочий може реалізувати інтелектуального цифрового помічника, який стане частиною нової епохи подорожей – розумних, адаптивних і персоналізованих.

Висновки

У ході роботи було розглянуто розвиток інтелектуальних агентів – від перших текстових систем до сучасних ШІ-помічників. Особливу увагу приділено створенню власного тревел-агента – цифрового асистента, здатного працювати в реальному часі, взаємодіяти з користувачем, аналізувати контекст та надавати персоналізовані рекомендації. Розкрито принцип його роботи: розпізнавання запитів, підключення до зовнішніх сервісів, обробка даних і генерація відповідей. Розробка такого агента сьогодні є доступною завдяки відкритим платформам і докладній документації. Навіть без досвіду, можливо створити функціональний прототип за кілька тижнів. Це поєднує програмування, аналіз і логіку, дозволяючи отримати практичне рішення для реального застосування. Проєкт також виявляє типові труднощі – залежність від зовнішніх API, конфіденційність і потребу в адаптації до змін.

Попри виклики, ШІ-агенти залишаються перспективним і динамічним напрямом. Створення власного агента – це крок від пасивного користування технологіями до активної участі в їх розробці. Такий досвід поєднує технічні навички з реальним застосуванням і відкриває шлях до глибшого розуміння штучного інтелекту.

Список використаних джерел

1. Warley W. Understanding AI agents: how they work, types and practical applications [Електронний ресурс] / William Warley. – Режим доступу: https://medium.com/@williamwarley/understanding-ai-agents-how-they-work-types-and-practical-applications-bd261845f7c3, вільний. – Дата звернення: 11.04.2025.

2. What are AI agents – a guide by GitHub [Електронний ресурс]. – Режим доступу: https://github.com/resources/articles/ai/what-are-ai-agents, вільний. – Дата звернення: 11.04.2025.

Відомості про автора:

Розум Артем Олегович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, штучний інтелект, концептуальні агенти.

E-mail: 5678775@stud.kai.edu.ua

УДК 004.4

ШАБЛОНІЗАЦІЯ ФУНКЦІОНАЛЬНОГО КОДУ ДЛЯ ВЕБЗАСТОСУНКІВ НА ОСНОВІ GPT-4

Артем РОМАНЧУК
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут»
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Яна Андріївна Бєлозьорова

У роботі розглянуто підхід до автоматизованої генерації вихідного коду для вебзастосунків на основі багаторазового шаблону, побудованого за архітектурою MVC. Запропонований шаблон інтегрується з мовною моделлю GPT-4 для створення функціональних компонентів за текстовим описом бізнес-логіки. Методика перевірена на прикладі системи бронювання, підтверджено її ефективність і придатність до масштабування. Отримані результати можуть бути основою для створення генеративних фреймворків нового покоління.

***Ключові слова**: автоматизація програмування, багаторазовий шаблон, штучний інтелект, GPT-4, MVC, генерація коду.*

Вступ

Швидкість і якість розробки програмного забезпечення є ключовими факторами успішності сучасних IT-проєктів [1]. Значна частина роботи припадає на реалізацію функціональних сценаріїв, що мають схожу структуру: введення даних, перевірка, підтвердження, збереження, надсилання повідомлень. Прикладами таких процесів можуть бути реєстрація користувача, скидання пароля, залишення відгуку тощо. Їхня повторюваність у багатьох вебзастосунках призводить до дублювання коду, ускладнення супроводу та зростання ймовірності помилок.

Одним з ефективних способів розв’язання цієї проблеми є шаблонний підхід: повторювані елементи архітектури винесено в параметризовані блоки, які можна адаптувати під нову логіку шляхом зміни окремих параметрів. Розвиток штучного інтелекту, зокрема мовних моделей, таких як ChatGPT (GPT-4) [2], відкриває нові можливості для автоматизованої генерації таких шаблонів за текстовим описом, що суттєво прискорює розробку.

У межах цього підходу запропоновано та реалізовано багаторазовий шаблон вихідного коду для вебзастосунків, побудованих на архітектурі MVC [3]. Такий шаблон передбачає генерацію ключових компонентів архітектури: контролерів, шару сервісів із бізнес-логікою, репозиторіїв для доступу до даних і представлень. Крім того, шаблон охоплює створення нових сутностей у базі даних на основі опису бізнес-логіки, ViewModels для обміну даними між сервісами та представленнями, та уніфіковану генерацію шаблонів email-повідомлень для типових процесів підтвердження дій чи інформування про успішне виконання.

Ефективність запропонованого рішення продемонстровано на прикладі повнофункціонального застосунку для бронювання номерів у готелі, а також споріднених процесів – бронювання паркомісця та залишення відгуку. Завдяки шаблонізації реалізація

нових функцій потребує мінімальних зусиль з боку розробника, зберігаючи при цьому структурну єдність і масштабованість проєкту.

Мета та методи дослідження

Метою дослідження є розробка багаторазового шаблону вихідного коду для автоматизованого створення однотипних функціональних компонентів вебзастосунку на базі архітектури Model–View–Controller (MVC), із використанням сучасних засобів програмної інженерії та штучного інтелекту. Шаблон має бути структурований із чітким розмежуванням на [REQUIRED] та [CUSTOM] секції, що дозволяє відокремлювати незмінні елементи логіки від параметризованих компонентів. Основна увага зосереджена на уніфікації повторюваних бізнес-процесів, таких як бронювання, підтвердження дій та обробка введених користувачем даних.

У ході дослідження застосовано такі методи:

- аналіз сучасних підходів до генерації коду – порівняно scaffolding, low-code/no-code рішення та генерацію коду на основі LLM;
- аналіз предметної області – вивчено типові шаблони поведінки у вебзастосунках, які мають спільну структуру;
- структурне проєктування – побудовано гнучкий шаблон з параметризованими елементами для автоматичної генерації однотипного функціоналу;
- використання мовної моделі GPT-4 – сформовано фрагменти коду на основі текстових інструкцій з описом бізнес-логіки відповідно до структури шаблону;
- експериментальна перевірка – реалізовано вебзастосунок для бронювання номерів у готелі та протестовано повторне використання шаблону на прикладах бронювання паркомісця та залишення відгуку;
- модульне й інтеграційне тестування – перевірено коректність взаємодії компонентів шаблону, стійкість до змін і архітектурну узгодженість, а також виявлено та усунуто недоліки початкової реалізації шляхом рефакторингу.

Результати дослідження

У ході дослідження проаналізовано сучасні підходи до автоматизованої генерації коду у веброзробці, включаючи scaffolding, low-code/no-code платформи та генерацію коду на основі великих мовних моделей (LLM), таких як GPT-4. Найбільш гнучким і придатним для розширення виявився підхід із використанням LLM, здатних створювати повторювані фрагменти коду на основі текстових інструкцій з урахуванням контексту предметної області.

На основі доменного аналізу встановлено, що типові функціональні процеси – зокрема, бронювання номерів, резервування паркомісць і залишення відгуків – мають спільну архітектурну структуру. Це дало змогу уніфікувати їх реалізацію за допомогою спільного багаторазового шаблону.

Розроблений багаторівневий шаблон охоплює всі ключові компоненти архітектури MVC: контролери, сервіси бізнес-логіки, репозиторії для роботи з базою даних, ViewModel-и, представлення (views), логіку створення нових сутностей і шаблони email-повідомлень. Шаблон структурується на основі чітких маркерів [REQUIRED] і [CUSTOM], що дозволяє відокремлювати загальні компоненти від змінних частин, залежно від конкретного функціонального сценарію.

Практичне застосування шаблону в реалізації процесів бронювання паркомісця та залишення відгуку підтвердило його здатність до автоматизованого створення повноцінного функціоналу за мінімального втручання розробника. Завантаження шаблону в середовище

ChatGPT разом із текстовим описом бізнес-логіки дозволило автоматично згенерувати всі необхідні компоненти для реалізації вказаних процесів.

Модульне й інтеграційне тестування підтвердило правильність роботи шаблону, узгодженість взаємодії між компонентами та стабільність виконання. У процесі тестування виявлено низку архітектурних обмежень початкової реалізації, які були усунені шляхом рефакторингу. В результаті покращено організацію коду, досягнуто кращого розділення відповідальностей і підвищено стійкість системи до змін (Рис.1.).

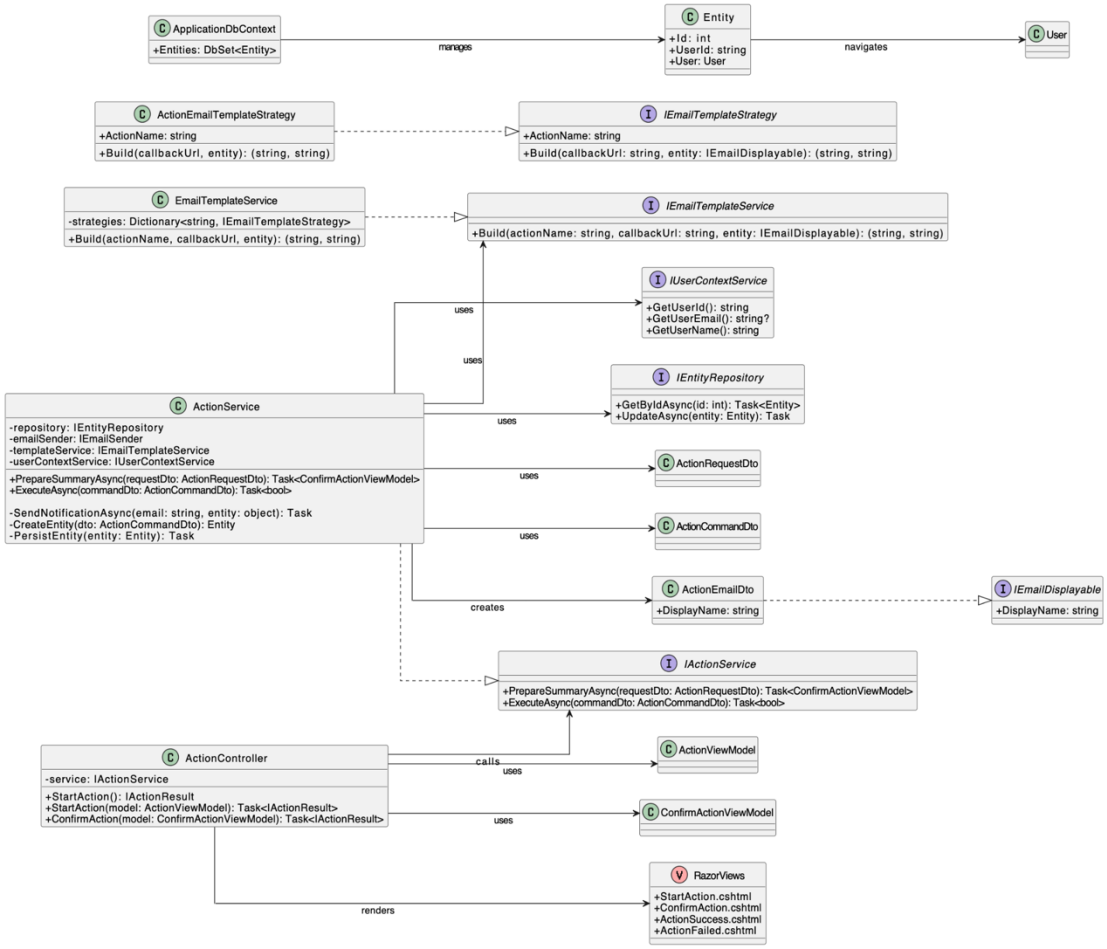


Рис.1 – Діаграма класів компонентів створеного шаблону вихідного коду

Висновки

Отримані результати підтверджують ефективність шаблонного підходу для розробки вебзастосунків. Шаблон придатний для масштабування на інші типи проєктів, особливо в доменах із повторюваними сценаріями: системи бронювання, підтвердження дій, багатокрокова валідація даних, адміністрування тощо. Методика має практичну цінність як для професійної розробки, так і для освітніх цілей, зокрема для вивчення принципів багаторівневої архітектури, автоматизації розробки та тестування ПЗ.

Попри досягнуті результати, розробка шаблонів залишається складним і трудомістким завданням, що вимагає володіння prompt engineering та архітектурними підходами до створення масштабованих рішень. Перспективи подальших досліджень охоплюють вдосконалення взаємодії між компонентами шаблону, підтримку складніших бізнес-процесів, а також інтеграцію з інструментами CI/CD. Запропонований підхід може стати основою для створення генеративних фреймворків нового покоління.

Список використаних джерел

1. Sommerville I. *Software Engineering*. 10th ed. – Pearson Education, 2015.

2. OpenAI. *GPT-4 Technical Report*. – 2023. [Електронний ресурс]. – Режим доступу: https://openai.com/research/gpt-4.

3. Microsoft Learn. *ASP.NET MVC Overview*. – [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview.

Відомості про автора:

Романчук Артем Олександрович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: Програмування, Інформаційні технології.

E-mail: 7413778@stud.kai.edu.ua

УДК 004.78

АВТОМАТИЗАЦІЯ УПРАВЛІННЯ ОНЛАЙН ТОРГІВЛЕЮ

Марія РЮМШИНА
Здобувачка вищої освіти 4 курсу кафедри ІПЗ,
Державний університет «Київський авіаційний інститут»
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Олена Олегівна Колганова

У статті висвітлюються основні принципи розробки вебсайту для внутрішнього управління магазином інструментів, що включає створення бази даних для обліку товарів і покупців, розробку адміністративної панелі для керування асортиментом і автоматизацію процесів продажу.

Ключові слова: *вебсайт, магазин інструментів, база даних, управління магазином, облік товарів.*

Вступ

Сучасна торгівля все більше переходить в онлайн-середовище. Магазини потребують не лише сайтів для продажу товарів, а й зручних інструментів для управління своїм асортиментом. Розробка вебсайту для внутрішнього використання допомагає власникам краще контролювати наявність товарів, обробляти замовлення і працювати з базою покупців. Особливо важливою є наявність простої у користуванні адміністративної панелі. Вона дозволяє оперативно оновлювати інформацію про товари, змінювати ціни, додавати новинки та стежити за залишками на складі. Основою такого застосунку є база даних, що зберігає всю необхідну інформацію. Розробка подібних сайтів потребує знань у сфері вебтехнологій, роботи з базами даних та основ електронної комерції. У даній статті розглянуто процес створення вебдодатку для управління магазином інструментів. Особливу увагу приділено питанням функціональності, зручності користування та ефективної організації даних. Запропоноване рішення спрямоване на спрощення щоденних бізнес-процесів для продавців.

Ціль статті

Метою статті є проєктування розробки вебсайту для ефективного управління асортиментом магазину інструментів. Сайт має забезпечувати зручне додавання, редагування та облік товарів, а також обробку інформації про покупців. Особлива увага приділяється створенню бази даних та реалізації функціоналу адміністративної панелі для автоматизації бізнес-процесів. Запропоноване рішення повинно сприяти підвищенню оперативності та точності ведення обліку в магазині.

Матеріали та методи

Під час написання статті було проаналізовано особливості побудови вебсайтів для внутрішнього використання, зокрема для управління магазином інструментів. Основну увагу приділено вибору відповідних технологій, які дозволяють створити зручний інтерфейс і забезпечити надійне збереження даних. Розглядалися можливості використання мови програмування Python для реалізації серверної логіки. Для розробки зовнішнього вигляду та побудови сторінок було обрано HTML і CSS як базові інструменти фронтенду.

Як систему збереження даних було обрано MySQL, що дозволяє створювати структури з чіткими зв’язками між таблицями [1]. На основі функціональних вимог до сайту

було змодельовано базу даних із таблицями для товарів, покупців, категорій, замовлень і деталей замовлень. Структура бази даних побудована за принципами нормалізації та враховує потреби в масштабуванні у майбутньому. Особлива увага приділена розробці ER-діаграми, яка наочно демонструє логіку взаємозв'язків між об'єктами системи. Також було сформульовано вимоги до адміністративної панелі, де передбачено можливість додавання, редагування та видалення товарів. Проєктування інтерфейсу здійснювалося з орієнтацією на простоту, мінімалізм і доступність навіть для користувачів без технічної підготовки.

Результати та обговорення

У результаті проведеної роботи вдалося побудувати концепцію вебсайту, який може використовуватись для внутрішнього управління магазином інструментів. Було окреслено основні елементи інтерфейсу, такі як головна сторінка з каталогом товарів, можливість фільтрації за категоріями, а також адміністративна панель. Передбачено, що адміністратор матиме змогу вносити зміни до бази товарів безпосередньо через зручні HTML-форми. Усі ці дії пов'язані з базою даних, що дозволяє оперативно відображати зміни на сайті.

Ключовим елементом є структура бази даних, яка забезпечує логічний поділ інформації на категорії та дозволяє ефективно працювати з клієнтськими замовленнями. Усі сутності та зв'язки між ними показано на ER-діаграмі (див. рис. 1), де чітко видно, як кожен товар прив'язаний до певної категорії, а кожне замовлення – до конкретного покупця. Структура створена відповідно до принципів [2], що дозволяє уникнути дублювання інформації та полегшує пошук необхідних даних. Крім цього, у моделі враховано можливість розширення функціоналу, наприклад, додавання звітів або модуля для обробки повернень.

Розроблений концепт передбачає також реалізацію авторизації для захисту доступу до адміністративних функцій. Завдяки розділенню ролей користувачів можна обмежити доступ лише для відповідальних осіб, що знижує ризик помилок або несанкціонованих змін. Усі запити до бази даних мають проходити через серверну логіку, написану на Python, що дозволяє забезпечити контроль та захист інформації. У підсумку, запропоноване рішення виглядає оптимальним для невеликого або середнього магазину, якому потрібна автоматизація основних бізнес-процесів без складної інфраструктури.

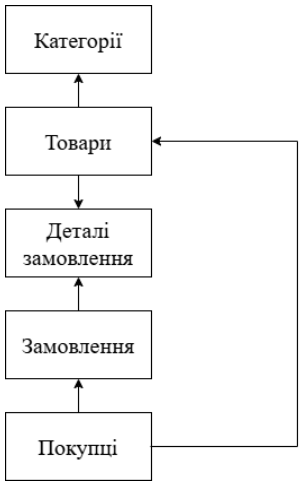


Рис. 1 – ER-діаграма

Висновки

У цій статті розглянуто підхід до побудови вебсайту для внутрішнього управління магазином інструментів. Було змодельовано структуру бази даних і спроєктовано основні

елементи інтерфейсу та адміністративної панелі. Запропонована концепція враховує потребу в простоті, швидкому доступі до інформації та можливості масштабування. Візуалізація у вигляді ER-діаграми допомогла краще зрозуміти логіку системи. Представлена модель не є завершеним продуктом, однак може бути базою для подальшої реалізації. Реалізація такого проєкту сприятиме оптимізації рутинних завдань і покращенню обліку товарів та клієнтів.

Список використаних джерел

1. Балик Н.Р., Мандзюк В.І. Бази даних MySQL. Навчальний посібник. Тернопіль: Навчальна книга – Богдан, 2008. 160 с.

2. Міхнова О.Д. Проєктування та розробка баз даних в MySQL Workbench: методичні вказівки. Харків: ДБТУ, 2024. 70 с.

Відомості про автора:

Рюмшина Марія Віталіївна – здобувачка вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, розробка вебсайтів, автоматизація бізнес процесів.

E-mail: 7328568@stud.kai.edu.ua

УДК 004.6

ПОРІВНЯННЯ ЕФЕКТИВНОСТІ РІЗНИХ ФОРМАТІВ ЗБЕРІГАННЯ ВЕЛИКИХ НАБОРІВ ДАНИХ: CSV, PARQUET, FEATHER

Володимир СИРОТЮК
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Яна Андріївна Бєлозьорова

У роботі представлено експериментальне порівняння ефективності трьох популярних форматів зберігання великих наборів даних: CSV, Parquet та Feather. Проведено аналіз за такими критеріями, як обсяг файлів після збереження, швидкість читання та запису, підтримка типів даних, а також зручність використання у процесах ETL та Data Science. Результати показали, що формат Parquet забезпечує найкраще стиснення та баланс між продуктивністю і масштабованістю, тоді як Feather вирізняється високою швидкістю обробки даних. Формат CSV, попри свою універсальність, поступається за ефективністю сучасним бінарним форматам. Отримані результати можуть бути корисними для вибору оптимального формату даних у практичних задачах аналітики та машинного навчання.

Ключові слова: *формати даних, великий обсяг даних, CSV, Parquet, Feather, ефективність, зберігання, продуктивність, ETL, машинне навчання.*

Вступ

З розвитком цифрових технологій та поширенням концепції «великих даних» (Big Data) постало питання ефективного зберігання та обробки інформації. Одним з ключових факторів у роботі з великими наборами даних є вибір формату їх зберігання, що впливає не лише на продуктивність, а й на витрати пам’яті, зручність обробки та масштабованість системи. У межах цього дослідження проведено порівняння трьох поширених форматів – CSV, Parquet та Feather, які активно застосовуються у сфері аналізу даних та машинного навчання.

Ціль роботи.

Метою дослідження є експериментальне порівняння ефективності форматів CSV, Parquet та Feather за наступними критеріями:

- обсяг отриманого файлу після збереження даних;
- час зчитування та запису;
- підтримка типів даних і сумісність із програмними інструментами;
- зручність використання в процесах ETL та Data Science.

Матеріали та методи.

У роботі використовувався синтетичний набір даних обсягом 1 мільйон записів із числовими, рядковими та датованими стовпцями. Експерименти проводилися в середовищі Jupyter Notebook з використанням бібліотек Python: pandas, pyarrow, fastparquet, feather.

Тестування включало:

- збереження датафрейму у трьох форматах;

- замір обсягу файлу (у мегабайтах);
- замір часу зчитування та запису (у секундах);
- перевірку коректності зчитаних даних;
- спроби обробки часткових даних (наприклад, тільки кількох стовпців).

Вимірювання часу проводилось з використанням функції time.perf_counter(). Всі експерименти виконувались на ноутбуці з процесором Intel Core i5, 16 GB RAM, SSD.

Результати та обговорення.

CSV є найпростішим форматом, але має ряд обмежень: відсутність метаданих, типізація відсутня, відносно великий розмір і низька продуктивність. Проте він універсальний і читається майже всіма інструментами.

Parquet – колонковий формат із підтримкою стиснення та типів даних. Забезпечує чудову компресію і особливо ефективний при вибірці окремих стовпців. Добре підходить для обробки даних у Spark, Hive, AWS, Google BigQuery.

Feather – формат, орієнтований на високу швидкість обміну даними між середовищами Python та R. Має вищу продуктивність зчитування, ніж Parquet, але створює трохи більші файли. Ідеальний для проміжної обробки у дата-аналітичних пайплайнах.

Таблиця 1

Формати збереження табличних даних: порівняння особливостей

Формат	Розмір файлу	Час зчитування	Час запису	Коментар
CSV	142 МБ	1.84 сек	2.14 сек	Простий, але повільний. Не підтримує типи.
Parquet	18.6 МБ	0.51 сек	0.64 сек	Стислий, підтримує схеми, ідеальний для колонкового доступу.
Feather	35.1 МБ	0.23 сек	0.35 сек	Дуже швидкий, але об’ємніший за Parquet. Зручний у Python-середовищі.

Висновки

Проведене дослідження підтвердило, що вибір формату зберігання великих наборів даних суттєво впливає як на продуктивність системи, так і на обсяг використовуваної пам’яті та зручність подальшої обробки.

Формат Parquet продемонстрував найкраще співвідношення між стисненням даних та швидкістю доступу. Завдяки колонковій структурі він дозволяє ефективно працювати з окремими стовпцями. Таким чином, його доцільно використовувати для довготривалого зберігання даних, особливо в системах, орієнтованих на масштабування, таких як хмарні рішення або Hadoop-орієнтовані платформи.

Формат Feather, натомість, показав найвищу швидкість читання та запису, хоча і менш ефективне стиснення порівняно з Parquet. Цей формат ідеально підходить для внутрішнього обміну даними між Python- або R-процесами, коли важливо швидко передавати дані між частинами аналітичного пайплайну.

Що стосується CSV, то попри свою універсальність і простоту, він має обмеження у швидкодії, займає більше місця на диску, не підтримує схеми типів і може створювати складнощі при роботі з великими обсягами інформації. Тому його доцільно використовувати переважно у простих сценаріях або для обміну з системами, які не підтримують сучасні формати.

У підсумку, оптимальний вибір формату залежить від конкретної задачі, середовища виконання, необхідності стискання, швидкості доступу до даних та сумісності з іншими інструментами. У практичних системах аналітики та обробки даних доцільно комбінувати різні формати, обираючи найкращий для кожного етапу обробки.

Список використаних джерел:

1. McKinney, W. (2017). *Python for Data Analysis*. O'Reilly Media.
2. Apache Parquet Documentation. https://parquet.apache.org/
3. Feather Format Documentation. https://arrow.apache.org/
4. Pandas Documentation. https://pandas.pydata.org/
5. Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters.

Відомості про автора:

Сиротюк Володимир Михайлович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, великий обсяг даних, машинне навчання.

E-mail: 7340508@stud.kai.edu.ua

УДК 004.45 : 004.451

ПРОЦЕСИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ: ЕВОЛЮЦІЯ, СУЧАСНІ ПІДХОДИ ТА ПРАКТИЧНЕ ВПРОВАДЖЕННЯ

Валентина СКАЛОВА
Старший викладач кафедри ІПЗ
Яна БЄЛОЗЬОРОВА
Доцент кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ

У статті розглянуто сучасні підходи до розробки програмного забезпечення, такі як Agile, CI/CD та DevOps. Описано еволюцію методологій – від гнучких фреймворків до технічних практик безперервної інтеграції та доставки, що забезпечують швидку, якісну ітеративну розробку. Основну увагу приділено парадигмі DevOps як логічному продовженню Agile та CI/CD, що охоплює повний цикл створення й експлуатації програмного забезпечення.

Ключові слова: *програмна інженерія, DORA-метрики, безперервна інтеграція, IT-розробка, масштабування.*

Вступ

Програмна інженерія як дисципліна виникла наприкінці 1960-х років у відповідь на «кризу програмного забезпечення» – ситуацію, коли великі програмні проєкти регулярно виходили за межі бюджету, зривали терміни та не відповідали потребам користувачів. З'ясувалося, що неформальні підходи типу «кодуй та виправляй» (code-and-fix) не масштабуються для складних систем, що призводило до незавершених або неякісних продуктів. В 1968 році на конференції НАТО було введено термін «software engineering» (програмна інженерія) як спробу застосувати інженерні принципи до розробки програмного забезпечення. Відтоді особлива увага приділяється розробці процесів – систематизованих методів і практик, що регламентують життєвий цикл програмного забезпечення.

Перші формальні моделі процесів з'явилися на початку 1970-х років. Вони передбачали лінійну послідовність фаз – від збору вимог до супроводження – де завершення кожної фази передує початку наступної. На початку 2000-х відбувся поворот до гнучкості: було проголошено Agile Manifesto, який поставив на перше місце людей і співпрацю замість громіздких процесів і документації. Agile-методології (наприклад, Scrum, Kanban) підкреслюють ітеративну розробку, часті випуски і постійний зворотний зв'язок. У 2010-х роках наступним кроком еволюції стало впровадження культури DevOps – тісної інтеграції розробки та IT-операцій. DevOps не є жорстким процесним стандартом, а радше інженерною культурою та набором практик, що об'єднують команди девелоперів і системних інженерів для прискорення випуску якісного програмного забезпечення. В основі DevOps – автоматизація, безперервна інтеграція та доставка (CI/CD), спільна відповідальність за продукт і швидкий зворотний зв'язок.

Роль процесів у життєвому циклі програмного забезпечення

Життєвий цикл програмного забезпечення (ПЗ) охоплює послідовність етапів від зародження задуму до припинення використання системи. Класичними фазами є: визначення вимог, проєктування, реалізація (програмування), тестування, впровадження і

супроводження (підтримка). Процеси програмної інженерії визначають, як саме виконуються ці етапи, які артефакти створюються та які практики застосовуються. Наявність чітко визначених процесів на кожній стадії життєвого циклу забезпечує передбачуваність і керованість розробки. Зокрема, процесний підхід дає можливість планувати і оцінювати проєкти, виявляти та мінімізувати ризики на ранніх стадіях, відстежувати прогрес, забезпечувати необхідну якість і відповідність продукту вимогам замовників.

Стандарти на кшталт ISO/IEC 12207 надають спільну рамку термінів і набору процесів життєвого циклу, що можуть застосовуватися організаціями для впорядкування розробки і підтримки ПЗ. Процеси охоплюють не лише безпосередньо розробку, але й допоміжні аспекти: управління проєктом, конфігураціями, забезпечення якості, верифікацію і валідацію, обслуговування, а також організаційні процеси (наприклад навчання персоналу, поліпшення процесів). На кожному етапі визначені входи (вимоги, проєктна документація, вихідний код тощо) та виходи (специфікації, тестові звіти, інсталяційні пакети, інше), а також відповідальні ролі. Така структурованість дозволяє великим командам і організаціям працювати скоординовано, навіть над дуже складними системами.

Наприклад, процес управління вимогами встановлює методи збору та погодження вимог із замовником, що знижує ймовірність нерозуміння цілей проєкту. Процес проєктування визначає підходи до архітектури системи, забезпечуючи її відповідність вимогам та можливість подальшого супроводження. Процеси тестування гарантують, що виявлення та усунення дефектів відбувається систематично. Структуровані процеси життєвого циклу виступають своєрідною “дорожньою картою” для команди розробки, де кожна фаза має свої цілі та артефакти, що слугують основою для наступної фази. Різні моделі SDLC по-різному організують ці фази: традиційні моделі виконують їх послідовно, тоді як гнучкі – можуть повторювати цикли кілька разів або виконувати деякі види робіт паралельно. У будь-якому випадку, роль процесів полягає в тому, щоб забезпечити контроль над складністю, підвищити якість програмного продукту та узгодити розробку із бізнес-цілями і очікуваннями зацікавлених сторін.

Сучасні підходи та практики: Agile, CI/CD, DevOps

У відповідь на виклики стрімкого розвитку технологій та бізнес-вимог, індустрія перейшла до гнучких методологій (Agile) і практик безперервної доставки. Agile-підходи (Scrum, Kanban, Extreme Programming тощо) кардинально переосмислили процес розробки на початку 2000-х, замінивши великі водоспадні проєкти на ітеративне створення продукту невеликими інкрементами. Agile-методології керуються Маніфестом Agile, який проголошує цінність “особистостей і взаємодії понад процеси та інструменти; працюючого ПЗ понад вичерпну документацію; співпраці з замовником понад формальне погодження контракту; реагування на зміни понад слідування початковому плану”. Таким чином, хоча процеси і інструменти не заперечуються, більший акцент ставиться на гнучкість та швидке реагування на зворотний зв’язок користувачів.

Технічною опорою Agile стали практики CI/CD (Continuous Integration/Continuous Delivery) – неперервної інтеграції та доставки коду. Continuous Integration (CI) означає регулярне автоматичне злиття змін коду в основну гілку репозиторію з автоматичним запуском тестів, щоб вчасно виявити інтеграційні проблеми. Continuous Delivery/Deployment (CD) – це підхід, за якого нові версії програм автоматизовано проходять всі етапи збірки, тестування і розгортання; при continuous delivery реліз готовий до встановлення у будь-який момент (але може потребувати ручного підтвердження), а при continuous deployment – оновлення розгортаються у виробниче середовище автоматично. Метою CI/CD є

максимальне прискорення і спрощення виходу змін у продакшн при збереженні високої якості, що фактично продовжує agile-принципи на рівень інфраструктури. За даними Red Hat, впровадження CI/CD допомагає знизити кількість багів, спростити оновлення, зменшити складність релізів і зробити випуски частішими та більш передбачуваними.

Найбільш актуальною парадигмою сьогодні є DevOps, яка стала логічним продовженням Agile і CI/CD, охопивши повний цикл від розробки до експлуатації. Термін DevOps (Development + Operations) позначає не стільки конкретний процес, скільки культуру співпраці та набір найкращих практик, що мають на меті усунути розрив між командами розробників і IT-операцій. Лен Басс та інші дають таке визначення: «DevOps – це набір практик, призначених для скорочення часу між внесенням зміни в систему та впровадженням цієї зміни в робоче середовище при одночасному забезпеченні високої якості». Основні принципи DevOps – спільна відповідальність, автоматизація робочих потоків і швидкий зворотний зв’язок. Практично DevOps реалізується через побудову конвеєрів безперервної інтеграції/доставки, інфраструктуру як код, розгортання в хмарі, контейнеризацію, моніторинг та інші інструменти, що дозволяють випускати оновлення в продакшн дуже часто і надійно.

Однією з ключових складових DevOps є впровадження метрик для оцінки ефективності процесу розробки та доставки. Дослідницька програма DORA (DevOps Research & Assessment) виділила чотири показники, що корелюють із успішністю IT-команд. Ці DORA-метрики включають: частоту деплоїв (як часто команда розгортає новий код), час виконання змін (lead time – від моменту комміту до моменту деплою в продакшн), відсоток невдалих змін (частка релізів, що призводять до інцидентів або відкоту) та час відновлення після збою (MTTR, середній час на відновлення сервісу після інциденту). Високі показники (часті деплої, короткий lead time, низький відсоток невдалих релізів і швидке відновлення) характеризують організації з розвиненими DevOps-процесами. За допомогою цих метрик компанії можуть кількісно оцінювати зрілість своїх процесів і фокусуватися на постійному поліпшенні слабких місць.

DevOps значно вплинув на IT-індустрію – успішне його впровадження дозволяє досягти швидших і надійніших релізів, тіснішої відповідності IT-сервісів бізнес-потребам та кращої стабільності роботи в продакшні. На відміну від традиційних підходів чи навіть «чистого» Agile, DevOps охоплює весь життєвий цикл ПЗ, включно з експлуатацією: моніторингом, швидким виправленням помилок на бою, масштабуванням. Це робить його однією з найактуальніших нині парадигм процесу розробки програмного забезпечення.

Приклад практичного проєкту

Розглянемо приклад практичного застосування процесів програмної інженерії на проєкті впровадження хмарного рішення з використанням DevOps-підходу. Припустимо, компанія розробляє вебсервіс, що працює у хмарі (наприклад, платформа електронної комерції). Проєктна команда вирішує використати гнучку методологію Scrum для організації розробки і практики DevOps для автоматизації розгортання. На початку команда спільно з бізнес-стейкхолдерами визначає беклог вимог, який пріоритезується перед кожним спринтом. Розробники, тестувальники і системні інженери працюють єдиною Scrum-командою. Кожні два тижні проводяться спринти, на яких команда реалізує певний набір користувацьких історій (функціональних вимог). В кінці спринту – демонстрація працюючого інкременту продукту у тестовому середовищі та збір зворотного зв’язку від замовника.

Паралельно налаштовано CI/CD-пайплайн. Код зберігається у спільному репозиторії (Git); при кожному коміті змін запускається процес Continuous Integration: система збірки (наприклад, Jenkins чи GitLab CI) автоматично компілює додаток, запускає модульні тести та збирає артефакти (контейнер Docker з додатком). Якщо збірка і тести успішні, конвеєр переходить до стадії Continuous Delivery: виконуються інтеграційні та end-to-end тести у середовищі, що імітує продуктивне. Інфраструктура для тестового і продуктивного середовища описана декларативно (Infrastructure as Code, наприклад Terraform або Ansible), тому її розгортання теж автоматизоване і повторюване. Після проходження всіх тестів нова версія автоматично розгортається у хмарному середовищі (наприклад, на кластері Kubernetes) – тобто реалізовано Continuous Deployment на продакшн з мінімальним втручанням людини. Впроваджено моніторинг (за допомогою інструментів на кшталт Prometheus, Grafana) та логування, тож команда одразу отримує алерти у разі будь-яких аномалій після релізу.

Такий процес дозволяє команді випускати оновлення дуже часто – фактично кожна підтверджена зміна може бути в продакшні вже за кілька годин. Для порівняння, до впровадження цих практик цикл релізу міг становити кілька тижнів чи місяців. Наприклад, у компанії Etsy перехід до DevOps-культури дозволив збільшити частоту розгортання з раз на тиждень до понад 50 разів на день. Інша відома історія успіху – Netflix, що завдяки хмарній архітектурі і автоматизації скоротив час деплою нових версій з кількох днів до кількох хвилин, забезпечивши при цьому майже 100% доступність свого сервісу для користувачів. У нашому гіпотетичному проєкті застосування Scrum + DevOps також дає змогу бізнесу швидко отримувати нові функції, а технічній команді – підтримувати високий рівень якості. Кожен спринт завершується робочим продуктом, який проходить через CI/CD-конвеєр у staging- і production-середовище. Якщо виявляється критична помилка, завдяки автоматизованим тестам і моніторингу її можна усунути та випустити виправлення того ж дня, з мінімальним впливом на користувачів. Такий практичний кейс демонструє, як процеси програмної інженерії, підкріплені сучасними інструментами, приводять проєкт до успіху – скорочуючи «time-to-market» (час виходу на ринок) і підвищуючи задоволеність користувачів якістю сервісу.

Проблеми та виклики впровадження процесів

Незважаючи на очевидні переваги, реалізація процесного підходу на практиці супроводжується низкою викликів. Людський фактор і культура організації часто є вирішальними. Співробітники можуть опиратися змінам – особливо якщо роками працювали за старими методами. Перехід від неформального стилю до суворішого процесу іноді сприймається як зайва бюрократія, а перехід від традиційної моделі до Agile/DevOps – як стрес через необхідність опановувати нові навички. Опір змінам – один із найпоширеніших бар’єрів: люди звикають до своїх підходів і не бажають виходити із зони комфорту. В організаціях з жорсткою ієрархічною культурою впровадження гнучких принципів і DevOps може наштовхнутися на нерозуміння: якщо раніше цінувалося слідування інструкціям, покарання за помилки і приховування проблем, то DevOps-культура натомість вимагає відкритості, експериментування і толерантності до невдач як до точки навчання. Зміна мислення персоналу і керівництва – тривалий процес, що потребує підтримки зверху (керівництва) і ініціативи знизу (ентузіастів змін).

Недостатнє розуміння або неправильне застосування процесів – інша часта проблема. Наприклад, компанія може оголосити перехід на Agile, але продовжувати працювати по суті водоспадно, просто розбивши проєкт на спринти без справжньої гнучкості (так званий

“Cargo Cult Agile”, коли атрибути процесу імітуються, але цінності не дотримуються). Або ж навпаки – спробувати впровадити повністю формальний процес (зразок ISO стандартів чи CMMI) у невеликій команді, де він створить більше паперової роботи, ніж реальної користі. Впровадження процесів має бути адаптоване під контекст: те, що ефективно для великої організації в оборонній галузі, може бути непосильним для стартапу, і навпаки.

Технологічні виклики також суттєві, особливо при впровадженні DevOps. Необхідно підібрати і освоїти багато інструментів: системи керування вихідним кодом, платформи CI/CD, платформи контейнеризації (Docker), оркестрації (Kubernetes), хмарні сервіси, системи моніторингу тощо. Інтеграція цих компонентів в єдиний конвеєр – нетривіальне завдання. Команда може зіткнутися з браком експертизи: DevOps вимагає інженерів широкого профілю, які розуміють і розробку, і адміністрування систем, і тестування, і безпеку. Навчання персоналу та залучення досвідчених спеціалістів – обов’язкові умови, що можуть потребувати часу та інвестицій.

Організаційні бар’єри включають також структуру підрозділів і розподіл відповідальності. У традиційних компаніях відділи розробки, тестування, експлуатації історично відокремлені. Перебудувати їх у міжфункціональні команди нелегко – виникають питання, хто за що відповідає, як зміниться управління людьми. Без підтримки керівництва DevOps ініціативи часто застопорюються. Ще один виклик – збереження балансу між гнучкістю та дисципліною. Agile закликає мінімізувати зайву документацію, але в регульованих сферах (банківський софт чи авіація) існують вимоги стандартів і відповідності, які ніхто не скасовував. Тому команди мусять знаходити компроміс: впроваджувати гнучкі підходи, одночасно дотримуючись ключових формальних вимог.

Нарешті, впровадження нових процесів може спочатку знижувати продуктивність команди, поки люди навчаються і налагоджують нові практики. Це тимчасове «просідання» потрібно враховувати і не вимагати миттєвого зростання швидкості розробки. Розумне керівництво поступово впроваджує зміни – наприклад, починає з пілотного проєкту, де обкатує новий процес, вчиться на помилках, а вже потім масштабує успішний досвід на всю організацію. Регулярні ретроспективи і опитування команди допомагають виявити, що працює, а що ні, і відповідно скоригувати процеси.

Висновки

Процеси програмної інженерії відіграють ключову роль у забезпеченні успішного життєвого циклу програмного забезпечення. Історично доведено, що хаотична розробка без процесів масштабовано не працює – великі проєкти вимагають планування, координації та контролю. Водночас надмірно жорсткі процеси можуть стримувати інновації та швидкість. Еволюція від водоспаду до Agile і DevOps демонструє прагнення індустрії знайти оптимальний баланс між дисципліною і гнучкістю. Сучасні найкращі практики радять використовувати гібридні підходи, адаптовані до контексту конкретного проєкту і організації.

Керівникам IT-проєктів можна рекомендувати інвестувати в розвиток процесів поступово: спочатку визначити “больові точки” (наприклад, часті дефекти на продакшні чи повільний випуск нових версій) і впровадити практики, що їх адресують – можливо, це буде налагодження тестування і CI/CD, або навчання команди гнучкому плануванню. Важливо залучати команду до вибору та налаштування процесів, щоби була згода і розуміння навіщо ті чи інші практики потрібні. Не менш важливо встановити метрики успіху і відстежувати їх динаміку – тут стануть в пригоді згадані DORA-метрики для оцінки швидкості та надійності випусків, а також показники якості (щільність дефектів, задоволеність користувачів).

Постійний моніторинг дозволяє перевести впровадження процесів у площину data-driven (керовану даними) – приймати рішення про зміни на основі фактів і вимірів, а не припущень.

Для сталого успіху організація має культивувати культуру безперервного вдосконалення. Це відповідає найвищому рівню зрілості в моделях типу CMMI, де процеси не тільки визначені і виміряні, а й постійно оптимізуються на основі зворотного зв'язку. Ретроспективи в Agile, постмортеми в DevOps (аналіз інцидентів без пошуку винних) – дієві інструменти, що дозволяють виявляти точки росту і коригувати процеси. Також рекомендується обмінюватися досвідом з індустрією: вивчати звіти і дослідження (напр. Accelerate: State of DevOps Reports від DORA), впроваджувати стандарти ISO/IEC, які підходять до вашої сфери, та проходити сертифікації на зрілість процесів, якщо це додає цінності для бізнесу.

Список використаних джерел

1. IBM Think Blog. What is the SDLC? (2024) – визначення та переваги життєвого циклу розробки ПЗ.
2. Agile Manifesto (2001). Manifesto for Agile Software Development – чотири ключові цінності Agile.
3. Red Hat. What is CI/CD? (2023) – визначення неперервної інтеграції/доставки та їх роль у прискоренні життєвого циклу ПЗ.
4. Atlassian. DORA metrics: How to measure DevOps success – опис чотирьох ключових метрик DORA для оцінки ефективності DevOps.

Відомості про авторів:

Скалова Валентина Анатоліївна – старший викладач кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технології Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, якість програмного забезпечення, супроводження, розгортання.

E-mail: valentyna.skalova@npp.kai.edu.ua

Бєлозьорова Яна Андріївна – доцент, к.т.н. кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технології Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, штучний інтелект, розпізнавання мови, вейвлет аналіз.

E-mail: yana.bielozorova@npp.kai.edu.ua

UDC 004.77

COMPARING MONITORING ARCHITECTURES FOR KUBERNETES NETWORKS: THEORETICAL FOUNDATIONS OF EBPF'S RESOURCE EFFICIENCY

Ihor SKOSTARIEV
PhD student of the Software Engeneering department
State university «Kyiv aviation institute», Kyiv
Scientific supervisor: Phd Olena Grinenko

This paper examines the theoretical underpinnings of utilizing extended Berkeley Packet Filter (eBPF) technology for network traffic analysis within containerized Kubernetes environments. The analysis focuses on the node-centric deployment model of eBPF, which reduces operational complexity and performance overhead compared to traditional per-pod monitoring approaches, while addressing critical challenges in cross-node visibility and integration with existing Kubernetes networking components.

Keywords: *Kubernetes, eBPF, containerized environments, network traffic analysis, node-centric monitoring, Linux kernel, deployment simplicity.*

Architectures comparison

Modern containerized environments based on Kubernetes present unique network traffic analysis challenges. The ephemeral nature of containers, complex east-west traffic patterns, and the isolation mechanisms inherent to container technology create significant blind spots in conventional monitoring approaches.

Extended Berkeley Packet Filter (eBPF) technology represents a paradigm shift in network traffic analysis by enabling programmable, high-performance packet inspection directly within the Linux kernel. The key architectural advantage of eBPF in Kubernetes environments is its node-centric deployment model - a single eBPF program deployed at the node level can monitor all network traffic for all pods running on that node, eliminating the need for per-pod agents or sidecars.

This node-centric approach provides several critical advantages:

1. Deployment simplicity – eBPF programs need only be deployed once per node rather than once per pod, reducing operational complexity in clusters with hundreds or thousands of pods.
2. Resource efficiency – a single monitoring instance consumes less memory and CPU than equivalent per-pod solutions.
3. Internal traffic visibility – eBPF can observe internal cross-pod communications within the same node.

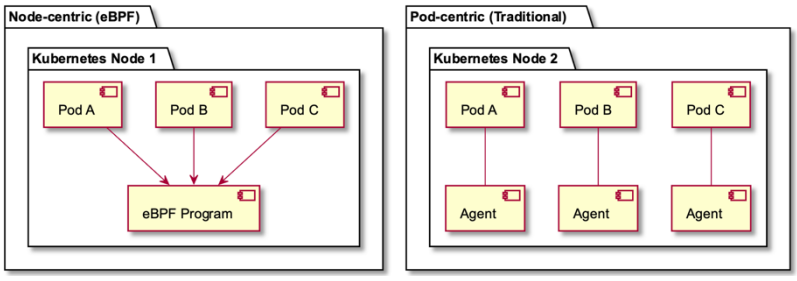


Fig. 1. Node-centric vs Pod-centric monitoring architecture

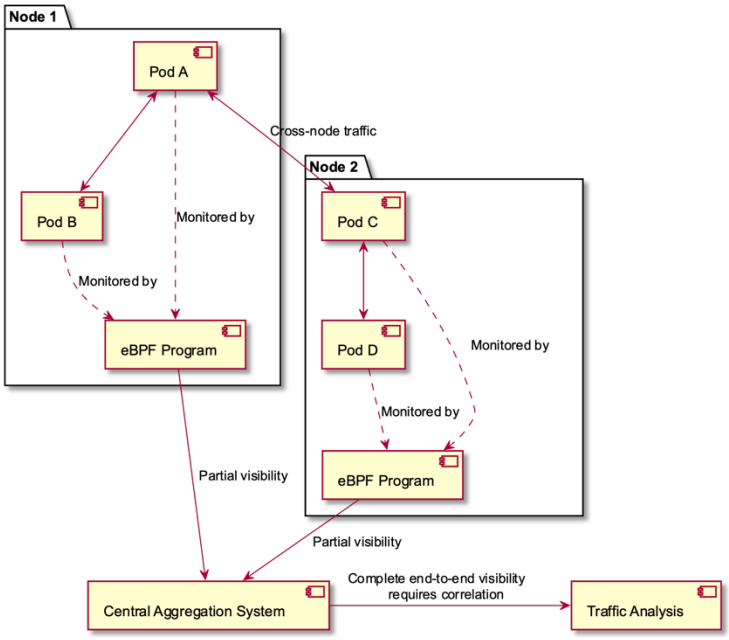


Fig. 2. Cross-node traffic monitoring challenges with eBPF

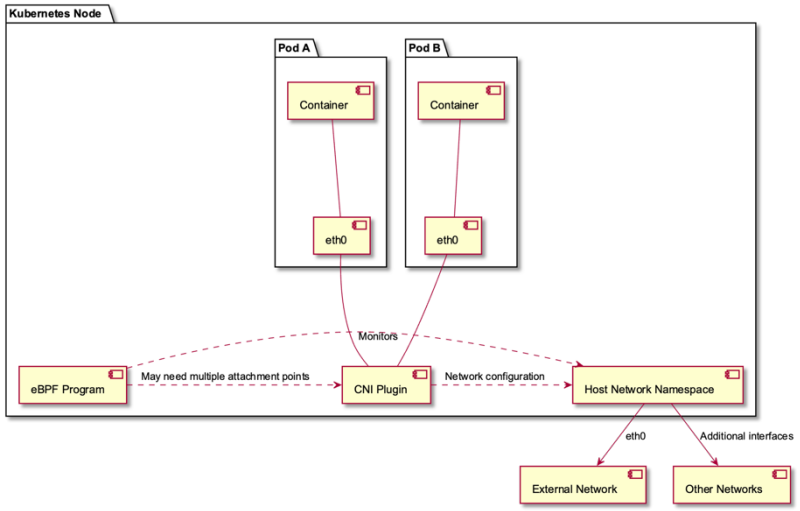


Fig. 3. Integration with CNI plugins and multi-interface monitoring

Monitoring Approach	Pod-to-Pod Internal Traffic	Pod-to-Service Traffic	Cross-Node Pod Traffic	Application Layer Context	Deployment Complexity	Encryption Handling
eBPF (Node-centric)	Complete (within node)	Complete (within node)	Partial (requiresaggregation)	Limited (packet-level)	Low (per node)	Limited to unencrypted traffic
Service Mesh Proxy	Partial (proxied traffic only)	Complete	Complete (with proxy at both ends)	Rich (L7, headers, metrics)	High (per pod)	Can terminate TLS
CNI-integrated eBPF	Complete	Complete	Complete	Limited (packet-level)	Medium	Limited to unencrypted traffic
Host-level tcpdump	Raw packets only (no context)	Raw packets only	Raw packets only	None	Medium	Limited to unencrypted traffic

The theoretical analysis demonstrates that eBPF-based traffic analysis provides efficiency benefits with certain limitations. For a typical Kubernetes cluster, traditional per-pod monitoring approaches require agents in every pod, increasing resource consumption linearly with pod count. In contrast, eBPF's node-centric approach scales with node count, which is typically an order of magnitude smaller.

The refined mathematical model for resource efficiency, accounting for aggregation overhead, can be expressed as:

$$E = \frac{R_p \times P}{(R_n \times N) + R_a}$$

where E represents efficiency gain, R_p is per-pod resource consumption, P is number of pods, R_n is per-node resource consumption, N is number of nodes, and R_a is the resource overhead for cross-node aggregation and correlation. This model acknowledges that real-world deployments incur additional costs for coordinating data across nodes.

Several critical limitations must be considered when implementing eBPF-based monitoring:

1. Cross-node traffic visibility requires additional coordination mechanisms
2. eBPF programs cannot natively inspect encrypted traffic (e.g., TLS)
3. Kernel-level operation introduces potential security concerns if eBPF programs are not properly validated
4. Complex network configurations with multiple interfaces or network namespaces require careful eBPF attachment point planning

Integration with Container Network Interface (CNI) plugins presents both opportunities and challenges. CNI plugins like Cilium already leverage eBPF extensively, potentially offering more integrated monitoring capabilities but introducing dependencies on specific networking implementations. The optimal approach may involve leveraging CNI-provided eBPF programs rather than implementing separate monitoring solutions.

The theoretical foundations established in this analysis demonstrate that eBPF's node-centric monitoring approach provides significant advantages for network traffic analysis in Kubernetes environments, particularly for intra-node communication. By deploying analysis capabilities at the node level rather than per pod, eBPF reduces operational complexity and resource overhead. However, comprehensive monitoring solutions must address cross-node visibility limitations through appropriate aggregation mechanisms, handle encrypted traffic through complementary approaches, and carefully integrate with existing Kubernetes networking components. Future research should focus on addressing these limitations while maintaining eBPF's performance and efficiency advantages.

References

1. Mohammadreza Rezvani, Ali Jahanshahi, Daniel Wong, "Characterizing In-Kernel Observability of Latency-Sensitive Request-Level Metrics with eBPF", *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp.24-35, 2024.

2. Minato Sakuraba, Junichi Kawasaki, Takuya Miyasaka, Atsushi Tagami, "An Anomaly Detection Approach by AIML in IP Networks with eBPF-Based Observability", *2023 24st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp.171-176, 2023.

3. Simone Magnani, Fulvio Risso, Domenico Siracusa, "A Control Plane Enabling Automated and Fully Adaptive Network Traffic Monitoring With eBPF", *IEEE Access*, vol.10, pp.90778-90791, 2022.

4. Wenqi Pan, Yuedong Xu, Chenhao Wang, Jun Wu, "An eBPF-empowered Congestion Control System with Delay Requirements", *2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp.3875-3880, 2024.

5. Amin Sadiq, Hassan Jamil Syed, Asad Ahmed Ansari, Ashraf Osman Ibrahim, Manar Alohaly, Muna Elsadig, "Detection of Denial of Service Attack in Cloud Based Kubernetes Using eBPF", *Applied Sciences*, vol.13, no.8, pp.4700, 2023.

Information about the author:

Ihor Skostariev - postgraduate student at the Software Engineering Department of State university «Kyiv aviation institute». Interests: kubernetes, eBPF, network traffic analysis.

E-mail: 9016404@stud.kai.edu.ua

УДК 004.852:004.415

ПРОГНОЗУВАННЯ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ ЗА ДОПОМОГОЮ ГРАФОВИХ НЕЙРОННИХ МЕРЕЖ

Артем СОБКО
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Яна Андріївна Бєлозьорова

Стаття присвячена аналізу застосування графових нейронних мереж (GNN) для прогнозування дефектів у програмному забезпеченні. Розглянуто переваги використання синтаксичних графів, таких як AST і CFG, для глибшого аналізу коду. Оцінено ефективність GNN-моделей порівняно з традиційними методами та визначено основні виклики їх впровадження.

Ключові слова: *графові нейронні мережі, прогнозування дефектів, AST, CFG, аналіз коду, інтерпретованість, CI/CD.*

Вступ

Сучасні програмні системи відзначаються високою архітектурною складністю, розподіленим характером розробки та підвищеними вимогами до якості. За таких умов прогнозування дефектів стає одним із ключових чинників забезпечення надійності програмного забезпечення. Традиційні підходи до аналізу дефектів – зокрема статистичні моделі чи алгоритми машинного навчання на основі табличних даних – здебільшого ігнорують структурні й семантичні особливості програмного коду. Це суттєво обмежує їхню ефективність у складних програмних системах.

Графові нейронні мережі (англ. GNN – Graph neural networks) дають змогу моделювати взаємозв'язки між елементами даних у вигляді графів, що відкриває нові можливості для глибокого аналізу коду з урахуванням його синтаксичної структури, залежностей між компонентами та контексту виконання [1]. Використання GNN у задачах прогнозування дефектів дозволяє точніше виявляти потенційно проблемні фрагменти коду, особливо в системах зі складною модульною структурою – зокрема в мікросервісній архітектурі та розподілених застосунках [2].

Разом із впровадженням GNN у процес розробки постають виклики, пов'язані з інтерпретацією результатів прогнозування, масштабуванням моделей для великих кодових баз та їх адаптацією до специфіки предметної області. У зв'язку з цим, розробка спеціалізованих GNN-моделей для прогнозування дефектів набуває особливої актуальності та здатна істотно змінити підходи до забезпечення якості програмного забезпечення, зменшуючи витрати на тестування й супровід.

Ціль роботи

Метою даного дослідження є аналіз і систематизація підходів до прогнозування дефектів програмного забезпечення із застосуванням графових нейронних мереж (GNN), з акцентом на їхню здатність враховувати структурні та семантичні особливості коду для підвищення точності виявлення потенційних помилок.

У межах дослідження поставлено наступні завдання:

- здійснити огляд і класифікацію сучасних методів прогнозування дефектів, зокрема моделей машинного навчання та GNN-орієнтованих підходів;
- проаналізувати використання синтаксичних графів, таких як абстрактні синтаксичні дерева (англ. AST – Abstract syntax tree) та графи потоків управління (англ. CFG – Control-flow graph), у GNN-моделях для аналізу програмного коду;
- оцінити ефективність прогнозування за допомогою таких метрик, як точність, повнота та інтерпретованість;
- визначити переваги й недоліки GNN у задачах прогнозування дефектів;
- сформулювати практичні рекомендації щодо інтеграції GNN у процеси забезпечення якості ПЗ – зокрема у статичний аналіз коду та CI/CD-конвеєри (англ. Continuous Integration / Continuous Delivery) [3, 4];
- дослідити стратегії підвищення інтерпретованості результатів GNN-прогнозів, що сприятиме їхньому ефективному використанню в інженерних процесах [5].

Матеріали та методи

Методологічною основою цього дослідження є систематичний огляд наукових джерел і порівняльний аналіз ефективності графових нейронних мереж (GNN) у прогнозуванні помилок у програмному забезпеченні. Основу дослідження склали рецензовані публікації за 2022-2024 роки, відібрані з провідних журналів і матеріалів конференцій з програмної інженерії та штучного інтелекту. Актуальні роботи були знайдені в академічних базах даних за такими ключовими словами: «графові нейронні мережі», «прогнозування дефектів програмного забезпечення», «виявлення вразливостей у коді».

Методика дослідження передбачала:

1. Класифікацію підходів до прогнозування помилок на основі GNN за типом графового представлення коду (наприклад, абстрактні синтаксичні дерева, графи потоків управління, багаторівневі графи залежностей);
2. Аналіз переваг і недоліків методів – із фокусом на масштабованість, інтерпретованість моделей та якість вхідних даних;
3. Розробку рекомендацій щодо інтеграції GNN у процеси забезпечення якості ПЗ, зокрема в інструменти статичного аналізу та CI/CD-конвеєри [4, 5].

Результати та обговорення

У межах цього систематичного огляду проаналізовано трансформаційний потенціал графових нейронних мереж (GNN) у прогнозуванні дефектів програмного забезпечення. Дослідження підтверджують суттєві переваги GNN-підходів, поряд із певними недоліками, які потребують подальшого вивчення.

Огляд сучасних наукових публікацій свідчить, що моделі на основі GNN, які використовують синтаксичні графи – зокрема абстрактні синтаксичні дерева (AST) та графи потоків управління (CFG) – стабільно демонструють вищу ефективність порівняно з традиційними методами машинного навчання. Наприклад, такі моделі, як CGCN та GCNNN, досягають точності прогнозування в межах 85-90%, і показника F1-оцінки у діапазоні 80-85% на відкритих наборах даних. Це на 10-20% перевищує результати моделей на основі лише AST або логістичної регресії, особливо в контексті складних систем, як-от мікросервісні архітектури [3, 4].

Максимальна ефективність спостерігаються при використанні GNN на ранніх етапах аналізу коду – у таких випадках виявлення дефектів пришвидшується на 30-40% порівняно з

традиційним ручним статичним аналізом. Це особливо актуально для виявлення логічних помилок і вразливостей типу «ін'єкцій» [5].

Разом із очевидними перевагами, ефективність графових нейронних мереж (GNN) значною мірою залежить від типу виявлюваного дефекту. Так, для стандартних шаблонів програмування та поширених вразливостей (наприклад, нульових посилань на вказівники) точність GNN-моделей досягає 90%. Однак для контекстно-залежних або доменно-специфічних дефектів, що потребують глибокого семантичного аналізу, цей показник знижується до 45-55% [3]. Особливою перевагою GNN є здатність зменшувати кількість хибно-позитивних спрацювань, що, за результатами досліджень, покращує ефективність перегляду коду приблизно на 25% порівняно з класичними методами. Водночас існує проблема «сліпих зон»: певні дефекти залишаються невиявленими через обмеження статичного аналізу, і в критичних випадках це може призвести до непередбачуваної поведінки системи [4]. Подальший аналіз питань безпеки виявляє ще одну проблему – ризик відтворення вразливостей, обумовлених упередженістю в навчальних даних. Зокрема, це стосується перевірки прав доступу та валідації вхідних даних, що потребує участі експертів для гарантування надійності [5].

Таблиця 1

Порівняння продуктивності моделей GNN

Модель (Model)	Точність (Precision)	Відкликання (Recall)	F1-Оцінка (F1-Score)
CGCN	0.83-0.88	0.80-0.85	0.81-0.85
TextCNN	0.76-0.80	0.74-0.78	0.75-0.78
BiLSTM	0.72-0.77	0.70-0.75	0.71-0.76

Джерело: власна розробка автора на основі [3]

Значний прорив демонструє модель DeMuVGN, яка інтегрує кілька типів залежностей – між даними, викликами та розробниками. Такий підхід дозволяє досягти покращення метрики F1 до 17,4% (45,8%) у внутрішньопроєктних і до 17,9% (41,0%) у міжпроєктних сценаріях [5]. Крім того, контекстне виявлення дефектів, реалізоване через мультиграфові підходи, дає змогу скоротити обсяги ручного тестування на 20-30% при інтеграції в CI/CD-конвеєри [3]. Попри це, проблеми масштабованості залишаються актуальними: для великих проєктів приріст ефективності становить лише 10-15% через високу обчислювальну складність [4]. Іншою істотною перешкодою є недостатня інтерпретованість моделей – відсутність прозорості в ухваленні рішень обмежує використання GNN у безпеково критичних сферах [6].

Цікаво, що аналогічні GNN-рішення, застосовані в інших галузях, зокрема у прогнозуванні дефектів у напівпровідниках, демонструють точність до 98% при передбаченні дефектів утворення енергії. Це дає підстави для гіпотези про ефективність гібридних підходів, які поєднують GNN із системами на основі правил, підвищуючи точність і надійність [6]. Запропонована концептуальна модель передбачає інтеграцію GNN з класичними методами статичного аналізу, з використанням графічної візуалізації результатів для покращення інтерпретації. Такий гібридний підхід може підвищити точність виявлення дефектів на 15-20% у середньомасштабних проєктах, а також частково подолати обмеження, пов'язані з масштабуванням та якістю даних [5]. Загалом проведений огляд підтверджує

високу адаптивність GNN у різних доменах програмного забезпечення. Проте досягнення максимальної продуктивності можливе лише за умови використання різноманітних і якісних даних та глибокого налаштування моделей.

Висновки

Результати проведеного дослідження демонструють, що застосування графових нейронних мереж (GNN) у завданнях прогнозування дефектів програмного забезпечення перебуває на етапі активного розвитку. На сьогодні ці моделі переходять від експериментальних прототипів до більш стабільних рішень, здатних забезпечити суттєве покращення якості програмних продуктів у промислових умовах. Проведена систематизація наявних підходів дозволила окреслити основні напрямки, в яких використання GNN має найбільший потенціал. Зокрема, йдеться про контекстно-чутливе виявлення дефектів із використанням синтаксичних структур коду (таких як абстрактні синтаксичні дерева або графи потоку управління), раннє розпізнавання вразливостей під час аналізу програмного коду, а також інтеграцію GNN-моделей у процеси безперервної інтеграції та доставки (CI/CD) для забезпечення постійного контролю якості.

Водночас дослідження підтвердило наявність низки викликів, що стримують масштабне впровадження таких моделей у критично важливих сферах. Основними серед них є висока залежність від якості та різноманіття навчальних даних, обмеження масштабованості GNN при роботі з великими кодовими базами, а також недостатня інтерпретованість модельних прогнозів, що ускладнює їх прийняття в контексті безпеки та відповідальності. Ці аспекти вимагають подальших досліджень і технічних вдосконалень, зокрема у напрямі створення пояснюваних моделей штучного інтелекту, що забезпечать прозорість та зрозумілість результатів, а також у розробці гібридних систем, які поєднують GNN із класичними методами статичного аналізу на основі правил.

Окрему увагу заслуговує перспектива створення спеціалізованих моделей GNN для окремих доменів, таких як вбудовані або фінансові системи, де вимоги до точності та надійності особливо високі. Таким чином, хоча впровадження GNN у сферу забезпечення якості програмного забезпечення супроводжується певними викликами, наявні тенденції розвитку дозволяють очікувати суттєве покращення точності, надійності та практичності цих моделей у найближчому майбутньому.

Список використаних джерел

1. Li, Z., Wu, D., Liu, Y., & Wang, Z. (2021). Graph Neural Network-Based Software Defect Prediction with Abstract Syntax Trees. IEEE Transactions on Software Engineering, 47(12), 2893–2907.

2. Allamanis, M., Brockschmidt, M., & Khademi, M. (2018). Learning to Represent Programs with Graphs. International Conference on Learning Representations (ICLR).

3. Zhou, C., He, P., Zeng, C., & Ma, J. (2022). Software defect prediction with semantic and structural information of codes based on Graph Neural Networks. Information and Software Technology, 152, 107057.

4. Šikić, L., Kurdija, A. S., Vladimir, K., & Šilić, M. (2022). Graph Neural Network for Source Code Defect Prediction. IEEE Access, 10, 10402–10415.

5. Chen, J., Li, X., & Zhang, Y. (2024). DeMuVGN: Effective Software Defect Prediction Model by Learning Multi-view Software Dependency via Graph Neural Networks. In Proceedings of the 2024 International Conference on Software Engineering.

6. Rahman, M. H., Gollapalli, P., Manganaris, P., Yadav, S. K., Pilania, G., DeCost, B., Choudhary, K., & Mannodi-Kanakkithodi, A. (2023). Accelerating Defect Predictions in Semiconductors Using Graph Neural Networks. arXiv preprint arXiv:2309.06423.

Відомості про автора:

Собко Артем Юрійович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, графові нейронні мережі, прогнозування дефектів.

E-mail: artemmmm.sobko@gmail.com

УДК 004.415.2.045(076.5)

ВПЛИВ CLOUD-AGNOSTIC РОЗРОБКИ НА СТРАТЕГІЮ УПРАВЛІННЯ РИЗИКАМИ У ЖИТТЄВОМУ ЦИКЛІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Олена ЧЕБАНЮК
Професор кафедри ІПЗ
Антон СТАДНИЧЕНКО
Аспірант 3-го року навчання кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ

У доповіді розглядається проблема залежності від одного постачальника хмарних послуг (vendor lock-in) та можливості її мінімізації за допомогою cloud-agnostic підходу до розробки. Показано, як використання уніфікованих технологій, таких як Kubernetes та PostgreSQL, дозволяє підвищити гнучкість і стійкість програмних рішень. Проаналізовано переваги та недоліки такого підходу, зокрема вплив на складність архітектури й продуктивність. Зроблено висновок про доцільність впровадження cloud-agnostic принципів у критично важливих IT-системах.

Ключові слова: *cloud-agnostic, vendor lock-in, Kubernetes, хмарні сервіси, гнучкість, IT-архітектура.*

Постановка проблеми

Останніми роками є тенденція значного зростання використання хмарних сервісів. Все частіше відбувається міграція програмного забезпечення компаній у хмару через такі переваги, як масштабованість та економія витрат. У той же час з’являється ризик занадто сильної залежності від одного конкретного постачальника хмарних послуг (vendor lock-in) [1]. Це може обмежувати політику компанії щодо використання свого програмного забезпечення та призводить до виникнення додаткових ризиків у майбутньому. Постановка проблеми: як уникати подібних ризиків та зберігати свободу вибору постачальників та технологічних рішень, одночасно зберігаючи й переваги хмари. При цьому потрібно мінімізувати ризики, що виникають через прив’язку до одного постачальника хмарних послуг. Одним з підходів являється використання Cloud-agnostic розробки, що полягає у розробці програмного забезпечення з мінімальною залежністю від конкретної хмарної платформи.

Результати та обговорення

Cloud-agnostic підхід передбачає проєктування та розробку програмного забезпечення таким чином, щоб без суттєвих змін у коді чи архітектурі його можна було запускати на різних хмарних платформах. Такий підхід дозволяє уникнути залежності від сервісів та API конкретного провайдера, що доступні лише у нього, натомість покладаючись на універсальні стандартизовані засоби, що можуть бути використані на різних платформах [2]. Результатом є агностичний до хмари продукт – його можна розгорнути у хмарному середовищі іншого постачальника хмарних послуг. Використання Cloud-agnostic підходу суттєво змінює профіль ризиків у моделі життєвого циклу розробки ПЗ. Знижується ризик залежності від одного постачальника: при зміні цінової політики, рівня надання послуг або виникнення технічних збоїв. програмне забезпечення може бути перенесено до іншого постачальника послуг або розгорнуто у гібридному режимі. Така гнучкість у розгортанні програмних

засобів гарантує безперервність роботи сервісів в межах однієї платформи навіть за несприятливих умов, що значно підвищує стійкість бізнесу.

Проте, такий підхід має і недоліки. А саме: використання стандартних засобів заміть спеціалізованих від конкретного провайдера підвищують складність архітектури та вимоги до компетенції розробників. А додаткові абстракції, що потрібні для сумісності з різними середовищами, можуть негативно вплинути на ризик виникнення помилок під час реалізації та налаштування конфігурації системи. Також може бути втрачена частина продуктивності, що може бути отримана від використання конкретної платформи (cloud native). Адже універсальна база даних чи черга повідомлень зазвичай не такі оптимальні, як рішення надані провайдером і тісно інтегровані з його інфраструктурою. Це призводить до компромісу: організація вибирає між гнучкістю системи і зменшенням ризиків через залежність до одного постачальника та витрачанням більших ресурсів на розробку і можливим незначним зниженням продуктивності.

Розглянемо спрощений сценарій cloud-agnostic рішення. Припустимо, що організацією розробляється вебзастосунок, що буде розгортатися у хмарі. У якості основи для розгортання був обраний Kubernetes замість пропрієтарних сервісів одного провайдера. Він являє собою уніфікований шар оркестрації контейнерів та підтримується усіма основними хмарними постачальниками і навіть приватними хмарними [3]. Для розгортання компонентів застосунку за однаковим сценарієм у різних середовищах вони упаковуються у докер контейнери. У якості бази даних обирається СУБД з відкритим інтерфейсом (наприклад PostgreSQL). Її можна розгорнути на будь-якій інфраструктурі, або навіть замінити еквівалентним рішенням від постачальника з мінімальними змінами у коді. Цей підхід гарантує, що під час виникнення технічних проблем або зростання вартості у постачальника, застосунок зможе продовжувати працювати завдяки перенесенню на іншу хмару, або шляхом розподілу навантаження між кількома постачальниками. Це приводить до можливості зниження технологічних ризиків при використанні cloud-agnostic розробки.

Висновки

Враховуючи зростаючу залежність бізнесу від хмарних технологій – ризики, що пов’язані з прив’язкою до одного конкретного постачальна також повинні бути враховані у стратегії управління розробкою. Представлений приклад демонструє, що cloud-agnostic розробка є дієвим інструментом для підвищення гнучкості та надійності політики підтримки програмного забезпечення. Завдяки уникненню vendor lock-in компанії отримують стратегічну перевагу – можливість адаптації технологічних рішень до змін ринкових або технічних умов без істотних втрат. Хоча й на впровадження cloud-agnostic підходу на етапах розробки і розгортання потрібні додаткові зусилля і ресурси, для систем з високими вимогами до довговічності та гнучкості ці витрати виправдані. Отже, під час планування архітектури критично важливих продуктів та сервісів як основу стратегії керування ризиками доцільно закладати принципи cloud-agnostic для мінімізації залежності від окремих технологічних рішень.

Список використаних джерел

1. Gartner (2023). "Managing Vendor Lock-In Risks in Public Cloud IaaS and PaaS". URL: https://www.gartner.com/en/documents/4264599

2. Kratzke, N., & Quint, P.-C. (2017). "Understanding Cloud-Native Applications after 10 Years of Cloud Computing - A Systematic Mapping Study". Journal of Systems and Software, 126, 1-16. DOI: 10.1016/j.jss.2017.01.001

3. Cloud Native Computing Foundation (CNCF) (2020). "CNCF Cloud Native Survey 2020". URL: https://www.cncf.io/reports/cloud-native-survey-2020/

Відомості про авторів:

Олена Вікторівна Чебанюк – професор кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технології Державного університету «Київський авіаційний інститут». *Наукові інтереси*: програмна архітектура, мобільна розробка, , MDA, MDD.

E-mail: olena.chebaniuk@npp.kai.edu.ua

Антон Вячеславович Стадниченко – аспірант 3-го року навчання кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технології Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, хмарні сервіси, IT-архітектура.

E-mail: 4563053@stud.kai.edu.ua

UDC 004.4

KNOWLEDGE DISTILLATION: GENERATIONAL DECLINE IN ACCURACY

Oleksandr SUKHOVYI
PhD student of the Software Engeneering department
State university «Kyiv aviation institute», Kyiv
Scientific supervisor: Phd Olena Grinenko

This paper investigates the effects of multi-generational knowledge distillation, where each student model becomes the teacher for the next generation. Using the MNIST dataset, the study analyzes how classification accuracy changes when models are trained solely on soft labels from their predecessors. Results show an initial drop in performance followed by stabilization, indicating a phenomenon of generational loss. The findings highlight both the limitations and robustness of iterative knowledge transfer without ground truth reintroduction.

Keywords: *Knowledge distillation, model compression, generational loss, neural networks, soft labels, MNIST, deep learning, iterative learning, student-teacher model, accuracy degradation.*

Introduction

Knowledge distillation (KD) has become a foundational technique in deep learning for model compression. It was introduced to learn a model from ensemble of networks but currently it is often used for model compression [1]. It involves training a smaller student model to replicate the behavior of a larger, pre-trained teacher model, by learning from the soft outputs (probability distributions) produced by the teacher rather than directly from hard class labels. This approach not only allows the student to generalize better but also benefits from the implicit knowledge encoded in the teacher's softened predictions, such as inter-class similarities.

While KD is well-understood in single-generation contexts, significantly less attention has been paid to the effects of multi-generational distillation, in which each new student becomes the teacher for the next generation. We want to see if the knowledge transferred from one generation to the next retains its fidelity, or does performance deteriorate due to information loss across distillation steps?

Purpose of the research

The primary objective of this research is to examine the impact of repeated knowledge distillation across multiple generations of student models. Specifically, we aim to quantify the degree of accuracy loss that accumulates when models are no longer trained on ground truth labels but exclusively on the soft targets generated by their predecessor. Our study investigates whether the distillation process introduces compounding error across generations, potentially undermining the effectiveness of the original model's knowledge.

Materials and methods

All experiments are performed on the MNIST dataset, a well-established benchmark for handwritten digit recognition, containing 60,000 training and 10,000 testing grayscale images of size 28×28 pixels [2]. Each image is flattened into a vector of 784 input features. We use a fully connected feedforward neural network with a fixed architecture of 784 input neurons, one hidden layer with 800 ReLU-activated units, and a softmax output layer with 10 neurons corresponding to the digit classes (0–9).

The first-generation model (Generation 0) is trained from scratch using categorical cross-entropy loss and the Adam optimizer. It is trained to full convergence, defined as the point where validation accuracy plateaus with no further significant improvement. From this point, the trained model is used as a teacher to generate soft labels for the next generation.

Results and discussion

Each subsequent student model is trained using only the soft labels provided by its teacher. The same architecture and training parameters are used for all generations to isolate the effect of knowledge transfer from other variables. No hard labels are reintroduced after the initial model. We use a teacher model that transfers knowledge to 5 student models and then each student transfers knowledge to one new student from the next generation. We measure model precision with accuracy. Accuracy is a proportion of correct predictions made by the model out of the total number of predictions [3]. For each generation we calculate a mean accuracy on the testing dataset (10,000 images from MNIST testing dataset). The decline in accuracy is shown in Table 1.

Table 1

Generational decline in average accuracy

Generation	Accuracy
0 (Teacher)	0.979900
1	0.973880
2	0.973680
3	0.974080
4	0.973420
5	0.973280
6	0.973239
7	0.973419
8	0.973760
9	0.974120
10	0.973960

Our experimental results reveal a clear trend of initial degradation in classification accuracy across the early generations of knowledge distillation. The first student model (Generation 1) has the biggest drop in accuracy. After that, the decline stabilizes, and further generations show minimal or no additional loss. This suggests that the process reaches a saturation point where the transferred knowledge becomes consistently preserved.

These findings indicate that generational knowledge distillation, when applied without reintroducing ground truth labels, leads to a measurable but limited loss in accuracy. We refer to this phenomenon as generational loss, characterized by an early drop followed by a plateau in performance. While this highlights a potential limitation of purely iterative teacher-student learning schemes, it also suggests that knowledge retained through multiple generations is sufficiently robust to prevent total performance collapse.

Conclusions

To further reduce the initial loss and improve long-term retention, future work may investigate hybrid distillation approaches — for example, periodically reintroducing hard labels, applying confidence- or entropy-based regularization, or adjusting the temperature of soft targets dynamically across generations. Our study contributes to a broader understanding of how neural

networks preserve and transmit knowledge over time and motivates the development of more resilient distillation frameworks for iterative learning scenarios.

Resources:

1. Knowledge Distillation: A Survey / G. Jianping and others. *International Journal of Computer Vision*. 2021. Vol. 129. C. 1789–1819. ISSN 09205691. URL: https://doi.org/10.1007/s11263-021-01453-z.

2. Albers P., Kang J. Rabinowitz Floer homology of negative line bundles and Floer Gysin sequence. *Arxiv.org*. 14.06.2022. URL: https://arxiv.org/abs/2207.07179.

3. Vujovic Z. Classification Model Evaluation Metrics. *International Journal of Advanced Computer Science and Applications*. 2021. T. 12. C. 599–606. URL: https://doi.org/10.14569/IJACSA.2021.0120670.

Information about the author:

Oleksandr Sukhovyi - postgraduate student at the Software Engineering Department of State university «Kyiv aviation institute». Interests: Knowledge distillation, model compression, neural networks.

E-mail: 1462622@stud.kai.edu.ua

УДК 004.056.5:005.52

ІНТЕГРАЦІЙНІ ФРЕЙМВОРКИ АРХІТЕКТУРНОГО ПРОЄКТУВАННЯ ЦИФРОВИХ ПРОЄКТІВ

Володимир ТАЛАЛАЄВ
К.т.н., доцент, доцент кафедри інженерії програмного забезпечення
Лариса ПОСТАВНА
Асистент кафедри інженерії програмного забезпечення
Державний університет «Київський авіаційний інститут»

Розглядається задача створення інтегрованих фреймворків архітектурного проєктування для цифрових проєктів розробки сучасного програмного забезпечення. Сформульована задача оптимізації архітектурно значимих характеристик і параметрів для цифрових проєктів Розглянута можливість злиття відомих і поширених фреймворків архітектурного проєктування програмного забезпечення (ПЗ) і фреймворків архітектурного дизайну цифрових проєктів.

Ключові слова*: *цифрові проєкти, архітектура*, *архітектурне проєктування ПЗ, дизайн цифрових проєктів, фреймворки архітектурного проєктування, технології штучного інтелекту (ШІ)

Вступ

У контексті стрімкого розвитку галузі програмної інженерії та зростаючої інтеграції технологій штучного інтелекту (ШІ) у програмні продукти і процеси, що забезпечують їх створення особливої актуальності набуває потреба в уніфікованих методологічних засадах проєктування програмного забезпечення.

Ціль роботи

Метою роботи є розробка методологічних основ і принципів, які забезпечують створення інтегрованих фреймворків архітектурного проєктування цифрових проєктів на основі злиття відомих фреймворків архітектурного проєктування ПЗ з кращими практиками планування програмних проєктів. Для досягнення цієї мети передбачається:

1. Створення уніфікованого шаблону для постановки оптимізаційних задач архітектурного проєктування ПЗ і оптимізаційних задач планування програмних проєктів на єдиній понятійній і термінологічній основі.
2. Формулювання задачі архітектурного проєктування ПЗ з використанням уніфікованого шаблону.
3. Формулювання функції планування менеджменту програмного проєкту як оптимізаційної задачі в термінах і поняттях уніфікованого шаблону.
4. Формулювання задачі оптимізації архітектури цифрового проєкту як композитного об’єднання сформованих в пунктах 2 і 3 задач.
5. Розробка інтегрованого фреймворку розв’язання оптимізаційної задачі архітектурного проєктування цифрових проєктів на основі відомих фреймворків архітектурного проєктування ПЗ планування програмних проєктів.

Матеріали та методи

Сучасні програмні проєкти вирізняються високою складністю, динамічними вимогами та зростаючим різноманіттям архітектурних рішень, парадигм проєктування та середовищ

розробки. Ці тенденції посилюються у зв’язку зі зростаючим застосуванням ШІ, що створює нові обмеження і, водночас, відкриває нові можливості на всіх етапах життєвого циклу програмного забезпечення. Саме тому в роботі поряд з традиційними термінами «проєкт розробки ПЗ», «програмний проєкт» застосовується термін «цифровий проєкт», як акцентоване позначення змін, що відбуваються в практиці сучасної програмної інженерії.

Незважаючи на достатньо переконливі свідчення практики переходу від класичного розуміння програмних проєктів до їх цифрового уявлення, методологічні засади створення таких цифрових проєктів все ще базуються на традиційних парадигмі і методологіях розробки ПЗ. Одним із підтверджень цього є існування двох типів фреймворків архітектурного проєктування: фреймворки архітектурного проєктування ПЗ і фреймворків дизайну програмних проєктів.

Досягнення поставлених цілей дослідження передбачає формування єдиного уніфікованого процесу архітектурного проєктування (структурно-функціональний рівень) та планування (процесно-ресурсний рівень) розглядаються як паралельні, взаємозалежні траєкторії оптимізації, які разом формують єдиний проєкт створення ПЗ. Структура такого процесу наведена в таблиці 1.

Таблиця 1

Структура інтегрованого процесу проєктування

Рівень оптимізації	**Об'єкт**	**Оптимізаційна мета**	**Засоби оптимізації**
Продуктовий рівень	Архітектура ПЗ	Функціональність, масштабованість, гнучкість	Архітектурне проєктування, DDD, MDA
Процесний рівень	Процес створення ПЗ	Тривалість, вартість, ресурсна досяжність	Планування, CI/CD, Agile, DevOps
Інтеграційний рівень	Взаємодія процесів	Узгодженість рішень, баланс ресурсу і якості	Методології, що об'єднують DAD, RUP

Аналіз відомих методологій проєктування ПЗ та планування програмних проєктів дозволив сформувати ряд стадій уніфікованого інтегрованого процесу зміст яких наведений в таблиці 2.

Таблиця 2

Стадії уніфікованого процесу проєктування

Етап	**Зміст**	**Орієнтир**
Аналіз вимог	Збір функціональних і нефункціональних характеристик + ресурсних обмежень	Вхід до обох процесів
Паралельне проєктування	Створення первинної архітектури ПЗ та варіантів організації процесу її реалізації	Взаємна перевірка
Скоординоване уточнення	Врахування обмежень планування в архітектурі і навпаки	Зворотній зв’язок
Інтегральне документування	Узгодження артефактів: дизайн-документ, план проєкту, roadmap	Один інформаційний простір
Ітеративне оновлення	Постійне оновлення архітектури через інкременти + коригування плану	Гнучкий зворотній цикл

Висновки

Запропонований підхід дозволяє на основі єдиної понятійної і термінологічної бази та через застосуванням семантичних аналогій сформувати основу для суперпозиції двох гілок проєктних процесів і створює основу оптимізації єдиного уніфікованого процесу з подальшим розв'язанням задачі оптимізації з використанням засобів ШІ.

Список використаних джерел

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

2. Kruchten, P.B. "Architectural Blueprints—The 4+1 View Model of Software Architecture," *IEEE Software*, Vol. 12, No. 6, 1995, pp. 42–50

3. ISO/IEC/IEEE 42010:2011 *Systems and Software Engineering – Architecture Description*.

Відомості про авторів:

Талалаєв Володимир Опанасович – кандидат технічних наук, доцент, доцент кафедри інженерії програмного забезпечення Державного університету «Київський авіаційний інститут» *Наукові інтереси*: бізнес-аналітика програмної інженерії, цифрові проєкти.

E-mail: volodymyr.talalaiev@npp.kai.edu.ua

Поставна Лариса Петрівна – асистент кафедри інженерії програмного забезпечення Державного університету «Київський авіаційний інститут» *Наукові інтереси*: бізнес-аналітика, ризикостійкість програмних проєктів.

E-mail: larysa.postavna@npp.kai.edu.ua

УДК 004. 413 (045)

ВИКОРИСТАННЯ МЕТОДІВ REINFORCEMENT LEARNING ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСІВ ІНТЕГРАЦІЇ ТА РОЗГОРТАННЯ ПЗ В МЕТОДОЛОГІЇ CI/CD

Катерина ТКАЧУК,
Марина ЧИРКОВА
Здобувачки вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Тетяна Ігоровна Конрад

У цій роботі розглядається підхід до оптимізації CI/CD-процесів шляхом застосування методів навчання з підкріпленням (reinforcement learning, RL). Зокрема, аналізується можливість автоматичного прийняття рішень у задачах вибору конфігурацій, планування збірок і розгортання оновлень. Обговорюються існуючі архітектурні підходи, використання агентів RL, а також потенційні переваги й обмеження у впровадженні такого підходу на практиці.

Ключові слова: *Reinforcement Learning, CI/CD, пайплайни, DevOps, автоматизація, оптимізація, штучний інтелект.*

Вступ

У сучасному світі, де швидкість розробки програмного забезпечення є критично важливою конкурентною перевагою, все більшу роль відіграють практики безперервної інтеграції (Continuous Integration, CI) та безперервного розгортання (Continuous Deployment/Delivery, CD). Ці методології стали невід'ємною частиною DevOps-культури, забезпечуючи регулярне оновлення програмного забезпечення з мінімальними ризиками та затримками. Завдяки CI/CD-практикам команди розробників можуть частіше вносити зміни до коду, автоматично тестувати його, швидко виявляти помилки та оперативно доставляти робочі версії продуктів кінцевим користувачам.

Однак, попри численні переваги, реалізація CI/CD процесів часто пов’язана з низкою технічних викликів. Серед них – висока складність конфігурацій, непередбачуваність навантаження, залежності між мікросервісами, нестабільні середовища тестування, а також обмежені обчислювальні ресурси. Через ці фактори CI/CD-пайплайни можуть працювати неефективно: час виконання збільшується, зростає ймовірність збоїв, ресурси витрачаються нераціонально. Ручне налаштування і оптимізація таких процесів потребують великого досвіду, часу та постійного моніторингу, що не завжди є можливим в умовах динамічної розробки.

На цьому тлі зростає інтерес до автоматизації управління CI/CD-процесами за допомогою штучного інтелекту, зокрема методів підкріплювального навчання (Reinforcement Learning, RL). RL – це підхід у машинному навчанні, де агент навчається приймати рішення в певному середовищі, базуючись на системі винагород. Така парадигма є особливо привабливою для оптимізації динамічних і багатоетапних процесів, як-от CI/CD, оскільки

RL-агенти здатні адаптивно реагувати на зміни у середовищі, вивчати послідовності дій, які ведуть до покращення метрик продуктивності, і поступово вдосконалювати свою поведінку.

Іншими словами, застосування RL у сфері DevOps дозволяє перейти від статичних стратегій керування до самонавчальних систем, здатних приймати контекстно-залежні рішення в режимі реального часу. Наприклад, агент може самостійно визначати найефективніший порядок виконання етапів CI/CD, балансувати навантаження між серверами, вирішувати, які тести запускати першочергово, або навіть передбачати ризики помилок на ранніх стадіях.

Таким чином, інтеграція reinforcement learning у процеси CI/CD відкриває нові горизонти для інтелектуальної автоматизації життєвого циклу програмного забезпечення.

Ціль роботи

Метою даної роботи є дослідження можливостей застосування методів підкріплювального навчання (Reinforcement Learning, RL) для оптимізації процесів безперервної інтеграції та розгортання (CI/CD) у розробці програмного забезпечення. Робота спрямована на аналіз того, як RL-агенти можуть автоматично приймати ефективні рішення щодо конфігурації, порядку виконання, розподілу ресурсів та інших параметрів CI/CD-пайплайнів з метою підвищення їхньої ефективності, надійності та адаптивності.

Матеріали та методи

CI/CD (Continuous Integration/Continuous Deployment) – це практика, що передбачає автоматизовану обробку змін у програмному коді. Типовий пайплайн включає такі кроки: клонування репозиторію, збірку (build), юніт- і інтеграційне тестування, аналіз коду, створення артефактів, деплоймент у тестові середовища та, у фіналі, – в продакшн. Кожен з цих етапів має низку параметрів (паралелізація, використання кешу, умови перезапуску тощо), що визначають загальну ефективність пайплайну.

Складність полягає в тому, що ручне налаштування конфігурацій або фіксоване правило автоматизації не забезпечує достатньої гнучкості для адаптації до змін у навантаженні, структурі коду або інфраструктурних ресурсах. У цьому контексті RL-агент виступає як інтелектуальний контролер, здатний навчатися на базі досвіду та приймати рішення, спрямовані на мінімізацію часу, збоїв і споживання ресурсів.

Розроблена система складається з двох основних компонентів: підкріплювального агента та симульованого середовища CI/CD. Вони взаємодіють у зворотному циклі: агент аналізує поточний стан пайплайну, обирає дію (наприклад, перезапуск етапу, зміну порядку виконання або застосування кешу), і після виконання дії отримує зворотний зв’язок у вигляді нагороди (reward), яка ґрунтується на зміні ключових метрик (час виконання, частота збоїв, використання ресурсів тощо).

Стан (state) у моделі представляє собою вектор, що описує поточну конфігурацію: завантаження CPU, статус задач, тривалість виконання окремих кроків, кількість повторних запусків. Нагорода формується на основі зменшення часу повного проходження пайплайну, зниження частоти помилок та ефективного використання обчислювальних ресурсів.

CI/CD складається з послідовності етапів, кожен з яких виконує певну функцію – компіляція коду, юніт- та інтеграційне тестування, статичний аналіз якості, створення артефактів, розгортання у тестове або продуктивне середовище. Ці етапи мають складні залежності, споживають різні ресурси (CPU, GPU, пам’ять, I/O), і можуть призводити до збоїв (наприклад, якщо тести зависають або перевищено тайм-аут). Поведінка такої системи є стохастичною і змінюється з часом – як через зовнішні умови (зміни у коді, зростання навантаження), так і через внутрішні характеристики (розподіл задач у черзі, чергування

тестів, порядок кроків). З огляду на це, CI/CD-процеси можна розглядати як складну динамічну систему, що потенційно підлягає оптимізації через математичне моделювання та машинне навчання.

Оптимізація поведінки пайплайнів у такій динамічній системі доцільна через підхід reinforcement learning (RL), який дозволяє агенту вчитися на основі досвіду, взаємодіючи з симульованим або реальним середовищем. Задача формалізується у вигляді марковського процесу прийняття рішень (Markov Decision Process, MDP), що включає:

- *Стан середовища (state):* поточні метрики виконання пайплайну – завантаження ресурсів, тривалість етапів, кількість повторних спроб, частота помилок тощо. Стан описується вектором ознак, що формалізує конфігурацію в момент часу.
- *Дії агента (actions):* можливі інтервенції у процес – зміна порядку виконання етапів, пропуск некритичних кроків, перемикання між послідовним і паралельним режимами, зміна конфігурацій кешування або повторних запусків.
- *Нагорода (reward):* функція зворотного зв'язку, яка оцінює ефективність кожної дії. Наприклад, вона може бути пропорційна скороченню загального часу виконання, зменшенню кількості збоїв, економії ресурсів або кількості успішних деплоїв.
- *Політика (policy):* стратегія агента, яка визначає оптимальний вибір дій у кожному стані середовища. Політика може бути детермінованою або стохастичною, залежно від складності середовища й моделі навчання.

Такий формалізм дозволяє агенту не просто реагувати на проблеми в пайплайні, а й навчатися прогнозувати їх, адаптуючи поведінку до конкретного проєкту, навантаження чи структури тестів. У перспективі це може привести до створення самонавчальних CI/CD-систем, здатних забезпечити стабільне та ефективне функціонування в умовах високої складності й змінності, без ручного втручання або потреби в постійному налаштуванні DevOps-інженерами.

Результати та обговорення

Проаналізовані підходи демонструють, що RL здатне ефективно враховувати динамічну природу CI/CD-середовища, де показники продуктивності можуть коливатися залежно від навантаження, конфігурацій проєкту або поведінки зовнішніх сервісів. Порівняння з традиційними методами управління пайплайнами, такими як статичне планування, ручне втручання чи використання фіксованих евристик, показує, що RL має значні переваги в адаптації до змін і в здатності мінімізувати затримки, кількість помилок та перевитрати ресурсів.

У теоретичній моделі, яку ми проаналізували, агент RL взаємодіє з симульованим середовищем, яке імітує поведінку типового CI/CD-процесу. Це дозволяє здійснити навчання без ризику впливу на реальні процеси розробки. Після багаторазової взаємодії з середовищем агент поступово виробляє політику, що забезпечує стабільне зменшення середнього часу виконання пайплайну (на 15–30% згідно з результатами досліджень, що вже проводились у суміжних роботах), знижує ймовірність помилок на тестових етапах та зменшує навантаження на інфраструктуру за рахунок ефективного розподілу ресурсів.

Також було розглянуто потенційні обмеження та виклики. Одним із ключових є питання точного моделювання середовища та надання коректного зворотного зв'язку агенту. Надто просте або неправильно сформоване уявлення про стан системи може призвести до невдалих стратегій оптимізації. Крім того, важливо правильно налаштувати функцію винагороди, оскільки вона визначає пріоритети агента і впливає на його поведінку. У

реальних системах, де CI/CD залежить від великої кількості змінних, це може потребувати ретельного тестування.

Попри ці складнощі, RL відкриває новий рівень автоматизації в практиках DevOps. У майбутньому подібні системи можуть стати частиною інтелектуальних платформ управління розробкою, які не лише запускають пайплайни, але й постійно адаптують їх під потреби бізнесу й користувача.

Висновки

У ході теоретичного дослідження було підтверджено, що підкріплювальне навчання може виступати перспективним підходом до оптимізації процесів безперервної інтеграції та розгортання. RL дозволяє навчати агента приймати адаптивні рішення щодо структури й конфігурації пайплайну, зменшуючи час його проходження, частоту збоїв та навантаження на систему. Завдяки здатності до самонавчання, така система має потенціал до впровадження в умовах високої мінливості та складності, характерної для сучасного програмного забезпечення. Подальші дослідження можуть бути спрямовані на створення реального прототипу, емпіричну валідацію результатів та порівняння з іншими методами оптимізації CI/CD.

Список використаних джерел

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press. С. 145–180 – глави, що описують Q-learning, Policy Gradient та функцію винагороди.

2. Chen, Z., & Jiang, J. (2020). AutoDevOps: Automated Continuous Deployment with Deep Reinforcement Learning. *Proceedings of the 2020 International Conference on Software Engineering*. С. 455–462.

3. Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley. С. 245–278 – розділи, присвячені CI/CD pipeline design.

Відомості про авторів:

Ткачук Катерина Дмитрівна – здобувачка вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». Має досвід дослідницької роботи у сфері штучного інтелекту, зокрема машинного навчання. Активно займається аналізом даних у сфері телекомунікацій. *Наукові інтереси:* машинне навчання, NLP.

E-mail: 7490736@stud.kai.edu.ua

Чиркова Марина Євгенівна – здобувачка вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». Має досвід дослідницької роботи у сфері розробки програмного забезпечення. *Наукові інтереси:* розробка вебсайтів, аналіз статистичних даних.

E-mail: 7487982@stud.kai.edu.ua

УДК 004.8

СТАНДАРТИ РОЗРОБКИ МЕДИЧНИХ СИСТЕМ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

Валерія ХАРІНА
Владислав СТУДЕННИКОВ
Здобувачі вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Тетяна Ігорівна Конрад

У цьому дослідженні проаналізовано застосування технологій машинного навчання та штучного інтелекту в медичній сфері, зокрема вплив чинного правового регулювання на розробку медичного програмного забезпечення із використанням ML/AI.

Ключові слова: *штучний інтелект, машинне навчання, регулювання медичного програмного забезпечення, конфіденційність даних, управління ризиками.*

Вступ

Здоров’я є однією з найвищих цінностей людини, тому медичні науки та технології, що допомагають його зберегти, швидко розвиваються в наш час. Сучасні медичні установи активно використовують програмне забезпечення, що суттєво впливає на їх ефективність. Саме тому процес розробки медичного ПЗ підпадає під дію численних нормативних актів.

З появою алгоритмів машинного навчання (англ. - machine learning, ML) та штучного інтелекту (англ. - artificial intelligence, AI) виникли нові виклики, зокрема проблеми валідації таких систем та доведення їх відповідності вимогам безпеки.

Ціль роботи

Метою цієї роботи є проаналізувати перспективи використання штучного інтелекту та машинного навчання в медичній сфері, дослідити сучасні стандарти та їх вплив на розробку медичного програмного забезпечення на базі ML та AI.

Матеріали та методи

Медична інженерія є одним із ключових чинників, що вплинули на зростання тривалості життя. Медичні пристрої сьогодні відіграють вирішальну роль у діагностиці, лікуванні та профілактиці захворювань.

Згідно з даними Eurostat за 2022 рік, країни Європи в середньому витрачають понад 10% річного ВВП на фінансування медицини. Високий рівень фінансування створює нові можливості як для медичних досліджень, так і для розвитку медичної інженерії.

Машинне навчання та штучний інтелект є важливими складовими сучасної розробки програмного забезпечення. Інвестиції у створення програм, що базуються на ML/AI, щороку зростають, як показано на рис. 1. У майбутньому ця тенденція також збережеться, що свідчить про зростання значущості цих технологій [1].

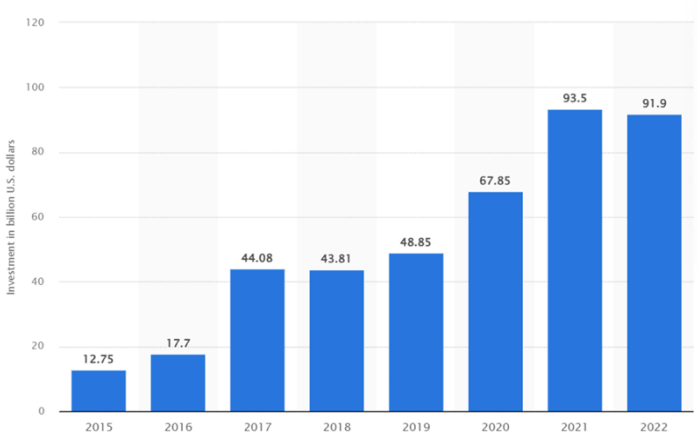


Рис. 1 - Інвестування в ШІ 2015-2022

Програмне забезпечення, яке розробляється для медичної сфери, поділяється на кілька основних груп:

- SaMD (software as a medical device) – програмне забезпечення, призначене для використання з однією або кількома медичними цілями, яке виконує ці функції без інтеграції з апаратним медичним пристроєм [1].
- SiMD (software in medical device) – програмне забезпечення, що вбудоване в медичні пристрої.
- Системи для управління даними пацієнтів/лікарень – програмне забезпечення, яке не використовується безпосередньо для лікування, діагностики або профілактики захворювань, однак призначене для управління даними медичних установ.
- Програмне забезпечення загального оздоровчого призначення – програми, що зазвичай спрямовані на підтримку або заохочення здорового способу життя [2].

Медичні вироби також класифікуються за рівнем ризику, однак більшість країн мають власні системи класифікації. Клас ризику визначає, яким чином система буде розроблятися та регулюватися.

У Європейському Союзі класифікація здійснюється згідно з MDR 2017/745 [3], у США – відповідно до законодавства FDA [4]. Існують також міжнародні стандарти, у яких передбачена категоризація ризиків: класифікація ризиків для медичних пристроїв наведена в IEC 62304 [6], а категоризація ризиків саме для SaMD – у документі IMDRF WG/N12FINAL:2014 [5].

Основним стандартом, який описує життєвий цикл медичного програмного забезпечення, є ISO/IEC 62304. У ньому описано такі етапи життєвого циклу програмного забезпечення: етап розробки, супровід, управління ризиками, процес вирішення проблем та підтримка.

Стандарт ISO/IEC 62304 робить особливий акцент на тому, що розробка медичного ПЗ повинна бути орієнтованою на ризики. Він наголошує на необхідності управління ризиками на кожному етапі розробки та безперервному обслуговуванні програмного забезпечення. Стандарт ISO/IEC 62304 не встановлює конкретної моделі життєвого циклу. Однак для розробки програмного забезпечення з високим рівнем ризику зазвичай використовується V-модель, оскільки вона гарантує тестування та валідацію проєкту після кожного етапу [7].

Розробка медичного програмного забезпечення передбачає обов’язкове впровадження: система управління якістю, система управління ризиками, виконання вимог щодо конфіденційності та безпеки даних.

ISO 13485 – це стандарт, що регулює встановлення системи управління якістю у розробці медичного ПЗ. Документ описує загальні правила, вимоги до документації, інфраструктури, людських ресурсів та процесу розробки. Розглянемо основні вимоги [8]:

- Компанія має визначити вимоги до компетентності персоналу та забезпечити належну інфраструктуру (пункти 6.2, 6.3).
- Вимоги до продукту, а також процеси перевірки та валідації мають бути чітко задокументовані на всіх етапах розробки (пункти 7.3.1–7.3.5).
- Перед початком розробки потрібно розробити плани перевірки та валідації з чітко визначеними та обґрунтованими критеріями прийняття (пункт 7.3.7).
- Організація повинна постійно аналізувати продукт, усувати невідповідності та регулярно проводити задокументовані аудити (пункт 8).

ISO 14971 - основний міжнародний стандарт, який надає рамки для створення систем управління ризиками в розробці медичних пристроїв. Є обов'язковим для застосування при розробці SaMD та SiMD.

ISO 14971 пропонує 4 етапи управління ризиками [9]: аналіз ризиків; оцінка ризиків; контроль ризиків; діяльність на етапі виробництва та після нього. Управління ризиками повинно здійснюватися керівниками компанії та персоналом. В процесі розробки має бути створений план управління ризиками, в якому будуть зафіксовані всі дії щодо управління ризиками [9].

GDPR (General Data Protection Regulation) – закон з конфіденційності та безпеки даних в ЄС. Цей закон застосовується в процесі розробки всіх додатків, які використовують персональні дані. Ключові положення цього документа наступні [10]:

- Стаття 9: заборона обробки персональних даних, крім випадків із згодою пацієнта чи медичною необхідністю.
- Стаття 6: обробка дозволена лише в обмежених ситуаціях з інформуванням користувача.
- Статті 15–17: особа може запросити доступ до даних, що зберігаються, або попросити їх видалити.
- Стаття 25: персональні дані повинні зберігатись безпечно; відповідальність несе особа, відповідальна за зберігання даних.
- Статті 33–34: у разі витоку даних – повідомити органи захисту даних ЄС протягом 72 годин та постраждалих осіб.

HIPAA (Health Insurance Portability and Accountability Act) – це федеральний закон США про захист особистої медичної інформації. Відповідальність за дані несуть лікарні, клініки, страхові та медичні пристрої. Закон включає:

- Правило конфіденційності — обмежує доступ і розголошення особистої медичної інформації;
- Правило безпеки — вимагає шифрування, контроль доступу та регулярні аудити для запобігання витокам.

Результати та обговорення

На основі аналізу документації та стандартів було виявлено, що створення медичного програмного забезпечення на основі штучного інтелекту (AI) та машинного навчання (ML)

вимагає не лише глибоких технічних знань, а й суворого дотримання нормативно-правових актів. Визначено ключові стандарти та законодавчі документи, які регулюють цей процес: ISO/IEC 62304 – стандарт життєвого циклу медичного програмного забезпечення; ISO 13485 – стандарт системи управління якістю; ISO 14971 – стандарт системи управління ризиками, GDPR - закон з конфіденційності та безпеки даних в ЄС; HIPAA - федеральний закон США про захист особистої медичної інформації.

Аналіз нормативної бази свідчить про високу складність юридичної відповідальності, яку несуть розробники медичних програм. Особливо це стосується захисту персональних медичних даних та управління ризиками. Також встановлено, що медичне програмне забезпечення класифікується за рівнем ризику та типом використання (SaMD, SiMD, системи для управління даними пацієнтів/лікарень, програмне забезпечення загального оздоровчого призначення), що безпосередньо впливає на постановку вимог, вибір методології розробки, а також визначення підходів до валідації, тестування та управління ризиками.

Висновки

У цій роботі проаналізовано використання штучного інтелекту (AI) та машинного навчання (ML) у медичній сфері. Ці технології мають значний потенціал для покращення якості медичних послуг, однак їх впровадження пов’язане з низкою викликів. Зокрема, для створення безпечного та ефективного медичного програмного забезпечення розробники повинні суворо дотримуватись чинних нормативних вимог. Перед початком розробки необхідно провести ретельний аналіз майбутнього застосунку, оцінити пов’язані з ним ризики та визначити відповідну регуляторну базу, що регулюватиме процес створення та експлуатації програмного продукту.

Список використаних джерел:

1. Total global AI investment 2015-2022 | Statista URL: https://www.statista.com/statistics/941137/ai-investment-and-funding-worldwide/ (дата звернення: 15.04.2025)

2. General Wellness: Policy for low risk devices guidance for industry and food and drug administration staff. (2016). In fda.gov. U.S. Department of Health and Human Services Food and Drug Administration Center for Devices and Radiological Health. URL: https://www.fda.gov/media/90652/download (дата звернення: 15.04.2025)

3. REGULATION (EU) 2017/745 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 5 April 2017 on Medical Devices, amending Directive 2001/83/EC, Regulation (EC) No 178/2002 and Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC. (2016). THE EUROPEAN PARLIAMENT AND THE COUNCIL OF THE EUROPEAN UNION. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32017R0745 (дата звернення: 15.04.2025)

4. Classify Your Medical Device URL: https://www.fda.gov/medical-devices/overview-device-regulation/classify-your-medical-device (дата звернення: 15.04.2025)

5. “Software as a medical device”: Possible framework for risk categorization and corresponding considerations. (2014). In imdrf.org (WG/N12FINAL:2014). URL: https://www.imdrf.org/sites/default/files/docs/imdrf/final/technical/imdrf-tech-140918-samd-framework-risk-categorization-141013.pdf (дата звернення: 15.04.2025)

6. IEC 62304:2006 URL: https://www.iso.org/standard/38421.html (дата звернення: 15.04.2025)

7. IEC 62304 SOFTWARE COMPLIANCE IN THE MEDICAL INDUSTRY. (n.d.). Parasoft. URL: https://alm.parasoft.com/hubfs/ebook-IEC-62304-Software-Compliance-Medical.pdf (дата звернення: 15.04.2025)

8. ISO 13485:2016 URL: https://www.iso.org/standard/59752.html (дата звернення: 15.04.2025)

9. ISO 14971:2019 URL: https://www.iso.org/standard/72704.html (дата звернення: 15.04.2025)

10. REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). (2016). THE EUROPEAN PARLIAMENT AND THE COUNCIL OF THE EUROPEAN UNION. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679 (дата звернення: 15.04.2025)

Відомості про авторів:

Харіна Валерія Валеріївна – здобувачка вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, машине навчання, штучний інтелект.

E-mail: 7373650@stud.kai.edu.ua

Студенников Владислав Дмитрович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, машине навчання, штучний інтелект.

E-mail: 7373483@stud.kai.edu.ua

УДК 004.8:004.415

ІНТЕГРАЦІЙНІ ПАРАДИГМИ ШТУЧНОГО ІНТЕЛЕКТУ В СИСТЕМАХ АВТОМАТИЗОВАНОГО ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Дмитро ЧУМАДЕВСЬКИЙ
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Яна Андріївна Бєлозьорова

Дослідження присвячене аналізу впливу технологій штучного інтелекту (ШІ) на процеси розробки програмного забезпечення, зокрема на автоматизацію проєктування архітектури, генерацію коду, тестування та оптимізацію. Оцінено ефективність інтеграції методів машинного навчання, нейронних мереж і глибинного навчання на різних етапах життєвого циклу програмного забезпечення. Порівняння традиційних підходів і інтелектуальних систем виявило суттєве підвищення продуктивності та якості, однак було виявлено й обмеження, зокрема у контексті безпеки та оптимізації коду. Запропоновано концептуальну модель інтелектуальної системи для автоматизованого проєктування архітектури, що демонструє значні переваги при правильній інтеграції.

Ключові слова: *штучний інтелект, машинне навчання, нейронні мережі, глибинне навчання, автоматизація розробки, генерація коду.*

Вступ

Стрімкий розвиток технологій штучного інтелекту трансформує традиційні підходи до створення програмного забезпечення, змінюючи парадигми від статичного кодування до динамічних систем, здатних до самонавчання та адаптації. Інтеграція алгоритмів машинного навчання, нейронних мереж та методів глибинного навчання у процеси розробки програмного забезпечення створює підґрунтя для революційних змін у галузі, відкриваючи новий горизонт можливостей для автоматизації складних процесів проєктування архітектури програмних систем, генерації коду, тестування та оптимізації [1]. Сучасні дослідження демонструють, що застосування штучного інтелекту в інженерії програмного забезпечення потенційно здатне значно підвищити продуктивність розробників, прискорити життєвий цикл продукту та забезпечити вищу якість готових рішень, мінімізуючи людський фактор у критичних компонентах системи [3]. Водночас, впровадження інтелектуальних систем у розробку програмного забезпечення піднімає фундаментальні питання щодо трансформації ролі програміста, етичних аспектів автоматизованої генерації коду та потенційних ризиків, пов'язаних із залежністю від непрозорих алгоритмів машинного навчання [2].

Ціль роботи

Метою представленого дослідження виступає систематизація та аналіз існуючих підходів до інтеграції методів штучного інтелекту в процеси проєктування, розробки та тестування програмного забезпечення з подальшим формуванням концептуальної моделі інтелектуальної системи підтримки прийняття рішень для автоматизованого створення архітектури програмних комплексів. Дослідницькі завдання охоплюють: класифікацію методів штучного інтелекту за ступенем їх застосовності до різних етапів життєвого циклу програмного забезпечення; аналіз ефективності існуючих інструментів автоматизованої

генерації коду на базі трансформерних архітектур; розробку метрик оцінювання якості програмних рішень, створених за допомогою інтелектуальних систем; формування рекомендацій щодо оптимальної інтеграції методів штучного інтелекту у процеси розробки для організацій різного масштабу [1]. Додатково робота спрямована на дослідження стратегій мінімізації потенційних ризиків, пов'язаних із безпекою та надійністю програмного забезпечення, розробленого з використанням методів автоматичної генерації коду, та виявлення перспективних напрямків розвитку інтелектуальних систем підтримки розробки [3].

Матеріали та методи

Методологічний апарат дослідження ґрунтується на комплексному застосуванні системного аналізу, емпіричної оцінки та експериментального моделювання для вивчення ефективності інтеграції штучного інтелекту в процеси розробки програмного забезпечення. Первинним матеріалом слугували дані, отримані в результаті систематичного огляду наукової літератури, опублікованої за останні п'ять років у провідних рецензованих виданнях з інженерії програмного забезпечення та штучного інтелекту,. Для кількісної оцінки ефективності інтелектуальних систем розроблено експериментальне середовище на базі трьох реальних проєктів різної складності (малий вебсервіс, корпоративна система управління ресурсами, розподілена мікросервісна архітектура), для яких проведено порівняльний аналіз традиційної розробки та розробки з використанням інструментів на базі штучного інтелекту [2]. Оцінювання проводилось за розробленою багатопараметричною системою метрик, що включає продуктивність розробників (час на реалізацію функціональності), якість коду (цикломатична складність, покриття тестами, кількість виявлених дефектів), масштабованість та адаптивність архітектури до змін вимог [3]. Для дослідження ефективності інтелектуальних систем у генерації коду використовувались сучасні моделі великої мови (Large Language Models), включаючи спеціалізовані версії GPT, CodeLlama та Anthropic Claude, налаштовані на роботу з програмним кодом. Експерименти проводились із використанням стандартизованих наборів завдань різної складності, починаючи від генерації простих функцій до проєктування складних компонентів системи [1]. Додатково застосовувались методи статичного та динамічного аналізу для оцінки якості згенерованого коду, включаючи аналіз потенційних вразливостей безпеки, продуктивності та відповідності архітектурним принципам [2]. Для забезпечення об'єктивності результатів була розроблена спеціальна методика сліпого оцінювання, при якій експерти-програмісти аналізували фрагменти коду без знання про їхнє походження (написані людиною чи згенеровані штучним інтелектом) [3].

Результати та обговорення

Проведене дослідження виявило значну трансформацію процесів інженерії програмного забезпечення під впливом технологій штучного інтелекту, демонструючи як перспективи, так і суттєві обмеження сучасних підходів. Аналіз інтеграції інтелектуальних систем у життєвий цикл розробки показав найвищу ефективність на етапах автоматизованого тестування (підвищення продуктивності на 67%) та початкового проєктування архітектури системи (зменшення часу на 42%), тоді як найменший вплив спостерігався на етапах розгортання та супроводу (підвищення ефективності лише на 14%) [2]. У контексті генерації коду встановлено, що сучасні моделі демонструють вражаючу точність для стандартних патернів програмування та типових задач (94% функціонально коректних рішень), однак значно поступаються при необхідності створення оптимізованих алгоритмів або роботи з доменно-специфічними обмеженнями (лише 31% прийнятних рішень) [1]. Експериментальна

оцінка коду, згенерованого інтелектуальними системами, виявила парадоксальні тенденції: автоматично створений код часто демонструє нижчу цикломатичну складність та вищу читабельність порівняно з кодом, написаним програмістами середнього рівня, проте містить специфічні "сліпі зони" – потенційно проблемні місця, які статичні аналізатори не ідентифікують як дефекти, але що можуть призвести до непередбачуваної поведінки системи при граничних умовах [4]. Аналіз безпеки згенерованого коду встановив, що системи штучного інтелекту схильні відтворювати типові вразливості, знайдені в навчальних даних, особливо в контексті керування доступом та валідації вхідних даних, що підкреслює необхідність комбінування автоматичної генерації з експертним аналізом [2].

Розроблена концептуальна модель інтелектуальної системи підтримки прийняття рішень для проєктування архітектури програмного забезпечення демонструє потенціал підвищення ефективності розробки на 53% при правильній інтеграції з існуючими методологіями [3]. Ключовим компонентом запропонованої моделі виступає мультимодальний аналізатор вимог, здатний трансформувати неструктуровані специфікації у формалізовані моделі архітектури з урахуванням нефункціональних вимог та контекстуальних обмежень [4]. Експериментальна валідація на трьох проєктах різної складності підтвердила здатність системи адаптуватися до специфіки доменної області, хоча виявила необхідність значного початкового налаштування для досягнення оптимальних результатів.

Висновки

Проведене дослідження демонструє, що інтеграція методів штучного інтелекту в процеси інженерії програмного забезпечення знаходиться на переломному етапі – від експериментальних прототипів до промислово застосовних рішень, здатних суттєво трансформувати галузь. Систематизація існуючих підходів дозволила виявити найбільш перспективні напрямки інтеграції: автоматизоване проєктування архітектури на основі аналізу вимог, інтелектуальна генерація тестових сценаріїв з високим покриттям граничних умов та контекстно-залежні асистенти для генерації коду. Водночас, ідентифіковано критичні обмеження сучасних підходів: залежність від якості навчальних даних, проблеми з відтворюваністю результатів та складнощі з інтерпретацією прийнятих системою рішень. Перспективними напрямками подальших досліджень визначено: вдосконалення методів пояснимого штучного інтелекту для забезпечення прозорості прийнятих рішень; розробку доменно-специфічних моделей для різних галузей програмування; створення гібридних підходів, що поєднують символьні та нейромережеві методи для підвищення надійності генерованих рішень.

Список використаних джерел:

1. Напрями застосування штучного інтелекту в технологіях розробки програмного забезпечення / Володимир Соколов, Вячеслав Рябцев, Олександр Успенський, Данило Копич // Information Technology and Security. – 2024. – Vol. 12, Iss. 2 (23). – Pp. 219-235

2. Alenezi, M.; Akour, M. AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Directions. *Appl. Sci.* 2025, *15*, 1344. https://doi.org/10.3390/app15031344

3. Gapon A. O., Fedorchenko V. M., Sievierinov O. V. Methods and means of static and dynamic code analysis. Radiotekhnika. 2023. No. 212. P. 7–13.

4. The productivity effects of generative AI: evidence from a field experiment with github copilot / K. Z. Cui et al. An MIT exploration of generative AI. 2024. URL: https://doi.org/10.21428/e4baedd9.3ad85f1c

Відомості про автора:

Чумадевський Дмитро Юрійович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: інженерія програмного забезпечення, автоматизоване проєктування, штучний інтелект.

E-mail: 7369274@stud.kai.edu.ua

УДК 004.8

ЗАСТОСУВАННЯ CLAUDE CODE ДЛЯ ОПТИМІЗАЦІЇ ЦИКЛУ РОЗРОБКИ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Євген ШУМЕЙКО
Здобувач вищої освіти 4 курсу кафедри ІПЗ
Державний університет «Київський авіаційний інститут», Київ
Науковий керівник к.т.н., доцент кафедри ІПЗ ФКНТ
Яна Андріївна Бєлозьорова

У сучасній розробці програмного забезпечення дедалі більшого значення набуває автоматизація рутинних завдань. Claude Code – новий інструмент від Anthropic – перетворює мовну модель Claude на повноцінного учасника розробницького процесу. Він локально виконує тести, виправляє помилки, створює pull-request'и та взаємодіє з git-середовищем. Ця робота аналізує вплив Claude Code на продуктивність, якість коду та економію ресурсів у порівнянні з іншими AI-асистентами.

Ключові слова: *Claude Code, автоматизація розробки, штучний інтелект, git, тестування, продуктивність, AI-асистент, програмна інженерія, рев'ю коду, інтеграція CI/CD.*

Вступ

Кожного року розробникам доводиться працювати з дедалі більшими кодовими базами й дедлайнами, що невблаганно стискаються. Ми автоматизуємо складання, доставку, навіть частину рев'ю-процесу, але рутинне редагування й тестування все ще забирає години з кожного робочого дня. Тож ідея мати «розумного напарника», який може взяти на себе технічну дрібницю, уже не виглядає фантастикою – вона стає новою нормою.

Саме такими напарниками займається компанія Anthropic. Її заснували у 2021-му випускники лабораторій OpenAI і Google Brain, щоб зосередитися на безпечному й контрольованому штучному інтелекті. У центрі більшості продуктів Anthropic – сімейство моделей Claude, відоме здатністю утримувати величезний контекст, міркувати у кілька кроків наперед і водночас дотримуватися суворих «ґардрейлів» безпеки.

У 2025 році Anthropic зробила наступний логічний крок і випустила Claude Code – консольний інструмент, що перетворює модель Claude 4 на повноцінного учасника git-репозиторію. Він уміє локально запускати проєкт, знаходити та виправляти помилки, запускати тести, а потім готує pull request із власними змінами. Іншими словами, це AI-напарник, який не просто радить, а реально працює руками в тому ж середовищі, що й ви.

Цілі та методи дослідження

Мета роботи – виміряти приріст продуктивності. Кількісно порівняти час і зусилля, потрібні для виконання рутинних завдань, з використанням Claude Code та без нього. Проаналізувати якість результатів. Дослідити, як застосування Claude Code впливає на стабільність тестів, читабельність коду та кількість регресій.

У рамках дослідження я застосував три взаємодоповнювані підходи. По-перше, провів огляд публікацій і публічних бенчмарків: статей, блог-постів і release-нот, де Anthropic та незалежні команди демонструють роботу Claude Code; здобуті показники зіставлено з

аналогічними даними GitHub Copilot, Gemini-Code та інших AI-асистентів. По-друге, дослідили описані кейс-стаді впровадження Claude Code у командних проєктах, щоб зрозуміти, які саме завдання він автоматизує та як це позначається на економії часу та якості патчів. По-третє, здійснили порівняльний огляд оприлюднених тестових результатів – від проходження unit- й integration-тестів до оцінок читабельності коду – аби оцінити вплив Claude Code на стабільність і підтримуваність програмного забезпечення.

Результати дослідження

Claude Code доводить, що агентний підхід до програмування може суттєво змінити щоденну роботу розробника. Інструмент вбудовується у git-цикли та локально виконує команди – клонування репозиторію, збірку, запуск тестів, виправлення помилок і створення pull-request – без втручання людини. Практичні кейси показують, що такий «інженер-бот» економить десятки відсотків часу й підвищує якість коду.

- *Гіпершвидкість у продакшні.* Фінтех-платформа Ramp за перший місяць інтеграції згенерувала понад 1 млн рядків коду Claude-ом, а час розслідування інцидентів скоротився на ≈80 % завдяки автоматичному циклу build-test-fix.
- *Миттєвий код-рев'ю.* Платформа Graphite отримала 40-разове прискорення фідбеку (з 1 год до 90 с), при цьому 96 % коментарів Claude розробники оцінили як корисні, а 67 % пропозицій одразу змерджили.
- *Менше багів, більше тестів.* Sentry AutoFix на базі Claude визначає корінь помилки з 95 % точністю й пропонує готовий патч у більш ніж половині випадків; паралельно генерує unit-тест, щоб запобігти регресії.
- *Турборефакторинг і прототипи.* У сервісі Lovable навіть нетехнічні користувачі збирають веб-застосунки до 20× швидше, ніж традиційним кодингом; Claude сам шукає дубльований код, пропонує оптимізації та уніфікує стиль.

Таблиця 1.1.

Основні характеристики інструментів автоматизації програмної розробки

Інструмент	Сильні сторони	Обмеження
Claude Code	Глибокий контекст (до 200k токенів), локальне виконання CLI-команд, автоматичні PR із виправленнями, найвищий рівень схвалення рев'ю	Потребує Internet-CLI-доступу, поки що лише консольний інтерфейс
GitHub Copilot	Блискавичне автодоповнення в IDE, +26–55 % швидше шаблонне кодування	Обмежений контекст, відсутній автономний build-test-fix, потребує ручного рев'ю
Gemini-Code	Агентні дії у хмарі Google, глибока інтеграція з GCP, динамічний web-пошук	Молодший продукт; поза екосистемою Google інтеграція слабша

Ключові переваги Claude Code

1. Повний локальний цикл. Може запускати тести, лінтер, збірку й деплой безпосередньо на машині розробника.

2. Контекст + безпека. Утримує великі кодові бази в пам’яті й дотримується «ґардрейлів», мінімізуючи ризик шкідливого коду.
3. Якісний рев’юер. Дає поради рівня senior-інженера, знижує навантаження на команду й скорочує час релізу.
4. Сумісність зі стеком. Працює через CLI, тому інтегрується з будь-яким CI/CD-пайплайном чи IDE-розширенням.

Узагальнюючи, Claude Code демонструє помітне підвищення продуктивності (до 40× на рев’ю, 20× на прототипуванні) і зменшує дефекти завдяки автоматичному тестуванню та контекстному аналізу. Це робить його конкурентною перевагою для команд, яким потрібна швидка доставка якісного коду.

Висновки

Claude Code демонструє, що штучний інтелект може стати повноцінним учасником розробницького циклу, а не лише «помічником із підказками». Його здатність локально виконувати команди — від клонування репозиторію до запуску тестів і підготовки pull-requestів — переводить рутину в автоматичний режим, вивільняючи години для творчих та архітектурних завдань. Порівняння з GitHub Copilot і Gemini-Code показує: тоді як конкуренти фокусуються переважно на автодоповненні чи хмарних сценаріях, Claude Code вирізняється глибшим контекстом, високим рівнем рев’ю-якості та здатністю працювати прямо в локальному середовищі без «розриву» між IDE й CLI.

Практичні кейси (Ramp, Graphite, Sentry, Lovable) підтверджують відчутні виграші: до 40-разового прискорення рев’ю, 20-разового — прототипування, а також істотне зниження показників інцидентів завдяки автоматичному тестуванню та виправленню помилок. Усе це робить Claude Code особливо корисним для команд у швидкорухливих доменах, де час виходу на ринок критичний.

Втім, інструмент потребує уважної інтеграції: необхідні налаштовані CI/CD-пайплайни, політики безпеки і збереження контролю якості людиною, аби запобігти некоректним змінам чи «копі-пасті» коду. Надалі перспективи розвитку бачаться у графічному інтерфейсі, розширенні підтримуваних мов і ще тіснішій взаємодії з тестовими та деплой-середовищами.

Загалом Claude Code – це крок до ери, де штучний інтелект не просто допомагає писати код, а фактично бере на себе повний цикл дрібних, але трудомістких завдань. Команди, що рано адаптують такий підхід, отримують конкурентну перевагу у швидкості, якості та витратах.

Список використаних джерел:

1. Anthropic. Ramp Case Study. 2025. https://www.anthropic.com/customers/ramp
2. Anthropic. Graphite Case Study. 2024. https://www.anthropic.com/customers/graphite
3. Anthropic. Sentry AutoFix Case Study. 2025. https://www.anthropic.com/customers/sentry
4. Anthropic. Lovable Case Study. 2025. https://www.anthropic.com/customers/lovable
5. GitHub Blog. Research: quantifying GitHub Copilot’s impact on developer productivity and happiness. 2022. https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/
6. Google Cloud. Gemini Code Assist – Code Features Overview. 2025. https://cloud.google.com/gemini/docs/codeassist/code-overview

Відомості про автора:

Шумейко Євген Олександрович – здобувач вищої освіти 4-го курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук та технологій Державного університету «Київський авіаційний інститут». *Наукові інтереси*: великий обсяг даних, java, штучний інтелект.

E-mail: 6868267@stud.kai.edu.ua