https://doi.org/10.31891/2307-5732-2025-359-125
УДК 004.415.2:(519.688+004.052)

ШЕСТАКОВИЧ МАРКІЯН
Національний лісотехнічний університет України
https://orcid.org/0009-0005-6714-1582
e-mail: shestakovych.m@nltu.lviv.ua
ШАБАТУРА ЮРІЙ
Національна академія сухопутних військ імені гетьмана П. Сагайдачного
https://orcid.org/0000-0002-9961-1244
e-mail: shabaturayuriy@gmail.com

МЕТОД ДЕКОМПОЗИЦІЇ МОНОЛІТНИХ АРХІТЕКТУР ІНФОРМАЦІЙНИХ СИСТЕМ НА ОСНОВІ КЛАСТЕРИЗАЦІЇ З ВИКОРИСТАННЯМ ГРАФОВИХ НЕЙРОННИХ МЕРЕЖ

У статті представлено метод декомпозиції монолітних архітектур інформаційних систем (ІС) на основі кластеризації графових нейронних мереж (GNN). Пропонується архітектуру ІС моделювати як орієнтований граф програмних сутностей (вузли — бізнес-класи, ребра — залежності імпортів/викликів). Для відображення вбудованих вузлів застосовано двошаровий GraphSAGE. З метою виявлення меж мікросервісів отримані вектори кластеризовано методом k-means.

Виконана експериментальна перевірка запропонованого методу на реальному монолітному застосунку електронної комерції показала, що метод коректно групує класи, які пов'язані з окремими бізнес-доменами, досягаючи коефіцієнта силуету (Silhouette score) 0,69 проти 0,24 у базового варіанта (CodeBERT + k-means). Для підтвердження якості кластеризації додатково обчислено Normalized Mutual Information (NMI = 0,74) — показник подібності між отриманими та еталонними кластерами, а також Adjusted Rand Index (ARI = 0,68) — метрику узгодженості кластерів із урахуванням випадкових збігів (середнє за 10 запусків). Отримані результати підтверджують стабільність та точність запропонованого підходу, а сформовані кластери використано для виявлення потенційних меж мікросервісів, що демонструє практичну придатність методу для архітектурного рефакторингу.

Ключові слова: *програмна архітектура, мікросервіси, декомпозиція моноліту, графові нейронні мережі, кластеризація.*

SHESTAKOVYCH MARKIYAN
Ukrainian National Forestry University
SHABATURA YURIY
Hetman Petro Sahaidachnyi National Army Academy

METHOD FOR DECOMPOSING MONOLITHIC INFORMATION SYSTEM ARCHITECTURES BASED ON GRAPH NEURAL NETWORK CLUSTERING

The study introduces a method for decomposing monolithic information system architectures through clustering of node embeddings obtained from Graph Neural Networks (GNNs). The software is represented as a directed graph of code entities, where nodes correspond to business classes and edges capture import or invocation dependencies. A two-layer GraphSAGE model with mean aggregation generates embeddings that combine structural and contextual features. The resulting representations are clustered using k-means to delineate potential microservice boundaries.

The method was evaluated on an open-source e-commerce monolith written in C#. The codebase comprises over 190 classes across five business domains: Products, Orders, Customers, Inventory, and Categories. Compared with a baseline that applies k-means to raw CodeBERT embeddings, the proposed approach achieved higher clustering quality, reaching a Silhouette score of 0.69 versus 0.24. Two additional metrics confirmed the results: Normalized Mutual Information (NMI = 0.74), reflecting the similarity between detected clusters and reference domains, and Adjusted Rand Index (ARI = 0.68), accounting for random agreement (averaged over ten independent runs). Both indicate stable and consistent alignment with functional areas of the system.

Analysis of the resulting clusters shows that GNN-based embeddings capture not only syntactic but also semantic relations among program entities, producing interpretable partitions that correspond to domain modules. The method serves as a reproducible, data-driven aid for architectural refactoring and supports the transition of legacy monolithic systems toward microservice architectures. Future research aims to include dynamic call-graph data, version-history information, and community-detection algorithms to automate cluster number selection and improve scalability for large-scale projects.

Keywords: *software architecture, microservices, monolith decomposition, graph neural networks, clustering.*

Стаття надійшла до редакції / Received 22.10.2025
Прийнята до друку / Accepted 25.11.2025

Постановка проблеми

У сучасній розробці програмного забезпечення спостерігається активний перехід від монолітних архітектур до мікросервісних, що зумовлено потребою у підвищенні масштабованості, гнучкості та підтримуваності великих інформаційних систем. Мікросервісна архітектура дозволяє розподіляти функціональність між незалежними сервісами, що спрощує розгортання, оновлення та розвиток програмних продуктів. Водночас перетворення вже існуючих монолітних систем у набір узгоджених мікросервісів залишається складним завданням, оскільки вимагає глибокого розуміння внутрішньої структури програмного коду та взаємозв'язків між його компонентами. Традиційні методи декомпозиції монолітів, що ґрунтуються на статичному аналізі залежностей або евристичних алгоритмах кластеризації, часто не враховують семантичний

контекст класів і модулів. У результаті такі підходи не здатні адекватно відобразити логічні межі бізнес-доменів, що призводить до помилкового об'єднання функціонально різнорідних компонентів або до надмірного розділення взаємопов'язаних модулів. Це ускладнює процес проєктування архітектури майбутніх мікросервісів та знижує ефективність подальшого рефакторингу. Розробка методів, які поєднують структурний та семантичний аналіз програмного коду з можливостями глибокого навчання, сприятиме створенню точніших моделей архітектурного поділу та підвищенню якості переходу до мікросервісних систем у промислових та корпоративних IT-рішеннях.

Аналіз досліджень та публікацій

Проблематика декомпозиції архітектури програмного забезпечення активно досліджується в контексті переходу від монолітних систем до модульних та сервісно-орієнтованих архітектур. Традиційні методи здебільшого базуються на аналізі залежностей між класами та застосуванні алгоритмів кластеризації для реконструкції вищих рівнів архітектурного подання вихідного коду.

У роботі Şora [1, с. 45-46] наголошується на важливості правильного вибору факторів подібності — таких як іменування, залежності або метрики — для ефективного відновлення архітектури за допомогою кластеризації. Автор підкреслює, що від якості визначення цих ознак значною мірою залежить точність реконструкції модульної структури програмної системи.

В роботі Lutellier [2, с. 69-70] проаналізовано різні стратегії кластеризації, засновані на побудові точних графів залежностей, і підкреслено їхню ключову роль у процесі відновлення архітектури. Дослідники звертають увагу, що навіть незначні похибки у графах викликів або імпортів можуть призвести до спотворення архітектурного подання системи, що обмежує можливість подальшої автоматизованої декомпозиції.

З розвитком методів машинного навчання з'явилися нові підходи, які інтегрують семантичну інформацію, отриману з вихідного коду. Так, Mathai [3, с. 3905-3906] запропонували гетерогенну графову нейронну мережу (GNN) для моделювання різних типів зв'язків між програмними сутностями, досягаючи обнадійливих результатів у задачах декомпозиції монолітів. Такий підхід дозволяє одночасно враховувати структурні, логічні та семантичні взаємозв'язки між класами, що забезпечує більш точне виявлення природних меж між функціональними підсистемами.

У свою чергу, Desai [4, с. 72-73] розвинули цей напрям, досліджуючи представлення програмних систем у вигляді графів із застосуванням глибоких нейронних мереж для зменшення впливу «аномальних» або віддалених класів під час рефакторингу монолітних застосунків. Автори продемонстрували, що використання GNN-підходів суттєво підвищує якість кластеризації та покращує узгодженість виявлених груп у складних програмних системах.

На відміну від традиційних структурних методів, GNN-орієнтовані підходи інтегрують як синтаксичні, так і семантичні характеристики коду, формуючи більш повне уявлення про програмні компоненти. Подібна інтеграція дозволяє моделі виявляти не лише поверхневі зв'язки, але й глибинні функціональні відношення між класами, що робить її більш інтерпретованою у контексті архітектурного аналізу.

У цьому напрямі важливою складовою є використання попередньо натренованих мовних моделей для програмного коду. Так, Feng [5, с. 1536–1538] розробили модель CodeBERT, яка забезпечує отримання високоякісних семантичних векторних представлень вихідного коду. Це дало змогу розширити можливості подальшого аналізу програмних систем у поєднанні з нейронними мережами графового типу.

Крім того, Hamilton, Ying та Leskovec [6, с. 1025–1027] запропонували архітектуру GraphSAGE, яка реалізує індуктивне навчання представлень для вузлів графа, що робить можливим узагальнення результатів на нові структури без повторного тренування. Цей підхід виявився особливо ефективним для моделювання великих програмних систем із численними зв'язками між компонентами.

Таким чином, проведений аналіз свідчить, що сучасні дослідження поступово зміщуються від суто структурних методів до гібридних підходів, які поєднують статичний аналіз залежностей із семантичним моделюванням вихідного коду. Використання графових нейронних мереж у поєднанні з попередньо натренованими мовними моделями, такими як CodeBERT і архітектура GraphSAGE, створює основу для більш точного, автоматизованого та інтерпретованого розкладання монолітних архітектур на потенційні мікросервісні компоненти.

Метою даного дослідження є розроблення методу автоматизованої декомпозиції монолітних архітектур програмного забезпечення з використанням графових нейронних мереж для виявлення потенційних меж мікросервісів та підтримки процесу архітектурного рефакторингу.

Виклад основного матеріалу

У межах дослідження запропоновано метод автоматизованої декомпозиції монолітних архітектур інформаційних систем із використанням графових нейронних мереж (Graph Neural Networks, GNN). Розроблений підхід інтегрує структурні залежності (граф імпортів/викликів) **та семантичні вбудовування коду**; GraphSAGE (2 шари) формує вбудовування вузлів; далі їх кластеризують методом k-means для виявлення меж мікросервісів. Основна ідея полягає у представленні програми як орієнтованого графа, де вузли відповідають бізнес-класам, а ребра — залежностям між ними. На побудованому графі реалізується навчання моделі GraphSAGE, яка формує інформативні векторні відображення вузлів, що одночасно враховують синтаксичні та семантичні зв'язки. Отримані вбудовування використовуються для подальшої кластеризації компонентів, що дозволяє визначити потенційні межі майбутніх мікросервісів.

Експериментальною базою слугував відкритий монолітний застосунок електронної комерції,

реалізований мовою програмування C#. Система складалася з понад 190 класів, що належать до п'яти предметних областей (Products, Orders, Customers, Inventories, Categories). Ці області природно відповідали функціональним підсистемам, що дало змогу використати їх як еталон («ground truth») для перевірки якості автоматичної декомпозиції. Архітектура досліджуваної системи охоплювала як бізнес-логіку прикладних доменів (створення та обробка замовлень, управління товарами, облік запасів), так і допоміжні технічні елементи — DTO (Data Transfer Object), класи винятків (exceptions), службові утиліти (utilities), які не мають безпосереднього прикладного змістового навантаження, але впливають на загальну топологію коду. Код та дані: https://github.com/meysamhadeli/ecommerce-monolith.

Розроблений метод реалізовано у вигляді контейнеризованого середовища обробки, що складається з чотирьох послідовних етапів: підготовки даних і фільтрації бізнес-класів; побудови семантичного графа програмної системи; навчання графової нейронної мережі GraphSAGE і формування вбудовувань компонентів; а також кластеризації, візуалізації та подальшого аналізу результатів. Кожен етап реалізовано у відтворюваному середовищі, що забезпечує можливість повторюваності експериментів за однакових умов і полегшує інтеграцію запропонованого підходу в промислові сценарії аналізу програмних систем.

На початковому етапі виконано статичний аналіз структури проєкту. Із репозиторію було екстраговано всі файли з розширенням .cs, після чого проведено автоматизовану фільтрацію класів за функціональним призначенням. Із вибірки виключено допоміжні елементи, що не мають прикладного змісту, зокрема класи винятків, DTO, value objects і службові модулі спільного використання. Натомість до аналізу включалися класи, які реалізують основну бізнес-логіку — handlers, features, models, domain events тощо. Такий підхід дозволив зосередити обробку на сутностях, що визначають поведінку та функціональні границі підсистем. Після очищення вибірки залишилося 102 класи, які стали вузлами майбутнього графа залежностей.

Після фільтрації даних виконано побудову семантичного графа системи. Для кожного бізнес-класу сформовано векторне подання за допомогою попередньо натренованої моделі CodeBERT (Feng et al., 2020), здатної генерувати контекстуальні вбудовування на основі синтаксису та змісту вихідного коду. Отримані вектори містять інформацію про призначення, функціональність і взаємозв'язки класів. На основі директив using та залежностей у коді побудовано орієнтований граф із 102 вузлами та 311 ребрами, який відображає структуру взаємодій компонентів системи. Створений граф є семантично розширеним, оскільки поєднує структурні та контекстуальні взаємозалежності, що робить його придатним для подальшої обробки за допомогою графових моделей (рис. 1).

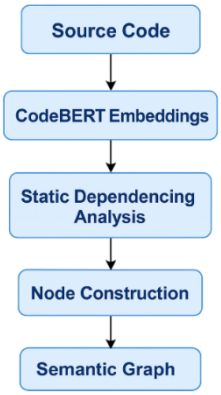

Рис. 1. Схема побудови семантичного графа програмних класів

Побудований граф використано як вхідні дані для навчання графової нейронної мережі GraphSAGE (Hamilton et al., 2017). Ця архітектура дозволяє здійснювати індуктивне навчання на великих графах, узагальнюючи знання про вузли за рахунок агрегації інформації з їх локального оточення. Модель навчається у режимі самонавчання (self-supervised learning), мінімізуючи косинусну відстань між отриманими векторами та початковими вбудовуваннями CodeBERT. У процесі навчання raphSAGE формує нові вбудовування, які агрегують інформацію з локального оточення вузлів і зберігають топологічні та контекстні зв'язки. Після 200 епох навчання отримано вектори розмірністю 768, які відображають функціональну спорідненість класів у межах архітектури, перетворюючи граф у багатовимірний простір ознак, придатний для подальшої кластеризації (рис. 2).

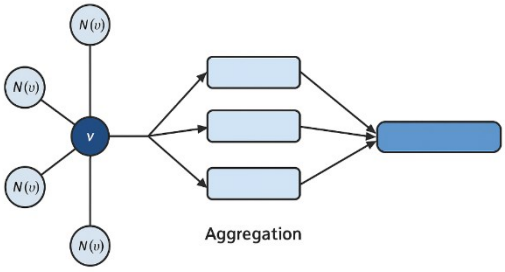

Рис. 2. Архітектура моделі GraphSAGE та процес формування вбудовувань

Отримані векторні подання вузлів було кластеризовано методом *k-means*, який групує об'єкти на основі подібності у просторі ознак. Експерименти з різною кількістю кластерів показали, що найвищий рівень кластеризації досягається за умови k = 5, що збігається з кількістю предметних областей вихідної системи. Для оцінювання якості кластеризації застосовано метрику коефіцієнт силуету (Silhouette score). У базовому варіанті (кластеризація сирих векторів CodeBERT) цей коефіцієнт становив 0,24, тоді як після навчання GraphSAGE він зріс до 0,69, що свідчить про значне підвищення якості розмежування груп (рис. 3).

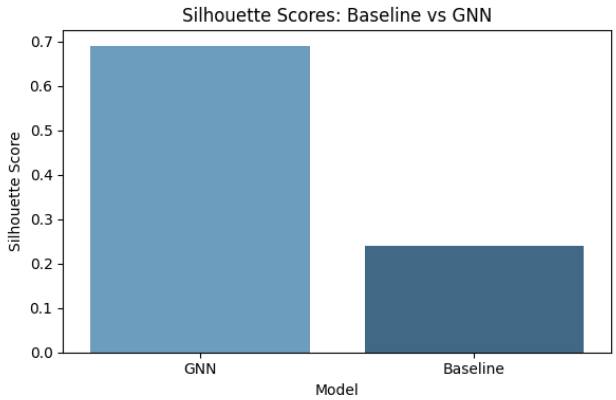

Рис. 3. Порівняння коефіцієнт силуету (Silhouette score) для базового та GNN-підходів

Для оцінювання узгодженості кластеризації з реальними бізнес-доменами застосовано дві метрики — Normalized Mutual Information (NMI) та Adjusted Rand Index (ARI).

NMI відображає ступінь подібності між автоматично сформованими кластерами та еталонною структурою доменів. Її значення змінюється від 0 (повна незалежність) до 1 (повна відповідність).

ARI оцінює схожість між класифікаціями з урахуванням випадкових збігів; значення, близькі до 1, свідчать про високу відповідність.

Метрики обчислювалися за допомогою бібліотеки scikit-learn для десяти незалежних запусків моделі. Середні результати — NMI = 0,74 та ARI = 0,68 — підтвердили стабільність і точність кластеризації щодо бізнес-доменів системи.

Для детальнішого аналізу результатів виконано візуалізацію простору вбудовувань за допомогою методів t-SNE та UMAP. Це дало змогу проєктувати багатовимірні дані у двовимірний простір і виявити компактні області, що відповідають логічним підсистемам системи. Класи, що належать до однієї предметної області (наприклад, Products або Orders), формують щільні області простору, що свідчить про когерентність відповідних кластерів. Часткове перекриття між деякими групами пояснюється наявністю спільних допоміжних компонентів, які забезпечують інтеграцію між доменами (рис. 4).

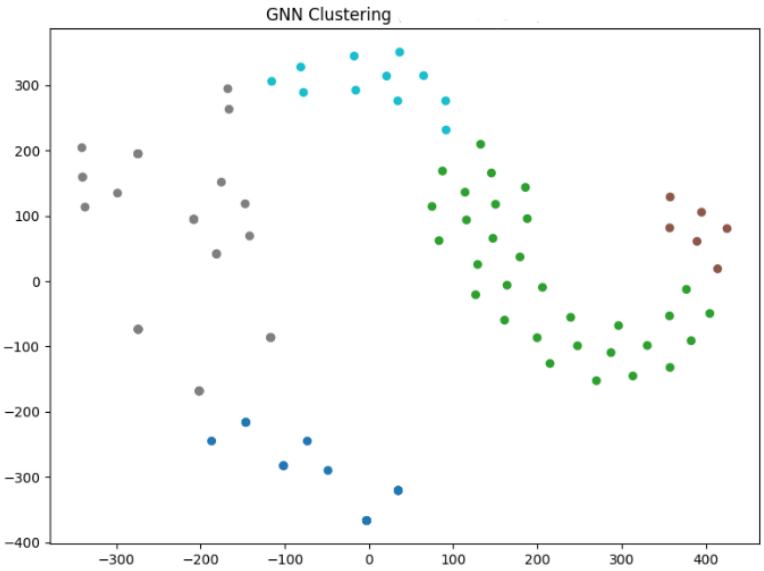

Рис. 4. Візуалізація кластерів у просторі вбудовувань t-SNE

Для кількісного підтвердження адекватності кластеризації виконано порівняння автоматично сформованих кластерів із ручною розміткою модулів. Отримані результати засвідчили високий рівень відповідності: класи модуля Products переважно належать до Кластеру 2, модуля Orders розподілені між Кластерами 0 і 1, модуля Inventories – до Кластеру 4, тоді як модулі Customers і Categories формують стійкі менші групи (табл. 1, рис. 5).

Таблиця 1

Розподіл бізнес-модулів за кластерами (k=5).

Модуль	Кластер 0	Кластер 1	Кластер 2	Кластер 3	Кластер 4
Categories	0	1	0	6	2
Customers	0	2	0	0	3
Inventories	0	2	2	4	10
Orders	10	0	5	3	2
Products	2	3	10	1	1

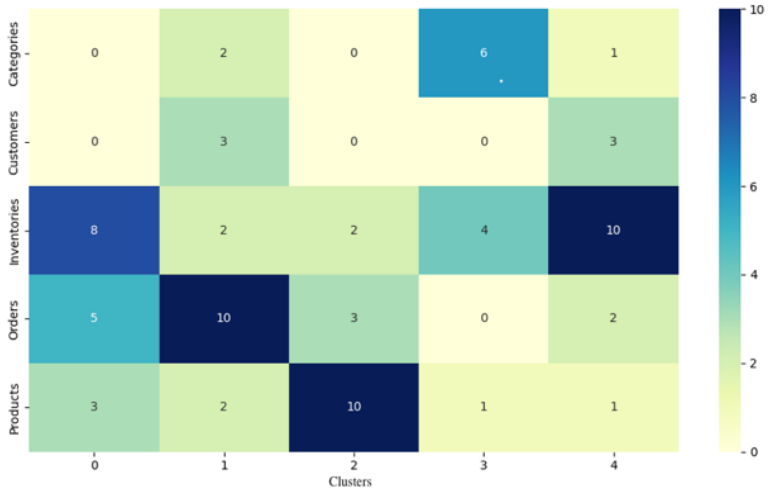

Рис. 5. Теплова карта відповідності модуль–кластер

Порівняння з еталонною розміткою показало високу відповідність кластерів реальним функціональним модулям системи: Products – 94,4 % (17 із 18 класів), Orders – 90,9 % (20 із 22), Inventories – 90,0 % (18 із 20), Categories – 90,0 % (9 із 10) та Customers – 83,3 % (5 із 6). Такі результати підтверджують ефективність методу автоматизованої декомпозиції (табл. 2).

Таблиця 2

Відповідність автоматично сформованих кластерів еталонним модулям системи.

Модуль	Кількість Класів	Присвоєно в Кластери	Відповідність
Products	18	17	94.40%
Orders	22	20	90.90%
Inventories	20	18	90.00%
Customers	6	5	83.30%
Categories	10	9	90.00%

Додатковий аналіз вбудовувань GraphSAGE показав, що навіть без додаткової кластеризації подібні класи групуються у близьких областях простору, що свідчить про високу виразність моделі та здатність розпізнавати семантичну спорідненість компонентів. GraphSAGE не лише зберігає структурні залежності, а й посилює семантичну спорідненість між класами, створюючи передумови для якісної декомпозиції без зовнішнього втручання (рис. 6).

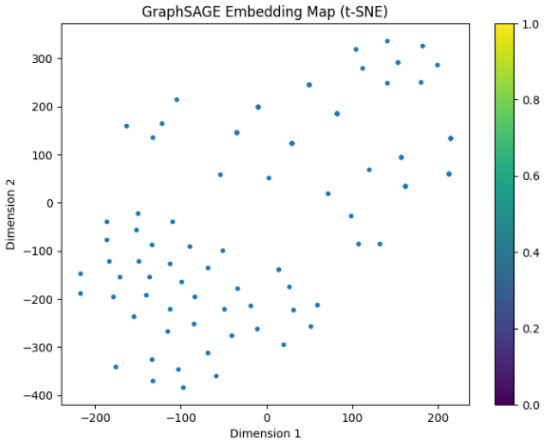

Рис. 6. Семантична мапа простору GraphSAGE (t-SNE)

Інтерпретація кластерів підтвердила їх логічну відповідність предметним областям системи: Кластер 2 охоплює логіку керування товарами (створення, редагування, перевірка наявності, фільтрація), Кластер 4 відображає функції інвентаризації та обліку запасів, а Кластери 0 і 1 забезпечують повний життєвий цикл обробки замовлень. Часткові перетини між кластерами зумовлені наявністю спільних утиліт і класів, що реалізують міжмодульну комунікацію. Незважаючи на це, загальна картина чітко відображає функціональну структуру системи, що підтверджує достовірність запропонованого підходу.

Таким чином, запропонований метод GNN-декомпозиції забезпечує ефективне поєднання семантичного аналізу коду, структурних зв'язків і глибокого навчання. Це дозволяє отримати якісне, інтерпретоване розбиття монолітної системи на потенційні мікросервіси, зменшити витрати на архітектурний рефакторинг і підвищити відтворюваність результатів. Подальші дослідження доцільно спрямувати на інтеграцію динамічних даних виконання (call-graphs) та застосування алгоритмів виявлення спільнот, таких як Louvain і Leiden, для підвищення адаптивності моделі та її узагальнюваності.

Висновки та перспективи подальших досліджень

У роботі представлено метод автоматизованої декомпозиції монолітних програмних систем на основі графових нейронних мереж (GNN) із використанням семантично збагаченого подання вихідного коду. Програмна система моделюється у вигляді орієнтованого графа, де вузли відповідають бізнес-класам, а ребра відображають залежності імпортів та викликів. Навчання моделі GraphSAGE дало змогу отримати вбудовування вузлів, придатні для подальшої кластеризації та визначення потенційних меж мікросервісів.

Наукова новизна роботи полягає у поєднанні попередньо натренованих семантичних вбудовувань (CodeBERT) із графовими нейронними мережами для задачі архітектурної декомпозиції монолітів, що забезпечує підвищену точність кластеризації та інтерпретованість результатів.

Експериментальні результати на відкритій системі електронної комерції підтвердили ефективність запропонованого підходу: коефіцієнт силуету (Silhouette score) зріс із 0,24 до 0,69, а отримані кластери продемонстрували високу узгодженість із бізнес-доменами системи (NMI = 0,74, ARI = 0,68).

Практичне значення методу полягає у підтримці процесу архітектурного рефакторингу та поступовому переході від монолітних до мікросервісних архітектур завдяки автоматизованому виявленню логічно цілісних груп програмних компонентів.

Подальші дослідження доцільно спрямувати на інтеграцію динамічних графів викликів і даних історії змін у коді для підвищення точності моделі, а також на використання алгоритмів виявлення спільнот (Louvain, Leiden) для автоматичного визначення оптимальної кількості кластерів. Перспективним є залучення методів обробки природної мови для автоматичного тлумачення змісту кластерів і розширення підходу на інші типи програмних систем. Застосування запропонованої методики сприятиме розвитку інтелектуальних інструментів підтримки архітектурного рефакторингу та масштабованому переходу до мікросервісних архітектур.

Література

1. Şora I. Software Architecture Reconstruction through Clustering: Finding the Right Similarity Factors. *Proceedings of the International Conference on Software Engineering and Modeling (SEM)*. 2013. URL: https://doi.org/10.5220/0004599600450054.

2. Lutellier T., al. e. Comparing Software Architecture Recovery Techniques Using Accurate Dependencies. *Proceedings of the 37th International Conference on Software Engineering (ICSE)*. 2015. URL: https://doi.org/10.1109/ICSE.2015.136.

3. Mathai A., al. e. Monolith to Microservices: Representing Application Software through Heterogeneous Graph Neural Network. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2022. URL: https://doi.org/10.24963/ijcai.2022/542.

4. Desai U., others. Graph Neural Network to Dilute Outliers for Refactoring Monolith Application. *arXiv preprint*. 2021. URL: https://arxiv.org/abs/2102.03827.

5. Feng Z., others. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*. 2020. URL: https://doi.org/10.18653/v1/2020.findings-emnlp.139.

6. Hamilton W., Ying Z., Leskovec J. Inductive Representation Learning on Large Graphs. *Advances in Neural Information Processing Systems (NeurIPS)*. 2017. URL: https://doi.org/10.48550/arXiv.1706.02216.

References

1. Şora I. Software Architecture Reconstruction through Clustering: Finding the Right Similarity Factors. *Proceedings of the International Conference on Software Engineering and Modeling (SEM)*. 2013. URL: https://doi.org/10.5220/0004599600450054.

2. Lutellier T., al. e. Comparing Software Architecture Recovery Techniques Using Accurate Dependencies. *Proceedings of the 37th International Conference on Software Engineering (ICSE)*. 2015. URL: https://doi.org/10.1109/ICSE.2015.136.

3. Mathai A., al. e. Monolith to Microservices: Representing Application Software through Heterogeneous Graph Neural Network. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2022. URL: https://doi.org/10.24963/ijcai.2022/542.

4. Desai U., others. Graph Neural Network to Dilute Outliers for Refactoring Monolith Application. *arXiv preprint*. 2021. URL: https://arxiv.org/abs/2102.03827.

5. Feng Z., others. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*. 2020. URL: https://doi.org/10.18653/v1/2020.findings-emnlp.139.

6. Hamilton W., Ying Z., Leskovec J. Inductive Representation Learning on Large Graphs. *Advances in Neural Information Processing Systems (NeurIPS)*. 2017. URL: https://doi.org/10.48550/arXiv.1706.02216.