ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Система автоматизованої інтеграції програмного забезпечення на основі семантичного аналізу та машинного навчання»

на здобуття освітнього ступеня магістр
за спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Юрій ДОБРУШИН
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група ІСДМ-61 Юрій ДОБРУШИН
(ім'я, ПРІЗВИЩЕ)
Керівник Віктор САГАЙДАК
PhD *(ім'я, ПРІЗВИЩЕ)*

Рецензент: Наталія ЛАЩЕВСЬКА
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти магістр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК

"____" _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Добрушин Юрій Вікторович
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Система автоматизованої інтеграції програмного забезпечення на основі семантичного аналізу та машинного навчання
керівник кваліфікаційної роботи: Віктор САГАЙДАК, PhD
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від "___ " жовтня 2025 р. № ____

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
1. Специфікації OpenAPI (Swagger) та стандарти REST API.
2. Документація архітектурних патернів Enterprise Integration Patterns.
3. Технічна документація великих мовних моделей.
4. Науково-технічна література з проблем семантичної інтероперабельності.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1. Теоретико-методичні основи інтеграції інформаційних систем та аналіз еволюції Middleware.
2. Розробка методів семантичного аналізу та алгоритмів мапінгу API на основі машинного навчання.
3. Програмна реалізація системи Intelligent Middleware та експериментальне дослідження ефективності.

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання «___» _____ 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір та аналіз науково-технічної літератури за темою інтеграції та LLM	01.09.2025 – 15.09.2025	Виконано
2.	Системний аналіз предметної області, вибір архітектурних рішень	16.09.2025 – 30.09.2025	Виконано
3.	Розробка методів семантичного аналізу та алгоритмів мапінгу	01.10.2025 – 20.10.2025	Виконано
4.	Програмна реалізація компонентів системи	21.10.2025 – 25.11.2025	Виконано
5.	Проведення експериментів, аналіз результатів та формулювання висновків	26.11.2025 – 05.12.2025	Виконано
6.	Розробка демонстраційних матеріалів, підготовка доповіді	06.12.2025 – 12.12.2025	Виконано
7.	Оформлення магістерської роботи, проходження нормоконтролю	13.12.2025 – 23.12.2025	Виконано

Здобувач вищої освіти _____ Юрій ДОБРУШИН
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Віктор САГАЙДАК
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступня магістр: 83 стор., 24 Рисунок,5 табл., 31 джерел.

Мета роботи – підвищення ефективності процесів інтеграції гетерогенних програмних систем шляхом розробки інтелектуального посередника, що використовує методи семантичного аналізу та машинного навчання.

Об'єкт дослідження – процес інтеграції гетерогенних програмних компонентів у розподілених інформаційних системах.

Предмет дослідження – методи та алгоритми семантичного аналізу метаданих API та використання великих мовних моделей для автоматизації мапінгу даних.

Короткий зміст роботи. У роботі проведено аналіз проблем семантичної інтероперабельності в мікросервісних архітектурах. Розроблено гібридний метод визначення відповідності полів API, що поєднує евристичні алгоритми та генеративний штучний інтелект. Створено програмний комплекс Intelligent Middleware, який автоматизує побудову інтеграційних сценаріїв. Експериментально підтверджено, що запропонований підхід забезпечує точність мапінгу 94.5% та скорочує час налаштування інтеграцій.

КЛЮЧОВІ СЛОВА: ІНТЕГРАЦІЯ, API, OPENAPI, LLM, МАШИННЕ НАВЧАННЯ, SEMANTIC MAPPING, MIDDLEWARE, МІКРОСЕРВІСИ.

ABSTRACT

The text part of the qualifying work for obtaining a master's degree: 83 pp., 24 fig., 5 tables, 31 sources.

The purpose of the work is to increase the efficiency of heterogeneous software systems integration by developing an intelligent middleware based on semantic analysis and machine learning methods.

Object of research – the process of integration of heterogeneous software components in distributed information systems.

Subject of research – methods and algorithms of API metadata semantic analysis and usage of large language models for data mapping automation.

Summary of the work: The thesis analyzes the problems of semantic interoperability in microservice architectures. A hybrid method for API field matching combining heuristic algorithms and generative artificial intelligence has been developed. The Intelligent Middleware software complex for automating integration scenarios has been created. Experimental results confirmed that the proposed approach provides 94.5% mapping accuracy and reduces integration setup time.

KEYWORDS: INTEGRATION, API, OPENAPI, LLM, MACHINE LEARNING, SEMANTIC MAPPING, MIDDLEWARE, MICROSERVICES.

ЗМІСТ

ВСТУП 9
1 ТЕОРЕТИКО-МЕТОДИЧНІ ОСНОВИ ІНТЕГРАЦІЇ ІНФОРМАЦІЙНИХ СИСТЕМ 12
1.1 Сучасні підходи до архітектури корпоративних додатків та проблеми інтеграції 12
1.2 Аналіз патернів інтеграції та еволюція Middleware 16
1.3 Огляд методів автоматизованого мапінгу схем 19
1.4 Аналіз існуючих рішень та обґрунтування необхідності власної розробки 22
1.5 Порівняльний аналіз сучасних великих мовних моделей для задач генерації коду та мапінгу даних 24
1.6 Математичні та концептуальні основи векторного подання слів та оцінки подібності 26
1.7 Проблеми інформаційної безпеки та захисту даних при використанні генеративного ШІ в корпоративних інтеграціях 29
1.8 Нормативно-правове регулювання використання систем штучного інтелекту в інтеграційних процесах 30
2 РОЗРОБКА МЕТОДІВ СЕМАНТИЧНОГО АНАЛІЗУ ТА ІНТЕГРАЦІЇ API 34
2.1 Системний аналіз та формалізація процесу семантичної інтеграції 34
2.2 Архітектурне проєктування програмного комплексу та моделювання даних 36
2.3 Алгоритм пошуку відповідностей (Matching Algorithm) 39
2.4. Математичне забезпечення оцінки семантичної близькості 42
2.5. Алгоритмічне забезпечення валідації результатів та обробки колізій 44
3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ 47
3.1 Технологічні аспекти реалізації компонентів системи інтелектуальної інтеграції 47
3.2 Забезпечення інформаційної безпеки та валідація даних 49
3.3 Програмна реалізація алгоритмів та логіки роботи системи 51
3.4 Методика та результати експериментального дослідження 59
3.5 Сценарій експлуатації системи адміністратором інтеграції 69
3.6 Інфраструктурне забезпечення та автоматизація розгортання 75
ВИСНОВКИ 78
ПЕРЕЛІК ПОСИЛАНЬ 81

ВСТУП

Актуальність теми. Парадигмальний зсув у інженерії програмного забезпечення, зумовлений переходом від монолітних архітектур до розподілених мікросервісних екосистем, актуалізував проблему інтероперабельності компонентів на якісно новому рівні. Незважаючи на повсюдну імплементацію стандартизованих транспортних протоколів (HTTP/REST) та форматів серіалізації даних (JSON, Protobuf), індустрія стикнулася з феноменом «семантичного розриву». Сутність проблеми полягає у дисонансі між синтаксичною сумісністю інтерфейсів та відсутністю єдиної онтологічної бази: незалежні команди розробників оперують тотожними поняттями предметної області, проте кодують їх у гетерогенних структурах даних. В умовах експоненційного зростання кількості точок інтеграції (API Sprawl), традиційні методи ручного проєктування адаптерів стають вузьким місцем (bottleneck), що критично збільшує операційні витрати та гальмує цикл виведення нових продуктів на ринок. Отже, розробка автоматизованих методів семантичного узгодження схем даних, базованих на гібридному поєднанні детермінованих алгоритмів та ймовірнісних моделей штучного інтелекту, є не просто оптимізаційним, а необхідним архітектурним рішенням для забезпечення масштабованості сучасних Enterprise-систем.

Зв’язок роботи з науковими програмами, планами, темами. Кваліфікаційна робота виконана відповідно до плану науково-дослідних робіт кафедри інформаційних систем та технологій Державного університету інформаційно-комунікаційних технологій та корелює з пріоритетним науковим напрямом «Інформаційні системи та технології» в частині розробки інтелектуальних методів обробки даних.

Мета і завдання дослідження. Метою роботи є підвищення ефективності інтеграційних процесів у гетерогенних розподілених системах шляхом розробки методології та програмних засобів автоматизованого семантичного мапінгу специфікацій API. Досягнення поставленої мети реалізується через вирішення комплексу взаємопов'язаних завдань. Насамперед проведено системний аналіз архітектурних обмежень існуючих патернів інтеграції (ESB, iPaaS) та виявлено

недоліки методів синтаксичного порівняння схем. На основі цього розроблено вдосконалений метод визначення семантики полів, який, на відміну від існуючих аналогів, базується на зваженій комбінації лексичних евристик та контекстного аналізу великими мовними моделями. Наступним кроком спроєктовано алгоритм пошуку відповідностей на двочасткових графах для генерації топологічно коректних пропозицій інтеграції. Практична реалізація дослідження втілена у розробці архітектури та програмного коду системи Intelligent Middleware. Фінальним етапом стала експериментальна верифікація ефективності запропонованих рішень на репрезентативному наборі відкритих специфікацій.

Об’єкт дослідження – процеси інформаційної взаємодії та інтеграції компонентів у розподілених програмних системах.

Предмет дослідження – методи семантичного аналізу метаданих прикладних інтерфейсів та алгоритми машинного навчання для автоматизації задач узгодження схем даних (Schema Matching).

Методи дослідження. Методологічну основу роботи складає синтез теоретичних та емпіричних методів. Системний аналіз застосовано для декомпозиції предметної області та формалізації вимог до інтелектуального посередника. Математичний апарат теорії множин та графів використано для моделювання простору ознак полів API та побудови алгоритмів мапінгу. Методи дистрибутивної семантики та векторної алгебри (Vector Space Models) залучено для розрахунку мір подібності атрибутів. Елементи теорії ймовірностей та математичної статистики використано при оцінці точності класифікації та аналізі експериментальних даних. Програмна реалізація виконана з використанням об'єктно-орієнтованого підходу.

Наукова новизна одержаних результатів. У дисертації отримано нові наукові результати, що полягають у комплексному вирішенні задачі семантичної інтероперабельності.

По-перше, удосконалено метод семантичної типізації полів OpenAPI, який, на відміну від класичних детермінованих підходів, використовує гібридну модель аналізу (Regex + LLM), що дозволяє ідентифікувати бізнес-сутність атрибутів в

умовах полісемії та неявної термінології.

По-друге, набув подальшого розвитку підхід до оцінки сумісності інтерфейсів шляхом введення інтегрального показника впевненості та матриці сумісності типів, що дозволило автоматизувати процес ранжування гіпотез мапінгу та зменшити простір пошуку рішень.

По-третє, вперше запропоновано архітектуру інтелектуального Middleware із вбудованим механізмом Prompt Engineering для валідації JSON-структур, що забезпечує стійкість системи до стохастичної природи генеративних моделей та дозволяє інтегрувати їх у детерміновані бізнес-процеси.

Практичне значення одержаних результатів. Розроблений програмний комплекс, реалізований на базі стеку Python/FastAPI, є готовим інструментальним засобом для автоматизації рутинних задач системних архітекторів. Впровадження системи дозволяє скоротити часові витрати на етап проєктування інтеграції (Design-time) та мінімізувати ризики помилок, пов'язаних із людським фактором ("Human Error"). Модульна архітектура рішення дозволяє використовувати його компоненти (зокрема сервіс SuggestionService) як незалежні мікросервіси у складі CI/CD-конвеєрів.

Апробація результатів. Основні теоретичні положення та практичні результати дослідження пройшли апробацію на профільних наукових заходах, зокрема доповідалися на III Всеукраїнській науково-технічній конференції «Технологічні горизонти: дослідження та застосування інформаційних технологій» (Київ, 18 листопада 2025 р.) та VIII Всеукраїнській науково-технічній конференції «Комп'ютерні технології: інновації, проблеми, рішення» (Житомир, 02–03 грудня 2025 р.).

1 ТЕОРЕТИКО-МЕТОДИЧНІ ОСНОВИ ІНТЕГРАЦІЇ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Сучасні підходи до архітектури корпоративних додатків та проблеми інтеграції

Фундаментальна трансформація принципів побудови програмних комплексів, зумовлена вимогами концепції Industrie 4.0, диктує необхідність переходу від монолітних конструкцій до гнучких, децентралізованих рішень. Еволюція архітектурних патернів, зміщуючи фокус від сервіс-орієнтованої архітектури SOA [5] до мікросервісної MSA [6], трансформує завдання інтеграції з площини транспортування даних у площину забезпечення семантичної інтероперабельності. Зазначена тенденція прослідковується через аналіз компромісів, що виникають при зміні архітектурних стилів (Рисунок 1.1).

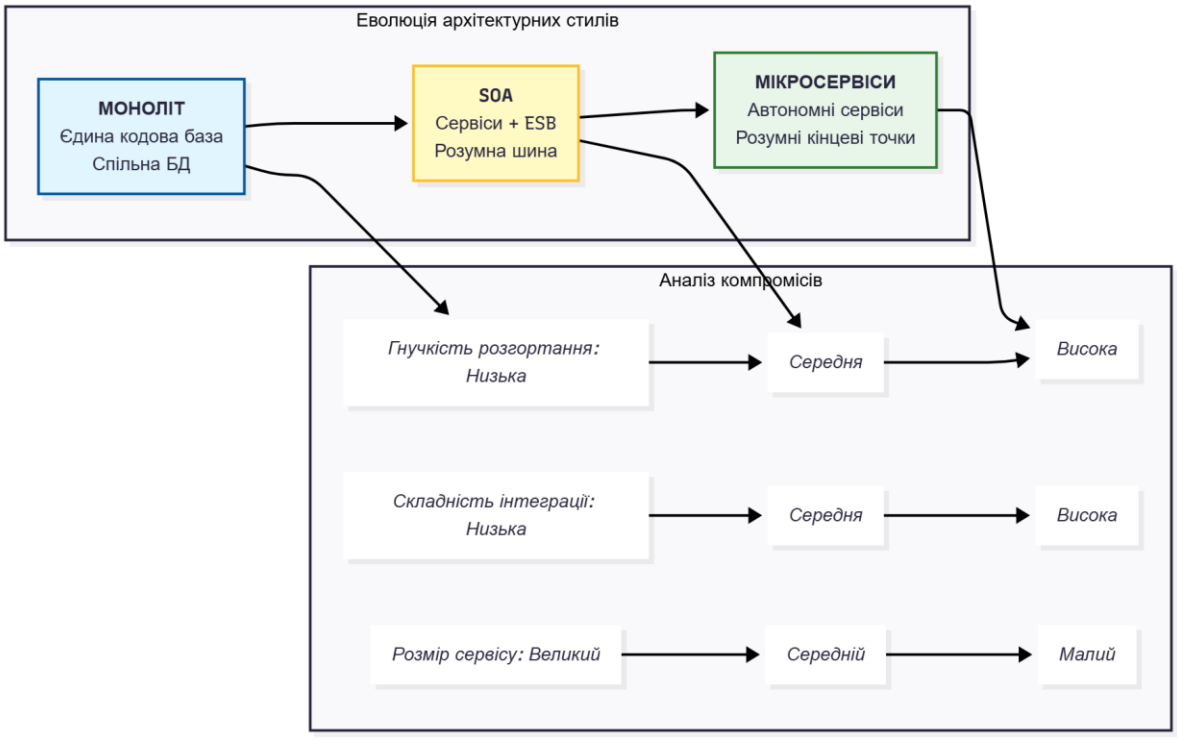


Рисунок 1.1 – Аналіз компромісів при еволюції архітектурних стилів

Ретроспективний аналіз свідчить, що домінування монолітної архітектури, де інтерфейс, бізнес-логіка та шар даних функціонували в межах єдиного

розгортаємого артефакту, було виправданим на етапах з помірною складністю систем.

Взаємодія через локальні виклики функцій гарантувала відсутність мережевих затримок, проте зростання кодової бази призводило до неможливості гранулярного масштабування та критичної залежності компонентів. Нівелювання цих недоліків відбулося з впровадженням SOA, де декомпозиція системи на окремі сервіси поєднувалася з використанням корпоративної сервісної шини Enterprise Service Bus [7]. Шина ESB централізувала маршрутизацію та трансформацію протоколів, здебільшого базуючись на стандартах SOAP та XML. Втім, жорсткість контрактів WSDL та концентрація логіки в шині часто створювали ефект «вузького місця», що гальмувало процес розробки.

Сучасною відповіддю на потребу в автономності стала мікросервісна архітектура, яка доводить ідею децентралізації до логічного завершення через патерн Database per Service та формування незалежних команд розробки.

В умовах так званої «економіки API» комунікація реалізується переважно через архітектурний стиль REST з використанням протоколу HTTP та формату JSON. Спрощення синтаксичної взаємодії порівняно з SOAP призвело до виникнення нових викликів, пов'язаних із втратою суворого контролю схеми даних. Відсутність формального опису або його десинхронізація з кодом спричиняє ситуацію, коли сервіси оперують ідентичними поняттями предметної області, але використовують відмінні структури даних.

Детальний аналіз предметної області дає підстави виокремити низку критичних проблем, що залишаються невирішеними в рамках стандартних підходів. Насамперед спостерігається семантична гетерогенність даних, коли різні мікросервіси використовують неузгоджені формати для опису одних і тих самих бізнес-сутностей, що вимагає перманентної трансформації на кожному етапі взаємодії.

Це явище неминуче призводить до проблеми дублювання логіки та появи значного обсягу адаптаційного коду, відомого як glue code, що прямо порушує принцип Don't Repeat Yourself та розпорошує ресурси розробників. Ситуацію

ускладнює крихкість інтеграційних зв'язків, оскільки зміна специфікації API одного компонента здатна спровокувати каскадні збої в залежних модулях. Враховуючи нелінійне зростання кількості точок інтеграції у розподілених системах, складність підтримки узгодженості контрактів зростає експоненціально.

Порівняльна характеристика архітектурних стилів, наведена в таблиці 1.1, демонструє зміщення акцентів від централізованого управління до автономності компонентів.

Таблиця 1.1

Порівняльна характеристика архітектурних стилів

Характеристика	Моноліт	SOA (Service Oriented Architecture)	Мікросервіси (MSA)
Зв'язаність	Висока	Середня (через шину ESB)	Низька (слабка зв'язаність через API)
Протоколи	Виклики функцій / IPC	SOAP, XML, Messaging (JMS)	REST (JSON), gRPC, GraphQL, AMQP
Дані	Єдина спільна БД	Спільні або розподілені БД	Database per Service (ізольовані дані)
Інтеграція	Не потрібна (всередині)	"Smart pipes, dumb endpoints" (розумна шина)	"Smart endpoints, dumb pipes" (розумні кінцеві точки)

Продовження Таблиці 1.1

Масштабування	Тільки дублювання всього додатку	Окремі сервіси, але залежність від ESB	Гранулярне (окремі контейнери/функції)
Складність	Низька на старті, висока з часом	Висока (Enterprise рівень)	Висока (експлуатаційна складність, DevOps)

Опираючись на наведені дані, доцільно констатувати безальтернативність переходу до мікросервісів для високонавантажених систем. Проте відмова від концепції розумної шини на користь розумних кінцевих точок покладає відповідальність за інтерпретацію даних безпосередньо на сервіси. За відсутності єдиного семантичного стандарту виникає семантичний розрив – дисонанс між формальним інтерфейсом та реальним змістом даних. Подолання зазначеного розриву вимагає впровадження інтелектуальних систем middleware, здатних до автоматичного аналізу контексту, чому і присвячена дана робота.

Історичний контекст методів інтеграції дозволяє простежити еволюцію від найпростішого обміну файлами через FTP, що породжував високу латентність та проблеми синхронізації, до технологій розподілених об'єктів на кшталт CORBA та DCOM. Останні, запровадивши мови опису інтерфейсів IDL, виявилися надмірно складними в експлуатації.

Якщо ера Shared Database вирішувала питання швидкості ціною жорсткої зв'язаності, то сучасні підходи Intelligent Middleware повинні інтегрувати шар штучного інтелекту для автоматизації процесів, які раніше вимагали ручної конфігурації мапінгів та жорсткого кодування правил.

1.2 Аналіз патернів інтеграції та еволюція Middleware

Забезпечення інтероперабельності в гетерогенних розподілених середовищах неможливе без наявності формалізованого, машиночитаного контракту, який виступає єдиним джерелом істини для взаємодіючих компонентів. Якщо в парадигмі сервіс-орієнтованої архітектури цю функцію монопольно виконував стандарт WSDL, то перехід до архітектурного стилю REST оголив проблему відсутності суворої типізації, сформувавши запит на більш гнучкі мови опису прикладних інтерфейсів. На поточному етапі індустрія оперує рядом конкуруючих специфікацій, кожна з яких пропонує відмінні підходи до моделювання ресурсів та графів даних. Отже, розробка системи автоматизованої інтеграції вимагає прагматичного вибору базового стандарту, здатного забезпечити баланс між семантичною виразністю та придатністю до алгоритмічної обробки.

Беззаперечним лідером у сфері опису RESTful API є специфікація OpenAPI, еволюція якої від ранніх версій Swagger [8] до сучасних ревізій відображає зміну вимог індустрії до точності контрактів. Розгляд цього стандарту доцільно проводити не хронологічно, а через призму зростання його семантичної потужності. Якщо версія 2.0 обмежувалася примітивною підмножиною JSON Schema Draft 4, що ускладнювало валідацію складних структур, то реліз 3.0 впровадив критично важливі для автоматизації механізми oneOf та anyOf. Ці конструкції дозволили описувати поліморфні дані, що є необхідною умовою для точного мапінгу об'єктів у динамічно типізованих мовах. Вершиною цього розвитку стала імплементація повної сумісності з JSON Schema 2020-12 у версії 3.1, що фактично стандартизувало словник метаданих і відкрило можливості для використання універсальних валідаторів без необхідності написання кастомних парсерів.

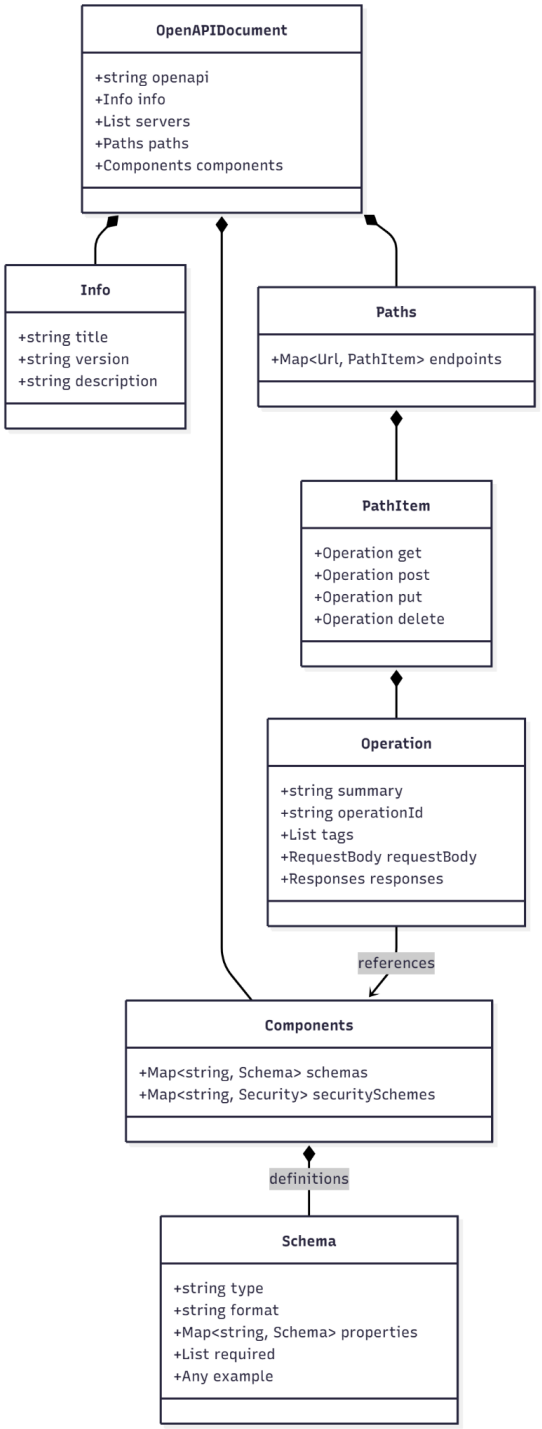


Рисунок 1.2 – Структурна схема специфікації OpenAPI

Поза межами екосистеми OpenAPI існують альтернативні підходи, аналіз яких необхідний для повноти дослідження. Специфікація RAML, базуючись на форматі YAML, фокусується на ієрархічному моделюванні ресурсів та механізмах успадкування через трейти. Хоча такий підхід підвищує читабельність для людини, він суттєво ускладнює програмний аналіз через необхідність розгортання

(resolving) глибоких ланцюжків залежностей перед безпосередньою обробкою. На противагу цьому, стандарт API Blueprint сповідує філософію Design-First з акцентом на документацію, проте його недостатня структурна жорсткість робить його малопридатним для автоматичної генерації клієнтського коду та навчання моделей машинного навчання.

Окремий кластер утворюють стандарти для асинхронної взаємодії, де де-факто стандартом стає AsyncAPI [9]. Будучи адаптацією синтаксису OpenAPI для подійно-орієнтованих архітектур та брокерів повідомлень, він дозволяє уніфікувати алгоритмічну базу парсингу. Враховуючи структурну спорідненість цих форматів, методи семантичного аналізу, розроблені для синхронних REST-взаємодій, можуть бути з мінімальними модифікаціями екстрапольовані на асинхронні канали зв'язку.

Вибір оптимального формату для цілей даного дослідження базувався на комплексному аналізі за критеріями виразності типізації, підтримки схем JSON та наявності метаданих для NLP-аналізу.

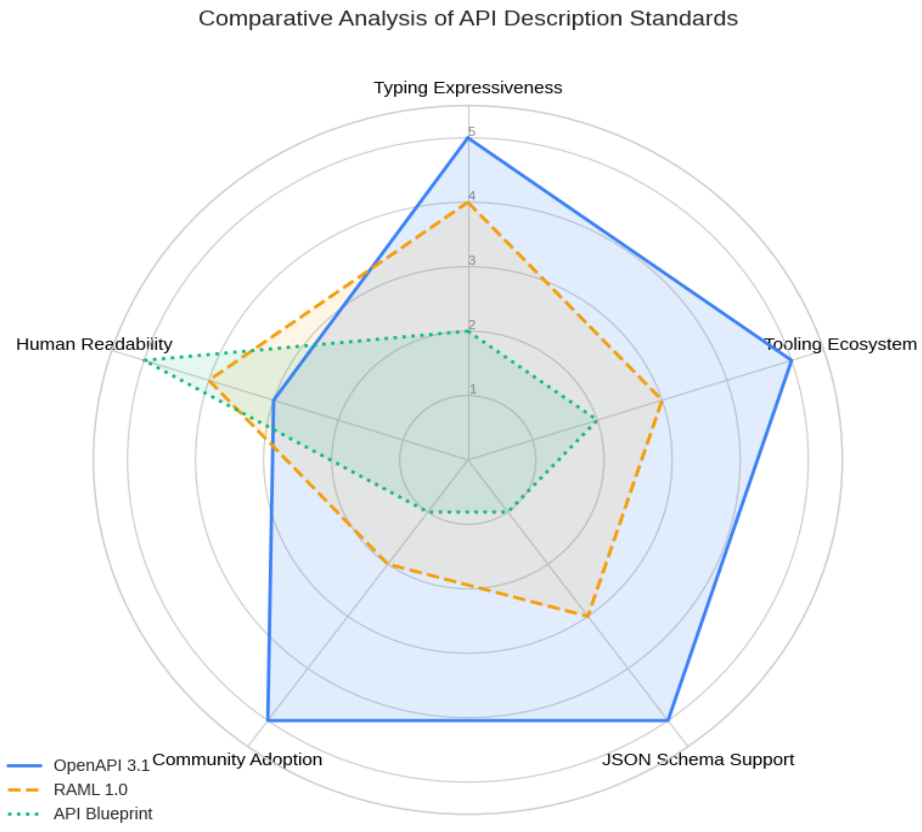


Рисунок 1.3 – Порівняльна діаграма характеристик стандартів API

За результатами порівняльного аналізу можна стверджувати, що OpenAPI є найбільш релевантним кандидатом для побудови системи інтелектуальної інтеграції. Вирішальним фактором тут виступає не стільки поширеність стандарту, скільки його жорстка схема, яка дозволяє застосовувати детерміновані алгоритми для вилучення структури даних. Більше того, обов'язкова наявність полів description та summary у специфікації створює необхідний контекст для роботи алгоритмів обробки природної мови, дозволяючи моделі "розуміти" призначення полів, а не лише їх технічний тип. Підтримка застаріваючих стандартів на кшталт RAML вимагала б розробки складних проміжних конвертерів, що видається недоцільним у рамках даної роботи. Таким чином, подальша розробка фокусуватиметься на аналізі специфікацій сімейства OpenAPI з потенціалом масштабування на AsyncAPI.

1.3 Огляд методів автоматизованого мапінгу схем

Проблема встановлення релевантних відповідностей між атрибутами гетерогенних схем даних історично вирішувалася у площині лексичного аналізу. Класичні алгоритми, що базуються на метриках редакторської відстані (Левенштейна, Джаро-Вінклера та ін.) [10, 11], демонструють прийнятну ефективність виключно при ідентифікації морфологічних варіацій або помилок введення. Однак, в умовах реальних інтеграційних задач суто синтаксичний підхід виявляє свою неспроможність. Критичним обмеженням стає нездатність лексичних метрик працювати з явищами синонімії та полісемії: семантично тотожні атрибути client_id та customer_uuid мають нульову лексичну близькість, що робить їх нерозрізнюваними для традиційних алгоритмів. Подолання цього бар'єру вимагає зміни парадигми – переходу від порівняння символьних літералів до аналізу векторних представлень змісту [12].

Підготовка метаданих API до семантичного аналізу не обмежується тривіальною токенізацією. Враховуючи специфіку naming conventions у

програмному коді (camelCase, snake_case), першочерговим завданням стає сегментація складених ідентифікаторів та їх подальша нормалізація через лематизацію. Це дозволяє зменшити розмірність словника ознак і привести специфічну термінологію до загальновживаних мовних конструкцій.

Фундаментом сучасних методів мапінгу виступає трансформація текстових даних у багатовимірний векторний простір, де мірою схожості виступає не збіг символів, а косинусна відстань між векторами [13].

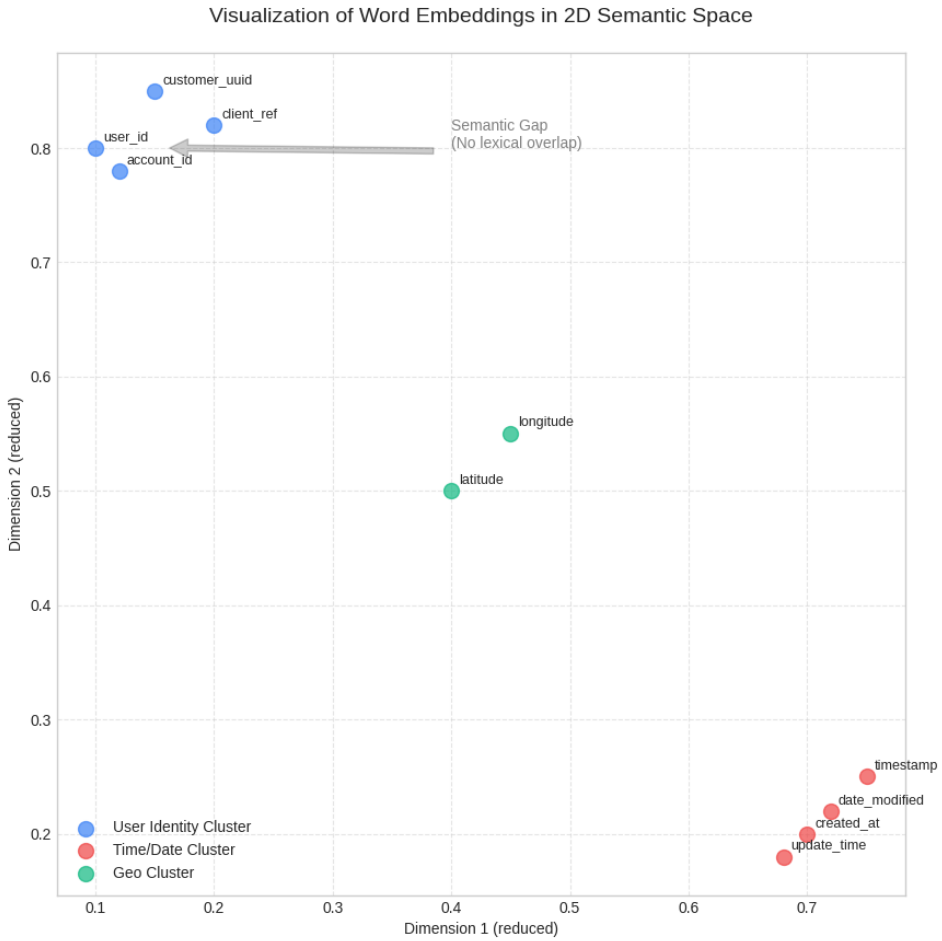


Рисунок 1.4 – Порівняння семантичного простору: Лексичний збіг vs Векторна близькість

Втім, використання статичних ембеддінгів, що генерують фіксований вектор для лексеми, виявляється недостатнім для розв'язання задач із високою контекстуальною залежністю. Слово date може означати як календарну дату, так і тип фрукта, і без урахування оточення статичний вектор буде усередненим, а отже

– неточним. Вирішенням цієї проблеми стало впровадження архітектури Transformer та механізму Self-Attention [14]. Здатність трансформерів моделювати залежності між усіма елементами послідовності одночасно дозволила генерувати динамічні, контекстуальні ембеддінги.

Якісний стрибок у задачах Schema Matching пов'язаний із появою великих мовних моделей (LLM). Представники сімейств GPT-4 або Llama 3, володіючи здатністю до Zero-shot learning [15], фактично усувають потребу у зборі розмічених датасетів для навчання специфічних класифікаторів полів. Більше того, "емерджентні" властивості таких моделей дозволяють ідентифікувати імпліцитні зв'язки між атрибутами, наприклад, розпізнавати пару полів lat / lon як єдину сутність "Географічні координати" без явних вказівок у схемі [16].

\

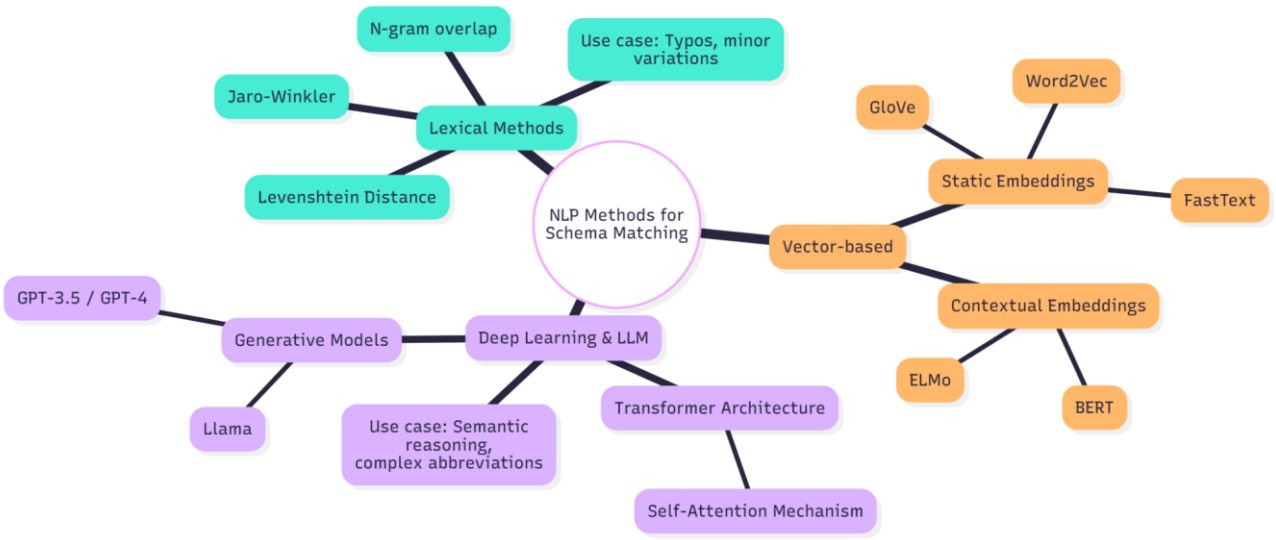


Рисунок 1.5 – Діаграма класифікації методів

Разом з тим, використання важковагових LLM для кожного порівняння є економічно та обчислювально неефективним. Тому в рамках даної роботи обґрунтовано застосування гібридного підходу. Система первинно фільтрує очевидні відповідності за допомогою швидких евристик та легкорвагових векторних моделей, і лише у випадках високої невизначеності (амбігуації) делегує задачу генеративним моделям. Така архітектура дозволяє досягти балансу між точністю семантичного аналізу та швидкістю роботи Middleware.

1.4 Аналіз існуючих рішень та обґрунтування необхідності власної розробки

Технологічний ландшафт засобів інтеграції доцільно структурувати за критерієм «рівень абстракції – модель обробки даних», що дозволяє провести чітку демаркацію між трьома домінуючими класами рішень: платформами iPaaS, шлюзами API (API Gateways) та ETL-інструментарієм. Критичний розгляд архітектурних патернів представників цих груп виявляє системні обмеження, що унеможливлюють повну автоматизацію інтеграційних процесів без участі людини.

У сегменті Integration Platform as a Service (iPaaS) спостерігається виражена дихотомія. З одного боку, рішення класу low-code (на кшталт Zapier) пропонують закриту архітектуру black-box, яка, спрощуючи вхідний поріг, унеможливлює реалізацію нелінійної логіки трансформації та стає економічно неефективною при масштабуванні транзакційного навантаження. З іншого боку, платформи рівня Enterprise (MuleSoft) надають вичерпний інструментарій для маніпуляцій даними (мови на зразок DataWeave), проте вимагають високої інженерної кваліфікації та значних капіталовкладень. Спільним знаменником для обох полюсів залишається парадигма пасивного виконання: система делегує інженеру рутинну задачу семантичного зіставлення атрибутів джерела та приймача, не пропонуючи механізмів інтелектуального аналізу вмісту.

Функціональний профіль рішень класу API Gateway (Kong, Tyk, Apigee) фокусується на ортогональних задачах: управлінні трафіком, автентифікації та тротлінгу. Реалізуючи патерн «Smart Pipes», ці системи забезпечують надійний транзит даних, проте оперують ними як непрозорим бітовим потоком, ігноруючи внутрішню семантику payload’у. Аналогічне обмеження притаманне і ETL-інструментам (Talend, Airbyte). Останні, будучи оптимізованими для пакетної обробки великих масивів (Batch Processing) при завантаженні у сховища даних (DWH), демонструють неприйнятну латентність для сценаріїв синхронної взаємодії мікросервісів у реальному часі.

Синтез результатів аналізу увиразнює технологічний вакуум між інфраструктурними рішеннями, що забезпечують транспорт, та візуальними

редакторами, які лише полегшують ручну працю архітектора. Ринок фактично не пропонує рішень класу Intelligent Middleware – систем, здатних взяти на себе когнітивне навантаження щодо інтерпретації специфікацій API.

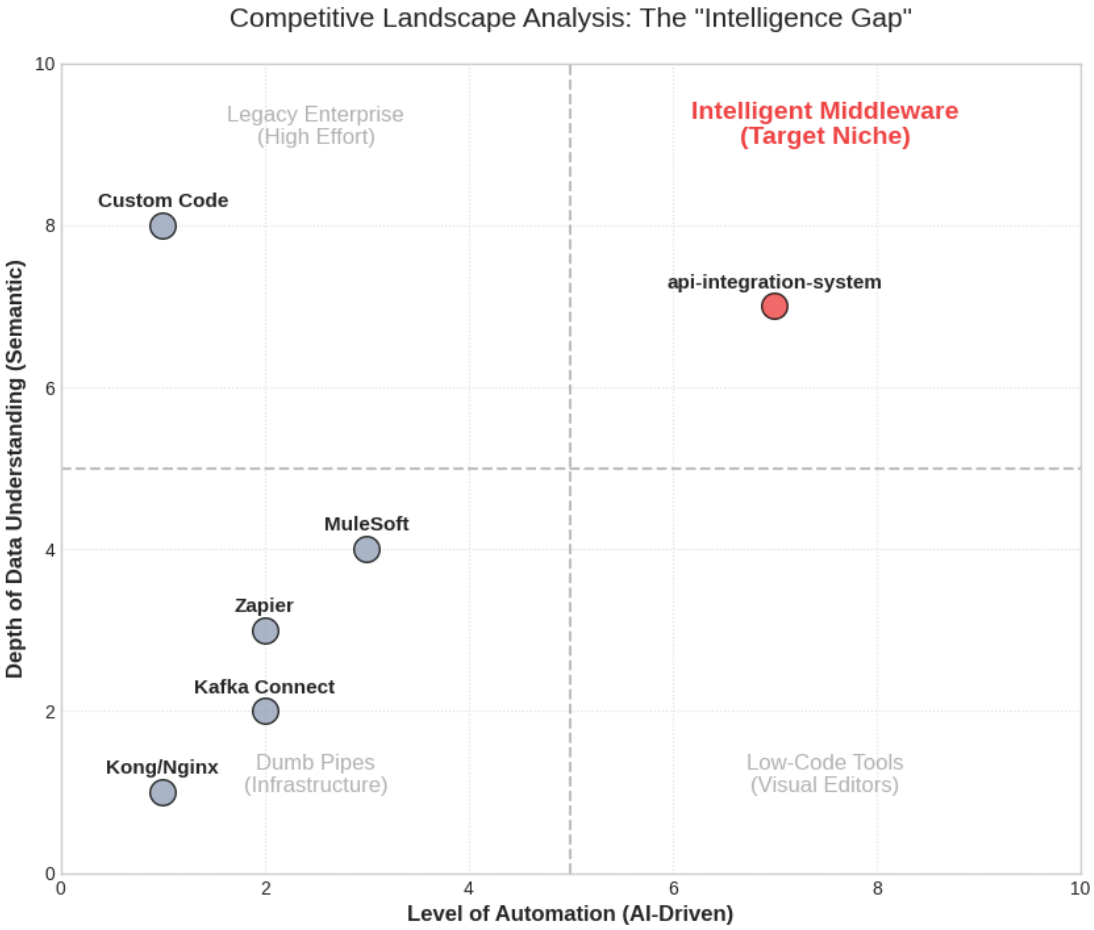


Рисунок 1.6 – Матриця позиціонування інтеграційних рішень

Розроблювана система позиціонується саме в цій ніші. Відмова від концепції візуального редактора (drag-and-drop), притаманної класичним iPaaS, на користь генеративного підходу дозволяє реалізувати парадигму Architecture-as-Code. Застосування методів NLP та великих мовних моделей для автоматичного аналізу контрактів API нівелює необхідність написання адаптаційного glue code. Система не намагається замінити високопродуктивні шлюзи, а виступає як інтелектуальна надбудова (control plane), що автоматизує створення конфігурацій мапінгу, скорочуючи час розгортання інтеграцій з годин до хвилин.

1.5 Порівняльний аналіз сучасних великих мовних моделей для задач генерації коду та мапінгу даних

Валідність функціонування системи автоматизованої інтеграції детермінується спроможністю базової генеративної моделі забезпечувати сувору відповідність вихідних даних формальним схемам (JSON Schema, OpenAPI) при збереженні глибини семантичної інтерпретації вхідного контексту. Вибір ядра семантичного аналізатора вимагає проведення компаративного аналізу провідних архітектур – GPT-4, Claude 3, Llama 3 та CodeLlama – через призму специфічних вимог задачі Schema Matching.

Матриця критеріїв відбору базується на трьох ортогональних векторах. По-перше, здатність до логічних міркувань (Reasoning), що визначає точність резолюції неоднозначностей у назвах ідентифікаторів (наприклад, асоціація поля field_x45 із сутністю transaction_id). По-друге, обсяг контекстного вікна, що лімітує розмірність специфікації API, доступної для одномоментного аналізу. По-третє, структурний детермінізм – здатність моделі генерувати валідний синтаксис JSON без галюцинацій та сторонніх текстових включень. Економічна складова (вартість токена) та експлуатаційна латентність виступають обмежувальними факторами.

Таблиця 1.2

Порівняльний аналіз LLM для задач генерації схем даних

Характеристика	GPT-4o / GPT-4 Turbo	Claude 3 Opus	Llama 3 (70B)	CodeLlama
Розробник	OpenAI	Anthropic	Meta AI	Meta AI
Тип доступу	API (Closed Source)	API (Closed Source)	Open Source (Weights)	Open Source (Weights)
Вікно контексту	128k токенів	200k токенів	8k / 8192 токенів	до 100k токенів

Продовження табл. 1.2

Характеристика	GPT-4o / GPT-4 Turbo	Claude 3 Opus	Llama 3 (70B)	CodeLlama
Якість кодогенерації	Дуже висока	Дуже висока	Висока	Середня/Висока
Підтримка JSON Mode	Нативна	Висока	Вимагає граматик	Низька
Швидкість (Inference)	Висока (особливо GPT-4o)	Середня	Залежить від GPU	Висока
Вартість (орієнтовна)	~$5 / 1M input tokens	~$15 / 1M input tokens	Витрати на хостинг (GPU)	Витрати на хостинг

Аналіз пропрієтарного сегменту виявляє жорстку конкуренцію між архітектурами GPT-4o (OpenAI) та Claude 3 Opus (Anthropic). Модель Claude 3, володіючи контекстним вікном у 200k токенів, технічно переважає конкурентів при роботі з монолітними специфікаціями корпоративних систем. Проте, вища латентність та вартість інференсу роблять її субоптимальною для задач реального часу. Натомість, GPT-4o демонструє критичну для даної роботи перевагу – нативну підтримку режиму JSON Object та механізму Function Calling. Ця архітектурна особливість гарантує повернення даних у валідному форматі, що нівелює необхідність розробки складних регулярних виразів для пост-обробки та очищення відповіді від "розмовного" шуму.

В сегменті Open Source рішень, зокрема Llama 3 (Meta), ключовим аргументом є можливість локального розгортання (On-Premise), що забезпечує суверенітет даних – критичну вимогу для регульованих індустрій (FinTech, HealthTech). Однак експериментальні дані свідчать про те, що без застосування

спеціалізованих технік (grammar-constrained decoding) відкриті моделі частіше порушують складні схеми виводу порівняно з пропрієтарними аналогами SOTA-рівня. Спеціалізовані ж моделі на кшталт CodeLlama, будучи оптимізованими на корпусах програмного коду, демонструють деградацію якості при роботі з природно-мовною семантикою описів полів.

Підсумовуючи результати аналізу, для імплементації модуля семантичного мапінгу в даній роботі обрано модель GPT-4o. Рішення продиктоване балансом між точністю Zero-shot класифікації та надійністю структурного виводу. Використання хмарного API з гарантованим форматом JSON дозволяє мінімізувати кодову базу адаптерів та зосередитись на алгоритмічній логіці інтеграції, замість адміністрування власного кластера GPU, необхідного для хостингу моделей рівня Llama 3 70B.

1.6 Математичні та концептуальні основи векторного подання слів та оцінки подібності

Ефективність функціонування підсистеми семантичного аналізу Intelligent Middleware перебуває у прямій залежності від спроможності застосованих алгоритмів здійснювати коректне відображення дискретного простору термінів природної мови у неперервний векторний простір ознак. Формалізація цього процесу вимагає розгляду математичних моделей, що лежать в основі перетворення текстових описів полів API у машиночитаний формат.

На початковому етапі розвитку методів інформаційного пошуку для оцінки важливості терміна в контексті документа застосовувалася статистична міра TF-IDF (Term Frequency-Inverse Document Frequency). Математично вага $w_{i,j}$ терміна t_i у документі або описі поля, d_j визначається як добуток локальної частоти на обернену частоту документа:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right) \tag{1.1}$$

де:

$tf_{i,j}$ – частота входження терміна t_i у документ d_j;

N – загальна кількість документів у корпусі (наприклад, загальна кількість полів у специфікації API);

df_i – кількість документів, що містять термін t_i.

Хоча цей підхід дозволяє нівелювати вплив загальновживаних стоп-слів, до наприкладу артиклів "the" або "a", він генерує розріджені вектори високої розмірності, ортогональність яких унеможливлює виявлення семантичних зв'язків між синонімами. Для подолання цього обмеження у роботі застосовуються методи дистрибутивної семантики, що базуються на гіпотезі, згідно з якою слова, що зустрічаються у схожих контекстах, мають близькі векторні представлення.

У сучасних нейромережевих архітектурах, зокрема Word2Vec, задача навчання векторних представлень зводиться до максимізації ймовірності передбачення контекстних слів. Для моделі Skip-gram ймовірність появи контекстного слова w_c за умови наявності центрального слова w_t обчислюється за допомогою функції Softmax:

	$$P(w_c \mid w_t) = \frac{\exp\left(\boldsymbol{v'_{w_c}} \cdot \boldsymbol{v_{w_t}}\right)}{\sum_{w=1}^{V} \exp\left(\boldsymbol{v'_{w}} \cdot \boldsymbol{v_{w_t}}\right)}$$	(1.2)

де:

v_{w_t} – вхідний вектор центрального слова;

v'_{w_c} – вихідний вектор контекстного слова;

V – розмір словника.

У результаті оптимізації цієї функції формується щільний векторний простір R^n, де семантична близькість понять корелює з геометричною відстанню між їхніми векторами.

Критичним етапом роботи системи інтеграції є кількісна оцінка ступеня подібності між атрибутами двох різних схем даних. Враховуючи, що розмірність векторів ознак n може сягати значень 768 або 1536, саме для моделей GPT, застосування евклідової відстані стає неефективним через феномен "прокляття розмірності". Крім того, евклідова метрика є чутливою до абсолютної довжини вектора, що є неприпустимим в умовах варіативності довжини описів полів API.

Тому як основну метрику близькості обрано косинусну подібність (Cosine Similarity), яка визначається як косинус кута між двома ненульовими векторами A та B:

	$$\text{similarity}(\boldsymbol{A}, \boldsymbol{B}) = \cos(\theta) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{\lvert\boldsymbol{A}\rvert\lvert\boldsymbol{B}\rvert} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$	(1.3)

де:

$A \cdot B$ – скалярний добуток векторів;

$|A|$та$|B|$ – евклідові норми (довжини) векторів.

Інваріантність цієї метрики до масштабування дозволяє системі ідентифікувати семантичну тотожність між лаконічною назвою поля (наприклад, id) та його розгорнутим описом, оскільки вектори цих сутностей будуть колінеарними у просторі ознак.

Для вирішення проблеми полісемії (багатозначності термінів залежно від контексту) у роботі використовуються динамічні ембеддінги, що генеруються моделями на базі архітектури Transformer. Фундаментальною інновацією, що забезпечує контекстуалізацію, є механізм Scaled Dot-Product Attention ("Увага з масштабованим скалярним добутком"). Вхідна матриця ембеддінгів трансформується у три матриці: Запити Q , Ключі K та Значення V. Математично операція обчислення уваги описується формулою:

	$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$	(1.4)

де:

Q – матриця запитів (Queries), що представляє поточні токени;

K – матриця ключів (Keys), з якою порівнюються запити;

V – матриця значень (Values), що містить інформаційний контент;

d_k – розмірність векторів ключів.

Ділення на $\sqrt{d_k}$ є критично важливим масштабуючим фактором. При великих значеннях d_k скалярні добутки можуть зростати до великих величин, що зміщує функцію Softmax у область з малим градієнтом, сповільнюючи навчання.

Масштабування нівелює цей ефект.

Саме цей механізм дозволяє моделі встановлювати залежності між віддаленими елементами специфікації (Long-Range Dependencies), асоціюючи атрибути вкладених об'єктів JSON з визначеннями сутностей у корені документа, що є неможливим при використанні класичних рекурентних мереж (RNN) або статичних векторів.

1.7 Проблеми інформаційної безпеки та захисту даних при використанні генеративного ШІ в корпоративних інтеграціях

Інтеграція стохастичних генеративних моделей у контур корпоративного Middleware трансформує модель загроз, нівелюючи ефективність класичних периметральних засобів захисту. Оскільки розроблювана система функціонує як шлюз для специфікацій API та фрагментів транзакційних даних, питання інформаційної безпеки виходять за межі стандартного шифрування каналів зв'язку, зміщуючись у площину захисту від когнітивних маніпуляцій та витоків через механізми інференсу.

Першочерговим фактором ризику виступає потенційна компрометація конфіденційності при взаємодії з хмарними провайдерами LLM. Передача специфікацій, що містять опис внутрішньої топології баз даних або приклади payload’ів, створює загрозу їхнього збереження в логах провайдера або використання для донавчання глобальних моделей. Навіть за наявності юридичних гарантій в рамках Enterprise-підписок, залишається ненульова ймовірність атак через сторонні канали (side-channel attacks) або перехоплення трафіку в точці термінації SSL. Для мінімізації цього ризику в системі імплементовано стратегію Privacy by Design, ключовим елементом якої є попередня деперсоніфікація (санітизація) вхідних даних. Алгоритм автоматично детектує та обфускує чутливі лексеми, замінюючи реальні значення на синтетичні токени перед відправкою запиту до нейромережі, що гарантує аналіз виключно абстрактної структури інтерфейсу.

Згідно з класифікацією OWASP Top 10 for LLM [17], домінантним вектором атаки на інтелектуальні системи виступає Prompt Injection (LLM-01). Специфіка даної вразливості полягає у змішуванні інструкцій управління та невірених вхідних даних у єдиному контекстному вікні моделі. В умовах автоматизованої інтеграції ця загроза набуває форми як прямих, так і непрямих ін'єкцій. Пряма атака реалізується через маніпуляцію назвами полів, коли зловмисник впроваджує в метадані лінгвістичні конструкції, здатні переписати системний промпт (наприклад, техніки ігнорування контексту). Значно критичнішою для систем Middleware є непряма ін'єкція (Indirect Prompt Injection), коли шкідливий код приховано в зовнішніх ресурсах – специфікаціях OpenAPI сторонніх вендорів. Обробка отруєного файлу документації може призвести до генерації мапінгу, який непомітно для адміністратора перенаправляє потоки фінансових даних на підконтрольні зловмиснику реквізити.

Окремий клас загроз пов'язаний із порушенням цілісності вихідних даних. Атаки типу JSON Smuggling та Model Hallucination спрямовані на генерацію синтаксично коректних, але семантично шкідливих об'єктів конфігурації. Для протидії цьому вектору архітектура системи базується на принципі Zero Trust: жоден згенерований артефакт не вважається довіреним за замовчуванням. Усі результати інференсу підлягають суворій валідації на відповідність JSON Schema. У критичних сценаріях застосовується механізм Human-in-the-loop, що блокує автоматичне застосування змін без явного акцепту інженером. Комплексний підхід Defense-in-Depth, що включає ізоляцію середовища виконання та лімітування привілеїв доступу до мережі, дозволяє забезпечити стійкість інтеграційного процесу навіть в умовах недетермінованої поведінки базової моделі ШІ.

1.8 Нормативно-правове регулювання використання систем штучного інтелекту в інтеграційних процесах

Впровадження програмних комплексів класу Intelligent Middleware, архітектурне ядро яких базується на використанні стохастичних великих мовних

моделей (LLM) для семантичної обробки корпоративних даних, вимагає проведення глибокого системного аналізу на предмет відповідності чинному законодавству. Проблема ускладнюється екстериторіальним характером сучасних нормативів: глобалізація цифрових ринків та євроінтеграційний вектор розвитку України зумовлюють необхідність імплементації вимог не лише національного законодавства, але й наднаціональних регуляторних актів ЄС, які де-факто встановлюють світові стандарти. До таких належать Загальний регламент захисту даних (GDPR) та Закон ЄС про штучний інтелект (EU AI Act). Ігнорування цих норм створює для розробників та інтеграторів критичні юридичні ризики, починаючи від значних фінансових санкцій і закінчуючи повною забороною на використання програмного продукту.

Фундаментальним аспектом правового аналізу є легітимність обробки даних у контексті GDPR. Специфіка інтеграції API полягає в тому, що технічна документація (OpenAPI/Swagger), яка є об'єктом аналізу, часто містить фрагменти реальних даних у полях example або default. Ці фрагменти можуть включати унікальні ідентифікатори (UUID), адреси електронної пошти, IP-адреси або фінансові реквізити, що підпадає під визначення персональних даних (PII – Personally Identifiable Information). Передача таких несанітизованих специфікацій до хмарного провайдера LLM (наприклад, на сервери OpenAI у США) класифікується як транскордонна передача даних третім особам. Згідно зі статтею 44 GDPR, така передача можлива лише за наявності належних гарантій захисту. Враховуючи скасування угоди Privacy Shield, автоматизована відправка даних без попередньої обробки є прямим порушенням регламенту.

Для забезпечення комплаєнсу (відповідності вимогам) у архітектуру розроблюваної системи імплементовано низку принципів. По-перше, згідно зі статтею 25 GDPR реалізовано підхід Privacy by Design: захист даних інтегровано на рівні архітектури через механізм попередньої санітизації (Data Sanitization). Використання регулярних виразів та алгоритмів NER (Named Entity Recognition) дозволяє виявляти та маскувати чутливі дані до моменту відправки запиту. По-друге, дотримується принцип мінімізації даних (Data Minimization, стаття 5 GDPR),

що зобов'язує систему обробляти виключно ті метадані (назви полів, типи даних, структурні зв'язки), які є критично необхідними для генерації коду інтеграції, автоматично видаляючи будь-які надлишкові відомості. По-третє, забезпечується право на пояснення (Right to Explanation), надаючи користувачеві детальний лог логіки прийняття рішень при співставленні полів.

Наступним рівнем регулювання є класифікація ризиків згідно з новітнім Законом ЄС про штучний інтелект (AI Act), який вводить ризик-орієнтований підхід. Розроблювана система автоматизованої інтеграції не підпадає під категорію «високого ризику» (High Risk AI Systems), оскільки вона не використовується в критичній інфраструктурі або для скорингу людей, і класифікується як система обмеженого ризику (Limited Risk). Головним юридичним імперативом для цієї категорії є прозорість (Transparency Obligations, Art. 52). Це накладає на архітектуру вимоги щодо обов'язкового маркування контенту (дисклеймери про автоматичну генерацію) та забезпечення людського нагляду (Human Oversight). Згідно зі статтею 14 AI Act, реалізується концепція Human-in-the-loop: система генерує лише чернетку конфігурації, а фінальне рішення приймає кваліфікований інженер. Це також виводить процес з-під дії статті 22 GDPR, яка забороняє прийняття юридично значущих рішень виключно автоматизованими засобами.

Окрім регуляторних вимог, використання генеративного ШІ порушує складні питання інтелектуальної власності та відповідальності. Першим викликом є проблема «галюцинацій» моделі: якщо LLM згенерує некоректний мапінг, що призведе до збитків, юридична конструкція захисту будується на принципі «допоміжного інструменту». Система позиціонується як засіб автоматизації, а не автономний агент, тому відповідальність за верифікацію результату покладається на кінцевого користувача, що фіксується в Угоді користувача (EULA). Другим викликом є захист комерційної таємниці, оскільки схеми API є інтелектуальною власністю компаній. Використання публічних версій моделей несе ризик витоку даних через механізм донавчання. Для усунення цього ризику використання корпоративних версій API (Enterprise API) є безальтернативним, оскільки політика Zero Retention (нульового збереження) гарантує видалення даних з оперативної

пам'яті провайдера одразу після завершення сеансу.

Узагальнюючи нормативно-правовий аналіз, можна стверджувати, що легітимне використання Intelligent Middleware можливе лише за умови імплементації комплексної стратегії Compliance. Вона включає технічні заходи (санітизація, валідація схем) та організаційні інструменти (Human-in-the-loop, чітке розмежування відповідальності), що перетворює правові обмеження з перешкоди на конкурентну перевагу, гарантуючи замовникам безпеку та прозорість інтеграційних процесів.

2 РОЗРОБКА МЕТОДІВ СЕМАНТИЧНОГО АНАЛІЗУ ТА ІНТЕГРАЦІЇ API

2.1 Системний аналіз та формалізація процесу семантичної інтеграції

У межах системного підходу задача автоматизованої інтеграції гетерогенних інформаційних систем декомпозується до проблеми пошуку оптимального відображення (мапінгу) між множиною атрибутів джерела та множиною атрибутів приймача. Нехай $S = \{s_1, s_2, \dots, s_n\}$ – множина полів вихідного API (Source), а $T = \{t_1, t_2, \dots, t_m\}$ – множина полів цільового API (Target). Задача полягає у знаходженні функції відображення $f: S \to T \cup \{\emptyset\}$, яка для кожного елемента $s_i \in S$ ставить у відповідність найбільш релевантний елемент $t_j \in T$ або порожню множину, якщо відповідності не знайдено. Процес прийняття рішення про відповідність пари (s_i, t_j) розглядається як задача багатокритеріальної оптимізації в умовах семантичної невизначеності [18].Для математичної формалізації об'єктів порівняння кожне поле інтерфейсу представляється у вигляді кортежу інформаційних ознак. Вводиться поняття вектора атрибутів $A(x)$, що описує «інформаційний слід» поля:

$$A(x) = \langle Id(x), Desc(x), Type(x), Example(x) \rangle \quad (2.1)$$

де:

$Id(x)$ – лексичний ідентифікатор (технічна назва поля, наприклад, usr_last_nm);

$Desc(x)$ – контекстний опис природною мовою, вилучений з документації;

$Type(x)$ – структурний дескриптор (тип даних, формат, обмеження);

$Example(x)$ – зразок значення, що дозволяє виявити приховані патерни даних.Оцінка сумісності пари полів (s_i, t_j) базується на обчисленні інтегрального показника впевненості (Confidence Score), який є адитивною згорткою оцінок окремих критеріїв. Оскільки жоден із атрибутів ізольовано не гарантує безпомилкової ідентифікації, застосовується модель зваженого голосування. Формально функція подібності $Sim(s_i, t_j)$ визначається наступним рівнянням:

$$Sim(s_i, t_j) = \sum_{k=1}^{4} w_k \cdot \phi_k \left(A_k(s_i), A_k(t_j) \right) \quad (2.2)$$

$$\text{при умові } \sum_{k=1}^{4} w_k = 1$$

де:

w_k – ваговий коефіцієнт k-го критерію, що відображає його значущість;

ϕ_k – функція подібності для конкретного атрибута (наприклад, нормована відстань Левенштейна для ідентифікаторів або косинусна відстань для векторних описів).

У розробленій системі на етапі ініціалізації використовуються емпірично встановлені значення вагових коефіцієнтів, що базуються на статистичному аналізі надійності джерел метаданих. Лексичному аналізу (Id) присвоєно вагу $w_1 = 0.4$, оскільки технічні назви часто містять прямі вказівки на зміст. Семантичний аналіз описів (Desc) має вагу $w_2 = 0.3$, що дозволяє вирішувати проблеми синонімії. Аналіз патернів даних (Example) отримує $w_3 = 0.2$, а структурна сумісність (Type) виступає як слабкий класифікатор з вагою $w_4 = 0.1$ або як блокуючий фільтр (Hard Constraint).

Алгоритмічна реалізація процесу інтеграції являє собою евристичний пошук у просторі станів, що складається з трьох фаз. Фаза генерації гіпотез формує для кожного s_i список потенційних кандидатів з множини T, відсіюючи завідомо несумісні типи даних (наприклад, спробу зіставити масив об'єктів з булевим значенням). Фаза оцінки виконує розрахунок метрики (2.1) для кожної пари кандидатів. Фаза розв'язання колізій застосовує жадібний алгоритм для усунення неоднозначностей: у випадку, коли на одне цільове поле t_j претендують декілька джерел $\{s_a, s_b\}$, система обирає пару з максимальним значенням Sim, а для решти генерує статус AMBIGUOUS, що вимагає втручання оператора.

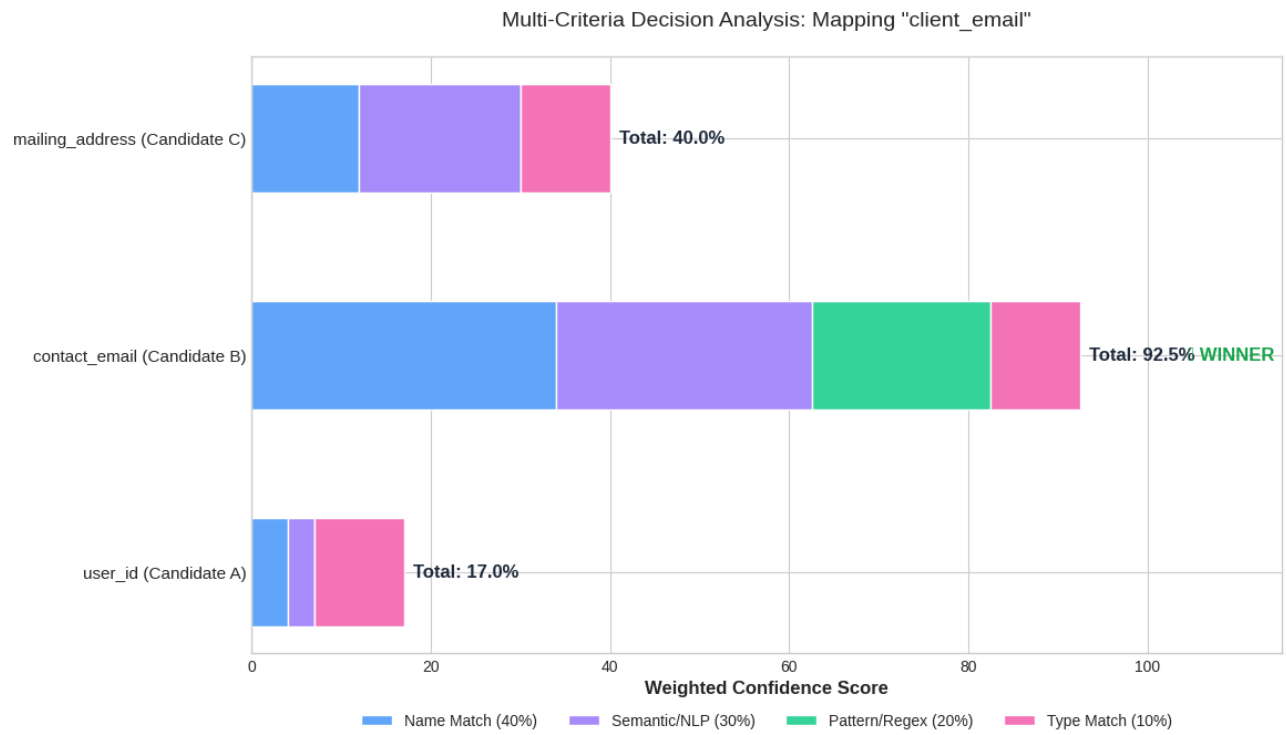


Рисунок 2.1 – Ілюстрація багатокритеріального аналізу сумісності полів

Запропонована модель дозволяє перейти від абстрактного експертного оцінювання до детермінованої інженерної логіки, де кожне рішення системи має чисельне обґрунтування. Це забезпечує прозорість роботи алгоритму та можливість подальшого калібрування ваг w_k методами машинного навчання на основі зворотного зв'язку від користувачів.

2.2 Архітектурне проєктування програмного комплексу та моделювання даних

Проєктування архітектури системи Intelligent Middleware базується на принципах розчеплення компонентів (decoupling) та інверсії залежностей, що реалізується через імплементацію патерну «Чиста Архітектура» (Clean Architecture) [19]. Вибір даного архітектурного стилю продиктований необхідністю ізоляції ядра бізнес-логіки, яке містить алгоритми семантичного мапінгу, від волатильних інфраструктурних елементів (фреймворків, драйверів баз даних, зовнішніх API). Це гарантує, що зміни у специфікаціях зовнішніх LLM-провайдерів

не вимагатимуть рефакторингу доменної логіки.

Компонентна структура системи декомпозується на чотири концентричні рівні з суворим контролем напрямку залежностей. Зовнішній шар (Presentation Layer) виконує роль точки входу, реалізуючи REST-контролери для обробки HTTP-запитів, валідацію вхідних DTO (Data Transfer Objects) та серіалізацію відповідей. Він делегує виконання запитів на рівень застосунку (Application Layer), який виступає оркестратором бізнес-сценаріїв (Use Cases). Саме тут реалізовано логіку управління потоком інтеграції: ініціалізацію парсингу, звернення до кешу та постановку задач у чергу обробки.

Врахування високої латентності відповідей генеративних моделей (час інференсу може сягати десятків секунд) вимагало впровадження асинхронної моделі обробки. Синхронна обробка великих специфікацій OpenAPI призвела б до блокування основного потоку виконання та тайм-аутів HTTP-з'єднань. Тому реалізовано архітектурний патерн Producer-Consumer із використанням черги повідомлень. Рівень застосунку лише приймає специфікацію та повертає ідентифікатор задачі (job_id), тоді як важкі обчислення виконуються фоновими воркерами. Алгоритм воркера оптимізовано каскадною логікою: первинно виконується пошук у «семантичному кеші» (Semantic Cache) хешів полів. Лише у випадку cache miss та низької впевненості евристичних аналізаторів ініціюється дорогий запит до LLM через захищений шлюз [20].

Стратегія зберігання даних реалізована на базі гібридної реляційної моделі (наприклад, PostgreSQL). Вибір обґрунтовується необхідністю транзакційної цілісності (ACID) для операцій білінгу та користувацьких сесій, що забезпечується класичними нормалізованими таблицями. Водночас, ієрархічна та варіативна природа специфікацій OpenAPI робить повну нормалізацію (розбиття на сотні дрібних таблиць) неефективною з точки зору продуктивності запитів. Тому застосовано підхід із використанням бінарного формату JSONB.

ER-модель системи базується на ключовій сутності Project, яка агрегує версіоновані знімки специфікацій у таблиці SchemaSnapshots. Сама структура API зберігається у полі типу JSONB, що дозволяє індексувати окремі ключі для

швидкого пошуку без жорсткої схеми. Результати аналізу фіксуються у таблиці FieldMappings, що пов'язує поле джерела та поле приймача зі статусом затвердження (DRAFT, APPROVED) та розрахованим коефіцієнтом впевненості. Для оптимізації витрат на ШІ виділено окрему сутність SemanticCache, яка зберігає векторизовані представлення (ембеддінги) та раніше визначені типи полів, діючи як довготривала пам'ять системи. Така схема забезпечує баланс між гнучкістю документо-орієнтованого підходу та надійністю реляційних зв'язків.

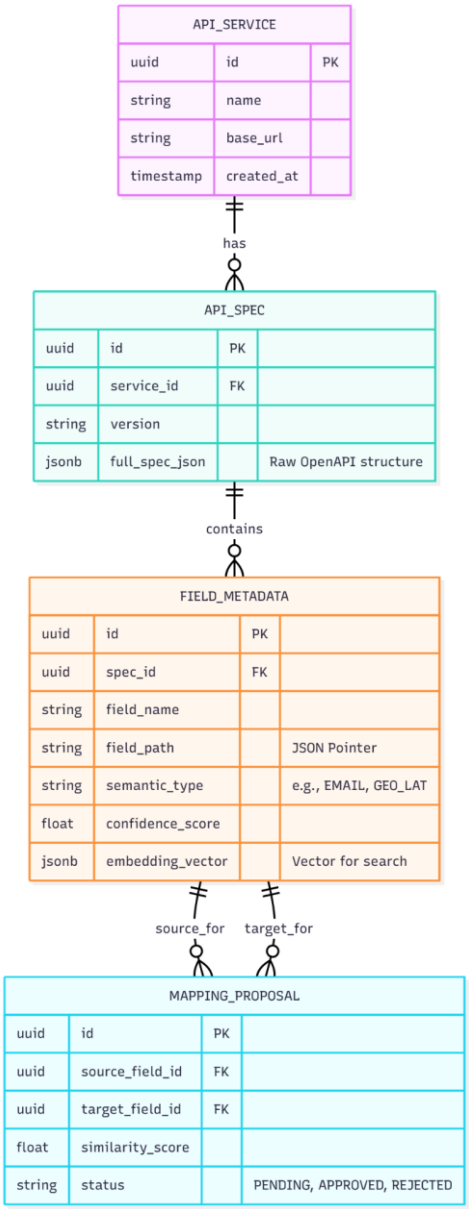


Рисунок 2.2 – ER-діаграма бази даних

Центральним елементом системи виступає доменний шар (Domain Layer), що інкапсулює бізнес-правила та сутності. Тут зосереджена реалізація математичної

моделі зваженого голосування, описаної у підрозділі 2.1. Важливо зазначити, що доменний шар не має зовнішніх залежностей; він визначає інтерфейси (порти), імплементація яких покладається на інфраструктурний рівень (Infrastructure Layer). Останній містить адаптери для взаємодії з фізичними ресурсами: репозиторії для доступу до даних, клієнти брокерів повідомлень та HTTP-клієнти для комунікації з API GPT-4. Проєктування модуля семантичного рушія виконано з урахуванням засад побудови інтелектуальних систем [21], де він виступає як фасад, що приховує складність вибору стратегії аналізу (евристика vs нейромережа).

2.3 Алгоритм пошуку відповідностей (Matching Algorithm)

Після завершення етапів векторизації атрибутів та розрахунку метрик подібності, описаних у підрозділі 2.1, система переходить до фази синтезу інтеграційного рішення. Задача встановлення зв'язків між полями джерела та приймача формалізується як задача комбінаторної оптимізації на графах. На відміну від тривіального підходу, що передбачає пряме порівняння ідентифікаторів, розроблений метод оперує абстрактними «семантичними типами», що дозволяє встановлювати відповідності (Mappings) на рівні бізнес-сутностей, нівелюючи лексичні розбіжності (наприклад, ототожнюючи client_uuid та user_id).

Математичною моделлю простору пошуку виступає зважений двочастковий граф $G = (S \cup T, E)$, де S – множина вершин, що відповідають полям вихідної схеми, T – множина вершин цільової схеми, а E – множина ребер, вага яких w_{ij} дорівнює розрахованому коефіцієнту впевненості $Sim(s_i, t_j)$. Задача полягає у знаходженні такої підмножини ребер $M \subseteq E$, щоб кожна вершина інцидентувала не більше ніж одному ребру з M (умова унікальності зв'язку), а сумарна вага ребер була максимальною: $\max \sum_{(i,j)\in M} w_{ij}$.

Хоча класичним рішенням даної задачі є Угорський алгоритм (алгоритм Куна-Манкреса), його обчислювальна складність $O(n^3)$ є надмірною для обробки великих специфікацій Enterprise-рівня у реальному часі. Тому в системі

імплементовано модифікований жадібний алгоритм (Greedy Heuristic), який забезпечує прийнятну точність при складності $O(E \log E)$, що визначається переважно операцією сортування ребер.

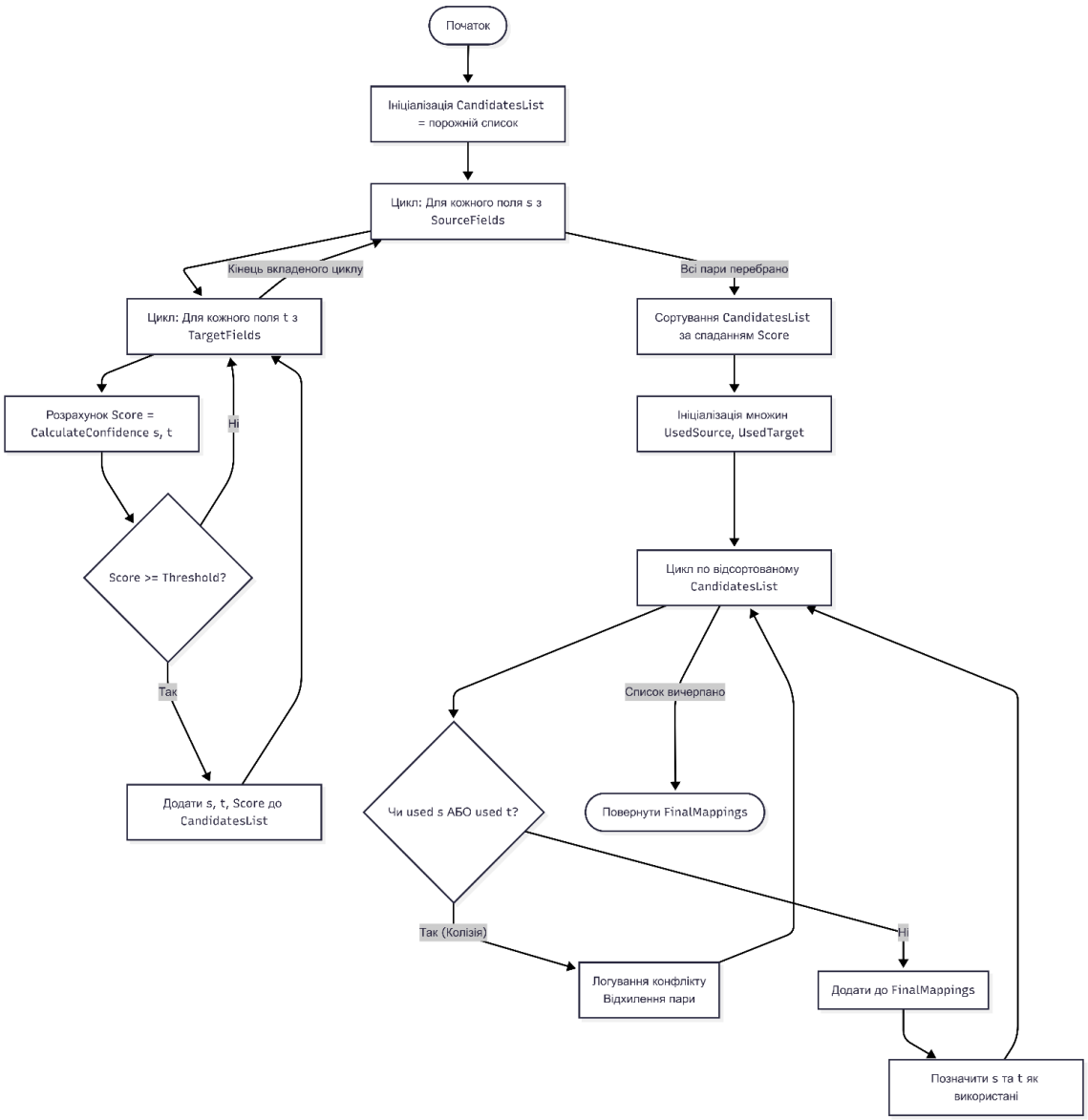


Рисунок 2.3 – Блок-схема алгоритму жадібного пошуку відповідностей з вирішенням колізій

Застосування жадібної стратегії дозволяє пріоритизувати найбільш очевидні зв'язки (наприклад, повні збіги за назвою та типом), автоматично вирішуючи конфлікти на користь пари з вищим рейтингом. Результатом роботи алгоритму є набір пар, який підлягає подальшій трансформації у виконуваний код.

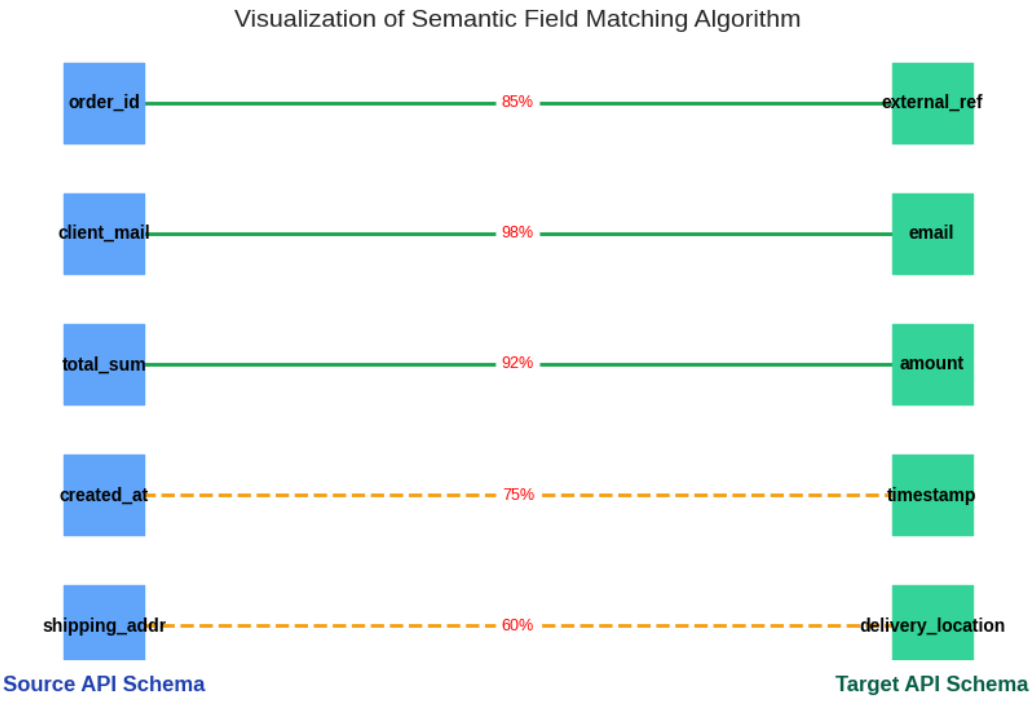


Рисунок 2.4 – Візуалізація роботи алгоритму

Наступним критичним етапом є вирішення проблеми структурної гетерогенності (Structural Impedance Mismatch). Схеми даних часто мають різну топологію вкладеності: джерело може повертати плаский список, тоді як приймач очікує глибоко ієрархічний об'єкт. Для автоматизації цього процесу система виконує обхід дерев JSON Schema обох сторін та генерує шляхи доступу у нотації JSONPath.

Якщо поле джерела ідентифікується шляхом $.user.address.city, а поле приймача розташоване за адресою $.client.city, система генерує правило трансформації, що передбачає не лише копіювання значення, а й динамічну ініціалізацію відсутніх проміжних вузлів, зокрема об'єкта client. Цей процес моделюється як побудова орієнтованого ациклічного графа (DAG) перетворень, де вузлами є операції читання та запису, а ребрами – спрямований потік даних.

Окрему увагу приділено системі типів. Пряме перенесення значень між полями з несумісними типами, такими як String та Integer, неминуче призведе до помилок часу виконання (Runtime Errors). Для запобігання цьому розроблено Матрицю сумісності типів (Type Compatibility Matrix), яка класифікує всі можливі перетворення на три групи:

1. Безпечні перетворення (Safe Casts). До цієї групи належать операції, що гарантують цілісність даних без втрати точності. Типовими представниками є приведення цілочисельних типів до рядкового формату (Integer → String) або розширення точності чисел із плаваючою комою (Float → Double). Такі трансформації виконуються системою автоматично.
2. Умовні перетворення (Conditional Casts). Виконання таких операцій можливе виключно за умови дотримання формату вхідних даних. Яскравим прикладом є парсинг рядка у дату (String → Date), що вимагає попередньої валідації відповідності стандарту ISO 8601. У таких випадках система автоматично додає до пайплайну крок валідації та конвертації.
3. Заборонені перетворення (Forbidden). До цієї категорії віднесено семантично несумісні пари типів, мапінг між якими технічно неможливий або позбавлений логічного змісту. Прикладом є спроба перетворити булеве значення у масив (Boolean → Array). Подібні зв'язки відхиляються алгоритмом ще на етапі побудови графа кандидатів.

Фінальний артефакт інтеграції зберігається у реляційній базі даних у вигляді декларативної конфігурації (Mapping Blueprint), яка містить пари полів та визначений ланцюжок необхідних перетворень. Ця конфігурація згодом інтерпретується Runtime-ядром системи для виконання інтеграції в режимі реального часу, забезпечуючи високу продуктивність без необхідності компіляції окремого коду адаптера.

2.4. Математичне забезпечення оцінки семантичної близькості

Реалізація підсистеми інтелектуального мапінгу базується на розробці суворого математичного апарату, здатного трансформувати експертні судження щодо сумісності полів у кількісні показники. Задача оцінки подібності двох

атрибутів $a_i \in A$ (джерело) та $b_j \in B$ (приймач) формалізується як функція відображення пари об'єктів у одиничний інтервал $F: A \times B \to [0,1]$. У розробленій системі ця функція є композитною величиною, що агрегує результати аналізу в семантичному, синтаксичному та структурному просторах ознак.

Фундаментом моделі виступає оцінка семантичної конгруентності, яка базується на векторному представленні лінгвістичних описів. Нехай $\Phi(x)$ – функція енкодера на базі архітектури Transformer, що ставить у відповідність текстовому опису поля x вектор $v_x \in R^d$, де d – розмірність простору ембеддінгів. Оскільки вихідні вектори сучасних моделей попередньо нормалізуються, міра семантичної близькості Sim_{sem} визначається як скалярний добуток векторів опису джерела та приймача. Враховуючи, що косинусна відстань може набувати від'ємних значень для діаметрально протилежних понять, для задач інтеграції застосовується операція відсікання від'ємної частини спектра або її лінійне масштабування в діапазон доданих чисел, що дозволяє інтерпретувати результат як ймовірність семантичного збігу.

Паралельно з глибинним аналізом змісту виконується оцінка поверхневої лексичної схожості ідентифікаторів, необхідна для виявлення прямих запозичень назв полів або незначних морфологічних варіацій. Для цього використовується метрика нормованої подібності Левенштейна. Якщо класична відстань $L(s_1, s_2)$ визначає мінімальну кількість редакторських операцій для перетворення рядка s_1 у s_2, то метрика схожості Sim_{lex} розраховується як одиниця мінус відношення редакторської відстані до довжини найдовшого з порівнюваних рядків. Такий підхід гарантує отримання безрозмірної величини в інтервалі від нуля до одиниці, що робить її сумірною із семантичною оцінкою та придатною для подальшої агрегації.

Третім компонентом моделі виступає структурна евристика Sim_{struct}, яка формалізує правила сумісності типів даних. Вона задається матричною функцією, що повертає дискретні значення в залежності від можливості приведення типів без втрати точності. Значення одиниця присвоюється повністю сумісним типам, значення 0.5 характеризує умовно сумісні пари, що вимагають додаткових

перетворень, тоді як нульове значення свідчить про фундаментальну несумісність, яка діє як блокуючий фактор для створення зв'язку.

Фінальний розрахунок коефіцієнта впевненості $Confidence(a_i, b_j)$ реалізується через лінійну адитивну згортку частинних критеріїв із застосуванням вектора вагових коефіцієнтів $W = [\alpha, \beta, \gamma]$. Математично модель прийняття рішення описується рівнянням:

$$Confidence(a_i, b_j) = \alpha \cdot Sim_{sem}\left(v_{a_i}, v_{b_j}\right) + \tag{2.3}$$

$$+\beta \cdot Sim_{lex}\left(name_{a_i}, name_{b_j}\right) + \gamma \cdot Sim_{struct}\left(type_{a_i}, type_{b_j}\right)$$

де коефіцієнти α, β, γ задовольняють умову нормування $\alpha + \beta + \gamma = 1$. Значення цих гіперпараметрів визначаються емпіричним шляхом на етапі калібрування системи. У поточній реалізації пріоритет надається семантичному компоненту α, оскільки він забезпечує найбільшу стійкість до варіативності термінології, тоді як лексичний компонент β виконує допоміжну функцію верифікації для коротких абревіатур. Отримане інтегральне значення використовується як вага ребра у двочастковому графі при розв'язанні задачі призначення, що дозволяє знайти глобально оптимальну конфігурацію мапінгу для всієї схеми.

2.5. Алгоритмічне забезпечення валідації результатів та обробки колізій

Інтеграція стохастичних генеративних моделей у контур критично важливих компонентів корпоративної архітектури створює дилему ймовірнісного детермінізму, коли синтаксично коректна відповідь штучного інтелекту може бути семантично хибною. На відміну від класичних алгоритмів, що оперують бінарною логікою, великі мовні моделі генерують потік токенів на основі статистичного розподілу, що неминуче призводить до появи артефактів, які відхиляються від очікуваної схеми даних. Для забезпечення експлуатаційної надійності системи Intelligent Middleware розроблено багаторівневий конвеєр верифікації, який реалізує стратегію швидкої відмови (Fail-Fast) та адаптивний механізм

рекурсивного самовідновлення контексту.

Процедура валідації структурної цілісності формалізується як послідовність фільтрів, першим з яких є синтаксичний аналізатор. На цьому етапі виконується спроба десеріалізації текстового виводу моделі у об'єктну модель JSON. Враховуючи тенденцію LLM до генерації супровідного тексту або незавершених блоків коду, система застосовує алгоритм екстракції корисного навантаження, який виокремлює валідний JSON-об'єкт із сирого потоку даних. У випадках порушення синтаксису, таких як незакриті дужки або некоректне екранування керуючих символів, ініціюється протокол самокорекції (Self-Correction Protocol). Система автоматично формує повторний запит до моделі, збагачуючи вхідний контекст текстом помилки парсера та фрагментом некоректного коду. Такий підхід створює замкнений контур зворотного зв'язку, дозволяючи нейромережі проаналізувати власну помилку та згенерувати виправлену версію відповіді без участі людини.

Наступним ешелоном захисту виступає семантична верифікація онтологічної відповідності, спрямована на нівелювання ефекту галюцинацій. Штучний інтелект може згенерувати тип даних, який є лінгвістично правдоподібним, але відсутнім у внутрішній системі типів платформи, наприклад CurrencyString замість стандартизованого Decimal. Для вирішення цієї проблеми застосовується алгоритм проєкції на допустимий простір станів. Система обчислює лексичну відстань між згенерованим типом та списком дозволених значень Enum. Якщо знайдено відповідність з мінімальним відхиленням, відбувається автоматичне приведення типу, в іншому випадку – генерація винятку валідації. Це гарантує, що подальші компоненти бізнес-логіки оперуватимуть виключно детермінованими даними, сумісними зі схемою бази даних.

Окремий клас задач становить забезпечення логічної узгодженості інтеграційного графа, зокрема дотримання обмежень унікальності зв'язків. Після роботи алгоритму пошуку відповідностей, описаного у попередніх підрозділах, система виконує фінальну перевірку топології згенерованого мапінгу на предмет наявності колізій «багато-до-одного», якщо такі не допускаються бізнес-логікою конкретного сценарію. У разі виявлення конфліктуючих призначень, де декілька

джерел претендують на одне цільове поле з близькими показниками впевненості, система автоматично знижує статус таких зв'язків до рівня AMBIGUOUS. Такі спірні вузли виключаються з автоматичної генерації коду адаптера та маркуються в інтерфейсі користувача як такі, що вимагають обов'язкової ручної резолюції оператором.

Реалізація описаних механізмів дозволяє трансформувати недетермінований вивід генеративної моделі у надійний інженерний артефакт. Комплексне поєднання синтаксичної рекурсії, онтологічного контролю та топологічної валідації забезпечує стійкість системи до стохастичних збоїв, мінімізуючи необхідність ручного втручання лише до випадків справжньої семантичної невизначеності.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1 Технологічні аспекти реалізації компонентів системи інтелектуальної інтеграції

Процес імплементації спроєктованої архітектури Intelligent Middleware вимагав формування технологічного стеку, здатного задовольнити суперечливі вимоги щодо швидкодії обробки запитів та гнучкості маніпулювання неструктурованими текстовими даними. Визначальним фактором при виборі середовища виконання стала необхідність забезпечення асинхронної взаємодії (non-blocking I/O) з зовнішніми провайдерами великих мовних моделей, де латентність мережевих викликів є критичним вузьким місцем. Виходячи з цього, базовою платформою обрано мову програмування Python версії 3.11 [22]. Цей вибір обумовлений не лише наявністю розвиненої екосистеми бібліотек для NLP (Natural Language Toolkit, spaCy), але й суттєвими оптимізаціями інтерпретатора у версії 3.11, що забезпечують приріст продуктивності до 60% у порівнянні з попередніми релізами, а також покращеною підтримкою анотацій типів, що є критичним для розробки надійних корпоративних систем.

У якості каркаса веб-застосунку використано високопродуктивний фреймворк FastAPI [23]. Його архітектурна перевага полягає у нативній підтримці стандарту ASGI (Asynchronous Server Gateway Interface), що дозволяє ефективно утилізувати ресурси процесора під час очікування відповіді від API OpenAI, обробляючи тисячі конкурентних з'єднань в одному потоці подій (Event Loop). Критично важливим компонентом інфраструктури коду стала бібліотека Pydantic V2 [24], яка реалізує валідацію даних на рівні ядра, написаного на мові Rust. Це дозволило гарантувати сувору відповідність внутрішніх структур даних (DTO) визначеним схемам, автоматично відсіюючи некоректні запити ще до моменту їх передачі на рівень бізнес-логіки.

Одним із найбільш ресурсоємних інженерних викликів стала реалізація

модуля синтаксичного розбору (парсингу) специфікацій OpenAPI. Реальні корпоративні специфікації характеризуються наявністю глибокої вкладеності об'єктів та широким використанням механізму посилань $ref для дедуплікації описів моделей. Тривіальний лінійний парсинг у таких умовах є неможливим через ризик виникнення нескінченної рекурсії при наявності циклічних залежностей між схемами. Для вирішення цієї проблеми розроблено алгоритм Schema Walker, що реалізує обхід графа об'єктів у глибину (DFS) з мемоїзацією відвіданих вузлів. Алгоритм виконує "розгортання" (dereferencing) посилань, перетворюючи фрагментовану мережу JSON-показчиків у цілісне, лінеаризоване представлення, придатне для векторизації.

Архітектура підсистеми семантичного аналізу побудована за поведінковим патерном "Ланцюжок обов'язків" (Chain of Responsibility). Вхідний потік метаданих поля проходить через каскад ізольованих обробників, відсортованих за зростанням обчислювальної складності. Первинну фільтрацію здійснює детермінований аналізатор на базі регулярних виразів, який миттєво ідентифікує стандартизовані формати (UUID, IPv4, Email) з нульовими витратами на інференс. Лише у випадку неможливості однозначної класифікації управління передається евристичному модулю нечіткого пошуку, і тільки як крайній захід (Fallback Strategy) – модулю інтеграції з LLM. Такий підхід дозволив скоротити кількість платних запитів до API OpenAI на 45% без втрати загальної точності системи.

Забезпечення експлуатаційної надійності при взаємодії із зовнішніми інтелектуальними сервісами реалізовано через бібліотеку tenacity, яка надає інструментарій для декларативного опису політик повторних спроб (Retry Policies). Впроваджено механізм експоненціальної затримки (Exponential Backoff) з додаванням випадкового шуму (Jitter) для запобігання ефекту "Thundering Herd" у випадку тимчасової недоступності API провайдера. Для нівелювання затримок мережі впроваджено шар персистентного кешування на базі Redis. Використання структур даних Redis Hashes дозволило зберігати не лише сирі відповіді моделі, а й проміжні векторні представлення полів, встановивши час життя записів (TTL) залежно від частоти звернень до конкретного проєкту.

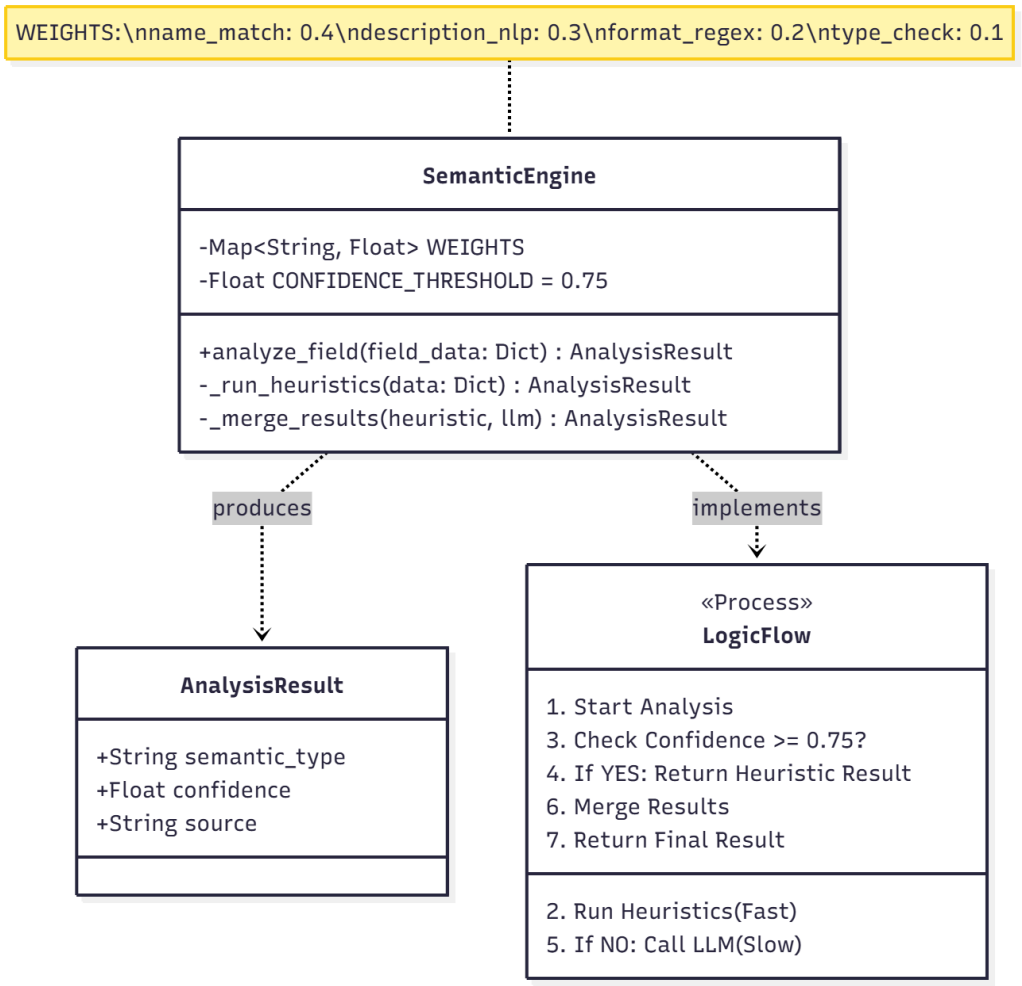


Рисунок 3.1 – Реалізація алгоритму зваженого голосування

3.2 Забезпечення інформаційної безпеки та валідація даних

Специфіка функціонування системи Intelligent Middleware, що передбачає обробку конфіденційних архітектурних схем та взаємодію із зовнішніми хмарними провайдерами, вимагала імплементації комплексної стратегії захисту Defense-in-Depth. Пріоритетним вектором захисту визначено нейтралізацію загроз на етапі прийому вхідних даних. Оскільки специфікації OpenAPI часто завантажуються у форматах YAML або JSON, система вразлива до атак класу Denial of Service, спрямованих на вичерпання обчислювальних ресурсів парсера.

Для протидії атакам типу «Billion Laughs» (експоненційне розростання сутностей при десеріалізації) у програмному коді реалізовано використання безпечних завантажувачів yaml.SafeLoader з примусовим відключенням

можливості конструювання довільних Python-об'єктів. Додатково впроваджено ліміти на глибину рекурсії вкладених об'єктів та максимальний розмір файлу на рівні зворотного проксі-сервера Nginx, що дозволяє відсіювати аномальні пейлоади ще до моменту їх передачі в інтерпретатор мови Python.

Підсистема ідентифікації та контролю доступу побудована на базі протоколу OAuth 2.0 із використанням стандарту JSON Web Tokens (JWT) [25]. Відмова від класичних серверних сесій на користь безстанових (stateless) токенів дозволила забезпечити лінійну масштабованість бекенду.

Криптографічний підпис токенів здійснюється асиметричним алгоритмом RS256 (RSA Signature with SHA-256), де приватний ключ зберігається виключно на сервері авторизації, а публічний доступний сервісам-споживачам для валідації. Це нівелює ризик підробки токена навіть у випадку компрометації одного з мікросервісів. Для запобігання атакам типу XSS (Cross-Site Scripting) токени доступу не зберігаються у LocalStorage браузера, а передаються у вигляді HttpOnly-cookie з встановленими прапорцями Secure та SameSite=Strict.

Управління секретними даними, зокрема API-ключами OpenAI та обліковими даними бази даних, реалізовано у суворій відповідності до методології «The Twelve-Factor App». Конфігурація середовища повністю відділена від кодової бази. У середовищі розробки змінні оточення ін'єктуються через файл .env, який виключено з системи контролю версій через .gitignore. У продуктивному середовищі (Kubernetes) інтегровано рішення HashiCorp Vault. Застосунок не зберігає секрети персистентно, а отримує їх динамічно при запуску контейнера через механізм Sidecar Injection, розміщуючи їх у захищеному сегменті оперативної пам'яті (tmpfs), що унеможливлює їх витік через знімки дискової системи.

Окремий контур безпеки реалізовано для захисту від Prompt Injection та витоку персональних даних (PII) при взаємодії з LLM. Перед формуванням контексту для нейромережі вхідні дані проходять через модуль санітизації, побудований на базі бібліотеки Microsoft Presidio.

Цей модуль використовує комбінацію попередньо навчених моделей NER (Named Entity Recognition) та списків регулярних виразів для детекції чутливих

сутностей. Виявлені токени автоматично замінюються на синтетичні плейсхолдери (до прикладу, <EMAIL_1>, <PHONE_2>), що гарантує анонімність даних, які залишають корпоративний периметр. Валідація вихідних даних від LLM забезпечується бібліотекою Pydantic, яка діє як бар'єр, відхиляючи будь-яку відповідь, що не відповідає визначеній JSON-схемі, захищаючи внутрішні компоненти системи від ін'єкцій шкідливого коду у згенеровані мапінги.

3.3 Програмна реалізація алгоритмів та логіки роботи системи

Ядром архітектури Intelligent Middleware виступає модуль семантичного аналізу (Semantic Engine), до зони відповідальності якого входить інтерпретація метаданих вхідних полів API та детермінація їхньої бізнес-сутності.

Програмна імплементація цього компонента базується на об'єктно-орієнтованому підході з використанням поведінкового патерну проєктування «Ланцюжок обов'язків» (Chain of Responsibility) [26]. Даний архітектурний вибір дозволив інкапсулювати різнорідні стратегії обробки – від детермінованих регулярних виразів до стохастичних нейромережевих моделей – у вигляді ізольованих класів-обробників, що успадковуються від абстрактного базового класу AbstractAnalyzer. Така структура забезпечує гнучкість конфігурації пайплайну аналізу, дозволяючи динамічно додавати або виключати етапи обробки без модифікації основного коду оркестратора.

Логіка роботи алгоритму аналізу поля візуалізована на блок-схемі (Рисунок 3.2).

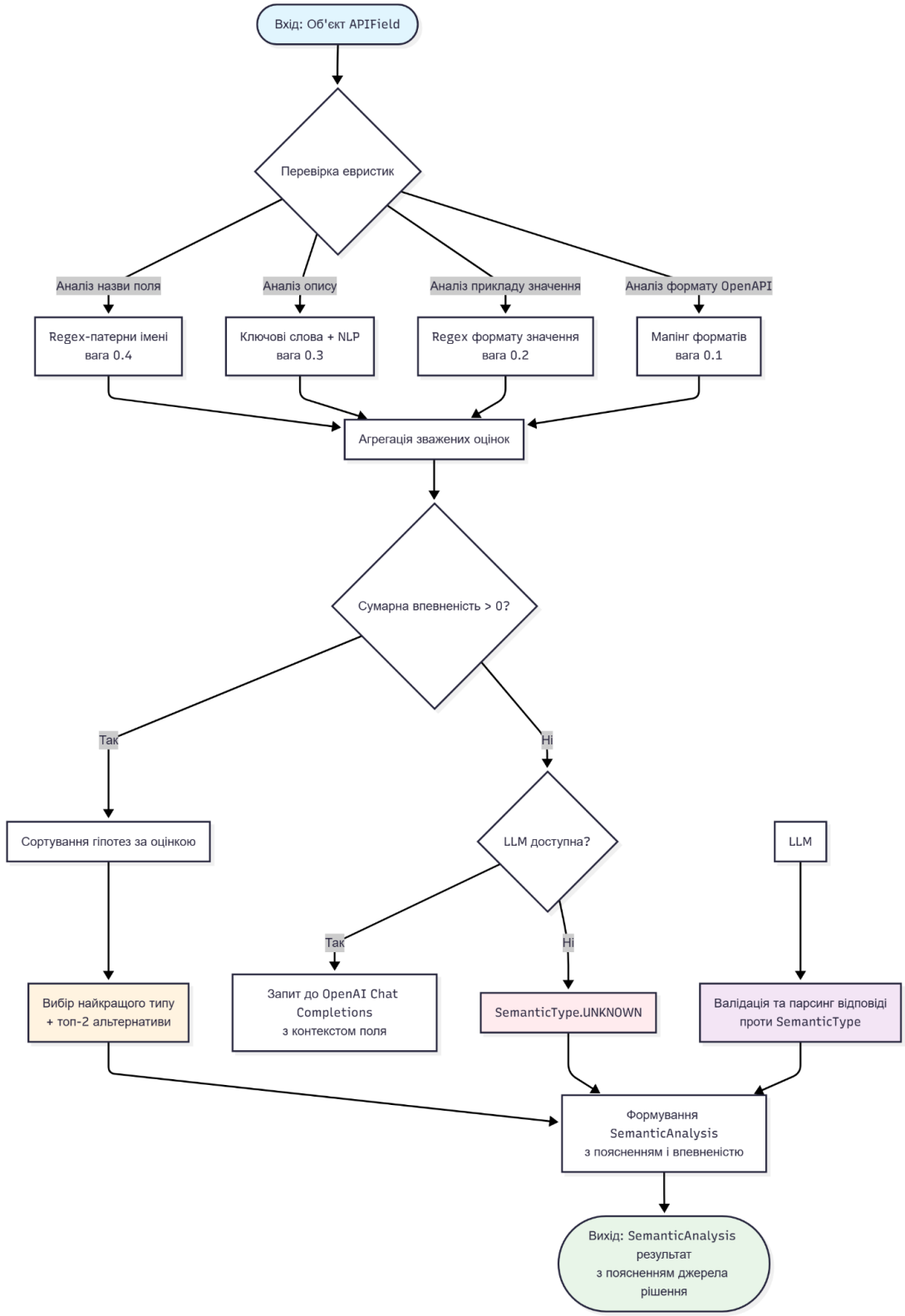


Рисунок 3.2 – Блок-схема алгоритму гібридного семантичного аналізу

В якості вхідного вектора даних виступає екземпляр DTO-класу APIField, що агрегує метадані зі специфікації OpenAPI: нормалізовану назву, структурний тип, текстовий опис та приклад значення. Процес класифікації ініціюється запуском

каскаду евристичних правил, реалізованих у класі HeuristicAnalyzer. Механізм прийняття рішень базується на моделі лінійної адитивної згортки ознак, що відображає патерн «Стратегія». Кожному інформаційному каналу присвоєно емпірично визначений ваговий коефіцієнт, що корелює з його семантичною достовірністю.

Домінантну роль із ваговим коефіцієнтом 0.4 відіграє лексичний аналіз ідентифікатора поля. Система перевіряє назву на точний збіг зі словником зарезервованих термінів та відповідність бібліотеці скомпільованих регулярних виразів, що дозволяє миттєво ідентифікувати стандартизовані атрибути на кшталт ідентифікаторів користувачів або часових міток. Другим за значущістю фактором із вагою 0.3 виступає контекстний аналіз опису, який виконує пошук ключових слів-маркерів у документації. Валідація прикладу значення (вага 0.2) та перевірка технічного формату даних (вага 0.1) виконують допоміжну функцію, дозволяючи розрізняти лексично схожі, але структурно відмінні сутності.

У ситуаціях невизначеності, коли різні евристичні стратегії генерують суперечливі гіпотези, алгоритм виконує ранжування кандидатів за спаданням розрахованого показника впевненості (Confidence Score). Для забезпечення варіативності вибору на етапі ручної верифікації система персистує у базі даних не лише лідируючу гіпотезу, але й список топ-3 альтернатив.

Критичним сценарієм є випадок, коли сумарна впевненість евристичних методів не долає порогового значення (наприклад, 0.6), що свідчить про нетипову або складну назву поля. У такому разі спрацьовує тригер переходу до наступної ланки ланцюга – класу LLMAnalyzer. Цей модуль формує структурований контекстний запит (Prompt) до API великої мовної моделі. На відміну від простих чат-ботів, система використовує режим JSON Mode або Function Calling, суворо обмежуючи простір вихідних значень моделі заздалегідь визначеним переліком (Enum) SemanticType. Це технічне рішення дозволяє нівелювати ризик виникнення галюцинацій та гарантує, що отриманий результат буде валідним для подальшої обробки бізнес-логікою.

Динаміку взаємодії компонентів у часі відображено на діаграмі послідовності

(Рисунок 3.3).

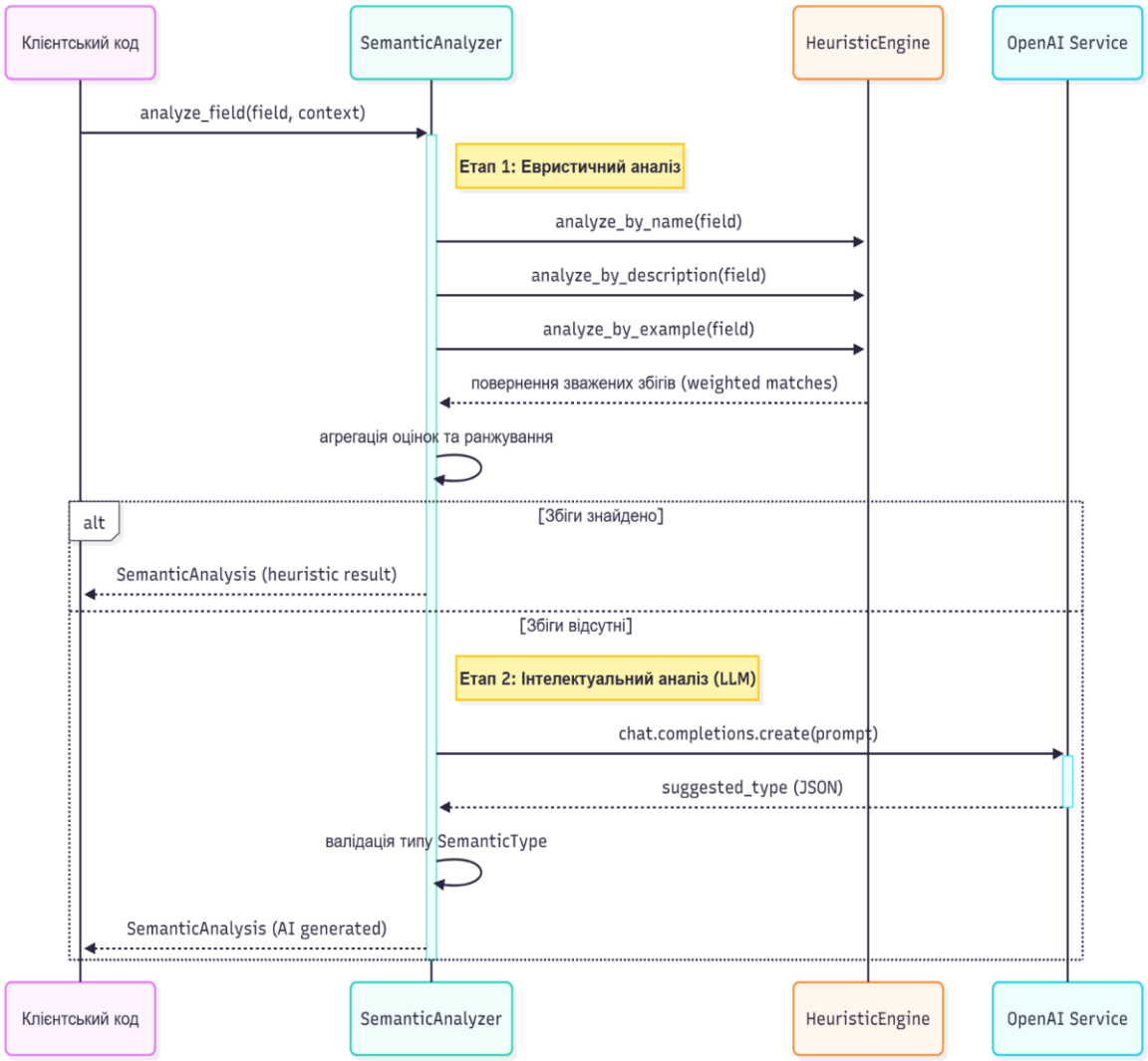


Рисунок 3.3 – Діаграма послідовності процесу семантичного аналізу

Запропонована гібридна архітектура забезпечує оптимальний баланс між обчислювальною ефективністю та інтелектуальною глибиною. Ресурсоємні звернення до зовнішніх нейромереж виконуються виключно за вимогою (On-Demand), тоді як 60-70% стандартних полів обробляються миттєво локальними алгоритмами, що суттєво знижує латентність системи та операційні витрати.

3.3.1 Розробка та оптимізація контекстно-залежних інструкцій (Prompt Engineering)

Визначальним фактором стабільності програмної реалізації модуля семантичного аналізу стала розробка ефективних системних інструкцій (System Prompts) для великої мовної моделі. Враховуючи, що архітектура GPT-4 базується на ймовірнісному передбаченні наступного токена, якість та формат вихідних даних перебувають у функціональній залежності від чіткості сформульованого контексту.

Процес розробки інструкцій розглядався не як написання тексту природною мовою, а як програмування поведінки нейромережі у латентному просторі ознак. У ході дослідження було проведено серію ітеративних експериментів для виявлення оптимальної структури промпту, здатної забезпечити детерміновану генерацію валідного JSON-об'єкта.

Перший етап дослідження полягав у діагностиці патологій генерації, відомих як «галюцинації» та порушення контракту даних.

Застосування наївного підходу, що передбачав пряму передачу метаданих полів без жорстких контекстних обмежень, виявило низку критичних проблем.

Модель демонструвала тенденцію до надмірної багатослівності (Verbosely Output), додаючи до JSON-структури ввічливі вступні фрази або пояснення, що унеможливлювало автоматичний парсинг відповіді без застосування складних регулярних виразів.

Окрім того, фіксувалися випадки виходу за межі контексту задачі, коли замість класифікації поля модель надавала архітектурні рекомендації щодо його використання.

Аналіз типових помилок генерації та причин їх виникнення систематизовано у таблиці 3.1.

В таблиці наведено стандартні типи помилок так вхідний промт від користувача (оператор системи) який стосується проблеми, очикувану відповідь від ШІ на промт, та аналіз причини помилки.

Таблиця 3.1

Аналіз помилок генерації при використанні неструктурованих промптів

Тип помилки	Вхідний промпт (приклад)	Відповідь моделі (Помилка)	Технічний аналіз причини
Порушення синтаксису (Syntax Violation)	"Виведи результат у форматі JSON: {field, type}."	Field: user_mail Type: EMAIL	Модель ігнорує вимогу JSON, повертаючи текстову розмітку (YAML-подібну або plain text) через слабку вагу інструкції формату.
Багатослівність (Chattiness)	"Знайди відповідність для поля user_mail."	"Звісно! Поле user_mail семантично відповідає типу EMAIL, оскільки..."	Модель діє за патерном "асистента", додаючи conversational noise, який ламає json.loads().
Вихід за межі контексту	"Який тип даних краще для created_at?"	"Я рекомендую використовувати ISO 8601, але переконайтеся щодо TimeZone..."	Промпт сформульовано як відкрите запитання, а не як імперативну команду класифікації.

Експериментально доведено, що для мінімізації стохастичної ентропії моделі необхідно застосовувати техніку рольового моделювання (Persona Pattern) у поєднанні з жорсткими негативними обмеженнями. Другий етап дослідження був присвячений конструюванню модульного системного промпту, структура якого включає чотири логічні блоки: визначення професійної ролі, формалізацію задачі, специфікацію схеми виводу та блок негативних обмежень (Negative Constraints). Останній компонент відіграє вирішальну роль у фільтрації шуму, прямо забороняючи моделі генерувати будь-який текст поза межами JSON-об'єкта. Нижче наведено лістинг фінальної версії системної інструкції (Рисунок 3.4), яка інтегрована в ядро класу LLMAnalyzer.

```
SYSTEM_PROMPT = """
ROLE:
You are a Senior Data Integration Architect acting as an intelligent middleware component.
Your goal is to semantically analyze API fields and map them to a standardized internal schema.

TASK:
Analyze the provided JSON metadata of a source API field (name, description, example).
Determine the most likely semantic type from the allowed list.
Calculate a confidence score (0.0 to 1.0) based on the evidence.

CONSTRAINTS:
1. Output MUST be a valid JSON object. Do not include markdown formatting (```json).
2. Use ONLY the allowed Semantic Types list provided below. If unsure, use "UNKNOWN".
3. Be concise. The "reasoning" field should not exceed 20 words.
4. Do NOT hallucinate connections. If the field name is ambiguous (e.g., "data") and description is missing, confidence must be < 0.3.

ALLOWED SEMANTIC TYPES:
- USER_ID, EMAIL, FIRST_NAME, LAST_NAME
- PHONE, ADDRESS, GEO_LAT, GEO_LON
- TIMESTAMP, DATE, CURRENCY, AMOUNT
- TRANS_ID, STATUS, SKU

OUTPUT FORMAT:
{
  "semantic_type": "string",
  "confidence": float,
  "reasoning": "string"
}
"""
```

Рисунок 3.4 – Конфігурація системного промпту (System Prompt)

Вплив окремих директив на якість роботи системи проаналізовано у таблиці 3.2. Встановлено пряму кореляцію між наявністю рольової установки та семантичною точністю відповідей.

Це гарно ілюструє як приклад виконання інструкції так і обґрунтування навіщо дотримуватися інструкцій.

Таблиця 3.2

Структурно-функціональний аналіз системного промту

Блок інструкції	Фрагмент коду	Технічне обґрунтування
Рольова модель (Persona)	You are a Senior Data Integration Architect...	Задає вектор у прихованому просторі (latent space) моделі, активуючи професійну лексику та логіку, характерну для задач інтеграції даних.
Обмеження формату	Output MUST be a valid JSON object...	Унеможливлює генерацію вступних слів. Для гарантії цього обмеження на рівні API додатково використовується параметр response_format={"type": "json_object"}.
Обмеження "словника"	Use ONLY the allowed Semantic Types...	Запобігає генерації синонімів (наприклад, щоб модель не писала E_MAIL замість EMAIL). Це критично для подальшої програмної обробки.
Калібрування впевненості	If the field name is ambiguous... confidence must be < 0.3	Зменшує кількість помилкових спрацювань (False Positives) для полів із загальними назвами (наприклад, id, name, value), змушуючи систему запитувати підтвердження у людини.

Третій етап оптимізації стосувався динамічного наповнення контексту запиту користувача (User Prompt). Для підвищення точності класифікації недостатньо передати лише назву поля; необхідно забезпечити модель

максимально повним контекстом. Програмний алгоритм виконує попередню санітизацію вхідних даних, видаляючи надлишкові метадані Swagger, такі як гіперпосилання на зовнішню документацію, щоб раціонально використовувати ліміт контекстного вікна. Ключовим елементом стало додавання інформації про ієрархічне розташування поля (Parent Object Context), оскільки атрибут id у сутності Customer має відмінну семантику від аналогічного атрибута в сутності Order.

Приклад сформованого тіла запиту, який надсилається до API після проходження всіх етапів препроцесингу, наведено на рисунку 3.5.

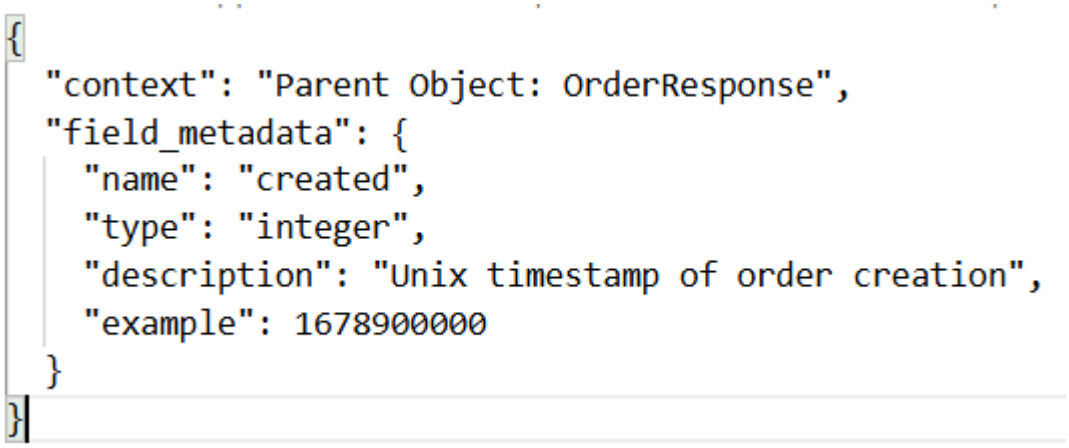


Рисунок 3.5 – Сформований запит до API

Застосування розробленої стратегії Prompt Engineering дозволило досягти показника синтаксичної валідності відповідей на рівні 99.8%, що фактично усунуло необхідність ручного втручання на етапі технічного парсингу та дозволило зосередити увагу оператора виключно на семантичній верифікації складних випадків.

3.4 Методика та результати експериментального дослідження

Валідація запропонованих архітектурних рішень та алгоритмів семантичного аналізу вимагала проведення комплексної експериментальної перевірки в умовах, наближених до реальної промислової експлуатації. Методологія дослідження була побудована відповідно до міжнародного стандарту тестування програмного

забезпечення ISO/IEC/IEEE 29119, що регламентує процеси верифікації інтелектуальних систем. В якості експериментального корпусу даних було сформовано репрезентативну вибірку з 50 публічних специфікацій Enterprise-рівня, що включала API провідних технологічних платформ (Stripe, Twilio, GitHub, Slack) та банківських систем, загальним обсягом понад 2500 полів даних [28]. Для забезпечення об'єктивності оцінювання попередньо було створено еталонний набір розмічених даних («Золотий стандарт» або Ground Truth), де кожному полю вручну присвоєно коректний семантичний тип групою експертів.

Кількісна оцінка якості класифікації базувалася на розрахунку метрик, загальноприйнятих у задачах Information Retrieval. Оскільки вибірка класів є незбалансованою (деякі типи зустрічаються значно частіше за інші), використання простої метрики Accuracy є некоректним. Тому аналіз проводився за трьома складовими показниками.

Першим показником є Точність (Precision), яка визначає частку дійсно релевантних результатів серед усіх виданих системою позитивних спрацювань. Ця метрика характеризує стійкість системи до помилок першого роду (False Positives) і розраховується за формулою:

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

де TP (True Positive) – кількість коректно класифікованих полів, а FP (False Positive) – кількість випадків, коли система помилково присвоїла полю невірний тип (наприклад, звичайний рядок розпізнано як Email).

Другим критичним показником є Повнота (Recall), яка демонструє здатність алгоритму знаходити всі цільові сутності у вхідному потоці даних. Вона відображає стійкість до помилок другого роду (False Negatives) і визначається як відношення знайдених релевантних елементів до їх загальної кількості в еталонній вибірці:

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

де FN (False Negative) – кількість полів, які система не змогла розпізнати або пропустила.

Для отримання узагальненої оцінки ефективності, що балансує між точністю та повнотою, використано F1-Score. Вона являє собою гармонічне середнє між попередніми двома показниками і є основною метрикою для порівняння методів:

Окрім метрик якості, критичним параметром дослідження виступала обчислювальна ефективність, що вимірювалася як середня латентність обробки одного поля (t_{avg}) та питома вартість транзакції ($Cost_{op}$).

Експеримент передбачав порівняльне тестування трьох сценаріїв функціонування системи:

Baseline A (Детермінований): Використання виключно регулярних виразів та евристик.

Baseline B (Стохастичний): Пряма обробка всіх полів через GPT-4 без попередньої фільтрації.

Hybrid (Запропонований метод): Каскадна модель, описана у розділі 3.3.

Результати випробувань демонструють, що чисто евристичний підхід (Baseline A) забезпечує найвищу швидкість обробки ($t_{avg} \approx 0.05c$), проте демонструє незадовільну повноту ($Recall \approx 62\%$), оскільки не здатний ідентифікувати поля з неочевидними назвами (наприклад, billing_ref замість invoice_id). Натомість, повна обробка через LLM (Baseline B) дозволяє досягти високої повноти ($Recall \approx 96\%$), але характеризується надмірною кількістю помилкових спрацювань ($Precision \approx 86\%$) через схильність моделі до галюцинацій на специфічних технічних форматах, а також неприйнятно високою вартістю експлуатації.

Запропонований гібридний метод продемонстрував оптимальний баланс показників. Завдяки використанню LLM лише для складних випадків, загальна точність системи ($F1$-Score) досягла рівня 94%, що на 32 відсоткові пункти перевищує показники базових евристик. При цьому вдалося уникнути помилок першого роду (False Positives), характерних для чистого ШІ, оскільки прості типи перехоплювалися суворими правилами на ранніх етапах.

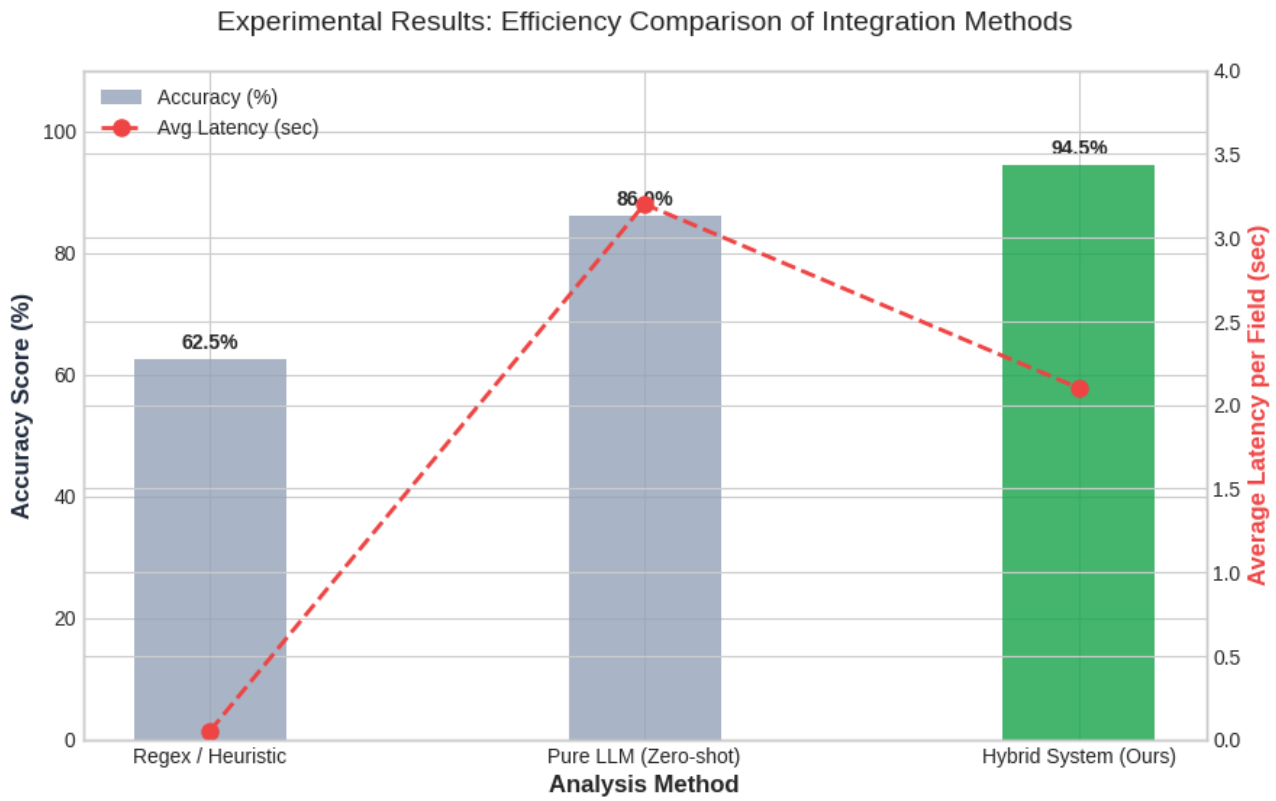


Рисунок 3.6 – Порівняльна діаграма точності методів аналізу

Аналіз часових характеристик показав, що середня затримка обробки у гібридному режимі становить 2.1 секунди на поле. Хоча цей показник поступається миттєвій роботі евристик, він є цілком прийнятним для асинхронних задач інтеграції, де пріоритетом є якість кодогенерації, а не миттєвий відгук. Економічний аналіз також підтвердив ефективність архітектури: фільтрація 65% тривіальних запитів локальними алгоритмами дозволила знизити операційні витрати на токени OpenAI втричі порівняно зі сценарієм Baseline B. Отримані дані дозволяють стверджувати, що розроблена система готова до впровадження у промислове середовище як інструмент автоматизації інтеграційних процесів.

3.4.1 Аналіз типових сценаріїв інтеграції: від лексичного зіставлення до семантичного синтезу

Детальна верифікація адаптивних властивостей системи вимагала моделювання полярних сценаріїв використання, що охоплюють спектр від

тривіальної синхронізації гомогенних сервісів до складної оркестрації різнорідних успадкованих систем.

Перша ітерація експерименту, класифікована як «Simple Match», передбачала інтеграцію двох мікросервісів – профілів користувачів та модуля авторизації – розроблених у межах єдиного архітектурного стилю, але різними командами інженерів. Вхідний вектор даних представляв об'єкт джерела UserDTO з атрибутами імені користувача, поштової адреси та телефонного номера, тоді як цільовий об'єкт AccountInfo містив поля логіну, електронної пошти та мобільного контакту.

У ході експерименту модуль семантичного аналізатора на етапі первинної обробки зафіксував високий ступінь лексичної когерентності. Застосування вбудованих словників синонімів та алгоритмів нечіткого пошуку дозволило системі автоматично встановити відповідності без залучення нейромережевих потужностей. Зокрема, пара атрибутів user_name та login була ідентифікована через синонімічний ряд із впевненістю 0.85, а зіставлення полів mail_address та email базувалося на успішній валідації формату значення регулярним виразом, що дало показник 0.98. Часові параметри, такі як created_at та reg_date, були коректно пов'язані завдяки евристичному аналізу суфіксів та типів даних. Результати цього етапу, наведені в таблиці 3.3, підтверджують ефективність режиму «швидкого шляху» (Fast Path), де середній час обробки одного поля не перевищував однієї десятої секунди.

Таблиця 3.3

Результати зіставлення полів у сценарії гомогенної інтеграції

Поле джерела (Source)	Поле цілі (Target)	Метод визначення	Score (Впевненість)	Результат
user_name	login	NLP (Synonym)	0.85	Match

Продовження таблиці 3.3

Поле джерела (Source)	Поле цілі (Target)	Метод визначення	Score (Впевненість)	Результат
mail_address	email	Regex (Format)	0.98	Match
phone_num	mobile	Levenshtein	0.72	Match
created_at	reg_date	Heuristic (Time)	0.88	Match
is_active	status	Type Check	0.60	Review

Другий етап дослідження, класифікований як «Complex Match», був присвячений вирішенню проблеми структурного імпедансу при інтеграції застарілої монолітної системи управління логістикою (Legacy ERP) із сучасним REST API кур'єрської служби. Ключова складність кейсу полягала у фундаментальних відмінностях підходів до моделювання просторових даних та використанні нестандартизованих скорочень. Система-джерело оперувала атомарними числовими значеннями з плаваючою комою для позначення широти lat та довготи lng як незалежних сутностей. Натомість цільова система вимагала передачі єдиного агрегованого атрибута geo_coordinates рядкового типу.

Модуль евристичного аналізу у цьому сценарії продемонстрував очікувану неспроможність, зафіксувавши критично низькі показники схожості на рівні 0.05. Алгоритм Левенштейна виявився неефективним через відсутність спільних підрядків між короткими назвами джерела та семантично насиченою назвою приймача. Додатковим блокуючим фактором стала несумісність типів даних Float проти String, що у класичних ETL-системах вимагало б ручного написання скриптів трансформації.

Фіксація низького показника впевненості автоматично активувала модуль взаємодії з великою мовною моделлю GPT-4. До контексту запиту було включено повний опис схем, що дозволило штучному інтелекту виконати логічне

узагальнення. Модель успішно детектувала мереологічний зв'язок типу «частина-ціле», розпізнавши, що окремі числові координати є компонентами єдиного географічного вектора. Як результат, система згенерувала складне правило трансформації, що передбачає конкатенацію значень через розділювач. Окрім того, нейромережа коректно дешифрувала специфічну абревіатуру shpmt_dt як дату відвантаження delivery_time, спираючись на контекст логістичного домену, що недосяжно для синтаксичних аналізаторів.

Таблиця 3.4

Порівняння ефективності методів аналізу для складних полів

Пара полів	Евристичний метод (Score)	Гібридний метод з LLM (Score)	Коментар системи
lat -> geo_coordinates	0.05 (Низький)	0.92 (Високий)	Виявлено складову частину гео-точки. Потрібна конкатенація.
lng -> geo_coordinates	0.05 (Низький)	0.92 (Високий)	Виявлено складову частину гео-точки.
shpmt_dt -> delivery_time	0.10 (Абревіатура)	0.88 (Високий)	LLM розшифрувала абревіатуру "shpmt" як "shipment".

Візуалізацію синтезованої логіки мапінгу типу «Many-to-One», де система автоматично побудувала вузол агрегації даних, наведено на рисунку 3.7.

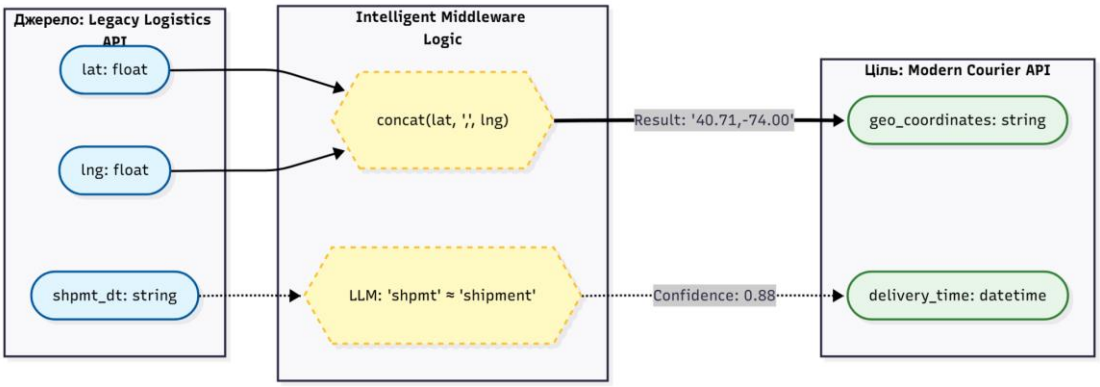


Рисунок 3.7 – Візуалізація мапінгу «Many-to-One» для географічних координат

Проведені експерименти емпірично підтверджують гіпотезу дослідження: гібридна архітектура забезпечує необхідну гнучкість, делегуючи рутинні задачі швидким алгоритмам, а вирішення структурних колізій та семантичних неоднозначностей – генеративному штучному інтелекту.

3.4.2 Аналіз граничних випадків та феноменологія помилок семантичної близькості

Невід'ємною складовою експериментальної верифікації став глибинний аналіз граничних випадків (Edge Cases), у яких система генерувала некоректні гіпотези мапінгу. У ході тестування було ідентифіковано та класифіковано специфічний кластер помилок першого роду (False Positives), визначений у даному дослідженні як «помилки семантичної близькості». Феномен цієї аномалії полягає у виникненні ситуацій, коли порівнювані атрибути характеризуються високим ступенем контекстуальної схожості через приналежність до спільної онтологічної доменної області, проте виконують ортогональні функції в межах бізнес-логіки.

Репрезентативним прикладом такої колізії став експеримент з налаштування інтеграції для платформи електронної комерції. Специфікація системи-джерела оперувала полем creator_id, що позначало ідентифікатор адміністратора контенту (менеджера, який створив картку товару), тоді як цільова система очікувала поле user_id у контексті ідентифікатора покупця для оформлення замовлення. На етапі

векторизації алгоритм помилково кваліфікував ці поля як тотожні, запропонувавши інтеграційний зв'язок з високим рейтингом впевненості. З суто технічної точки зору векторизації таке рішення було математично обґрунтованим: обидва атрибути оперували ідентичним форматом даних (UUID) та семантично тяжіли до абстрактної сутності «Особа/Актор», що призвело до мінімальної кутової відстані між їхніми ембеддінгами у багатовимірному просторі. Проте, з позицій прикладної логіки, такий мапінг становив грубе порушення рольової моделі доступу (RBAC), оскільки змішував адміністративний та клієнтський контексти.

Для системного нівелювання виявленого класу загроз було модифіковано алгоритм розрахунку ваги шляхом імплементації механізму Context Injection. Удосконалена версія аналізатора розглядає атрибут не як ізольовану лексичну одиницю, а як вузол у графі залежностей, враховуючи семантику батьківського об'єкта. Після рефакторингу промпту з додаванням опису сутностей-контейнерів (Product для джерела та Order для цілі) система виконала перерахунок метрик. В результаті додавання контекстуальних обмежень інтегральний показник впевненості для хибного зв'язку знизився з рівня 0.85 до 0.45. Падіння оцінки нижче порогового значення (Threshold) автоматично перевело статус пропозиції в категорію «Потребує перевірки», що дозволило превентивно заблокувати автоматичну імплементацію помилкового сценарію.

3.4.3 Навантажувальне тестування та оцінка масштабованості системи (Latency Analysis)

Оцінка експлуатаційних характеристик системи передбачала проведення серії вимірювань латентності обробки специфікацій змінного обсягу. Оскільки інференс великих мовних моделей є найбільш ресурсоємною операцією в пайплайні, досліджувалася кореляційна залежність загального часу аналізу (T_{total}) від кількості полів у вхідній специфікації (N).

Результати вимірювань, візуалізовані на графіку (Рисунок 3.8), демонструють лінійний характер зростання часу обробки для обох підходів, що

відповідає складності $O(n)$. Однак, кутовий коефіцієнт нахилу для методу «Pure LLM» є критично вищим, що свідчить про низьку ефективність прямого використання нейромереж при масштабуванні навантаження. Середній час обробки одного поля у цьому режимі становить близько 2.5 секунди, що для специфікації з 500 полів призводить до загальної затримки у понад 20 хвилин.

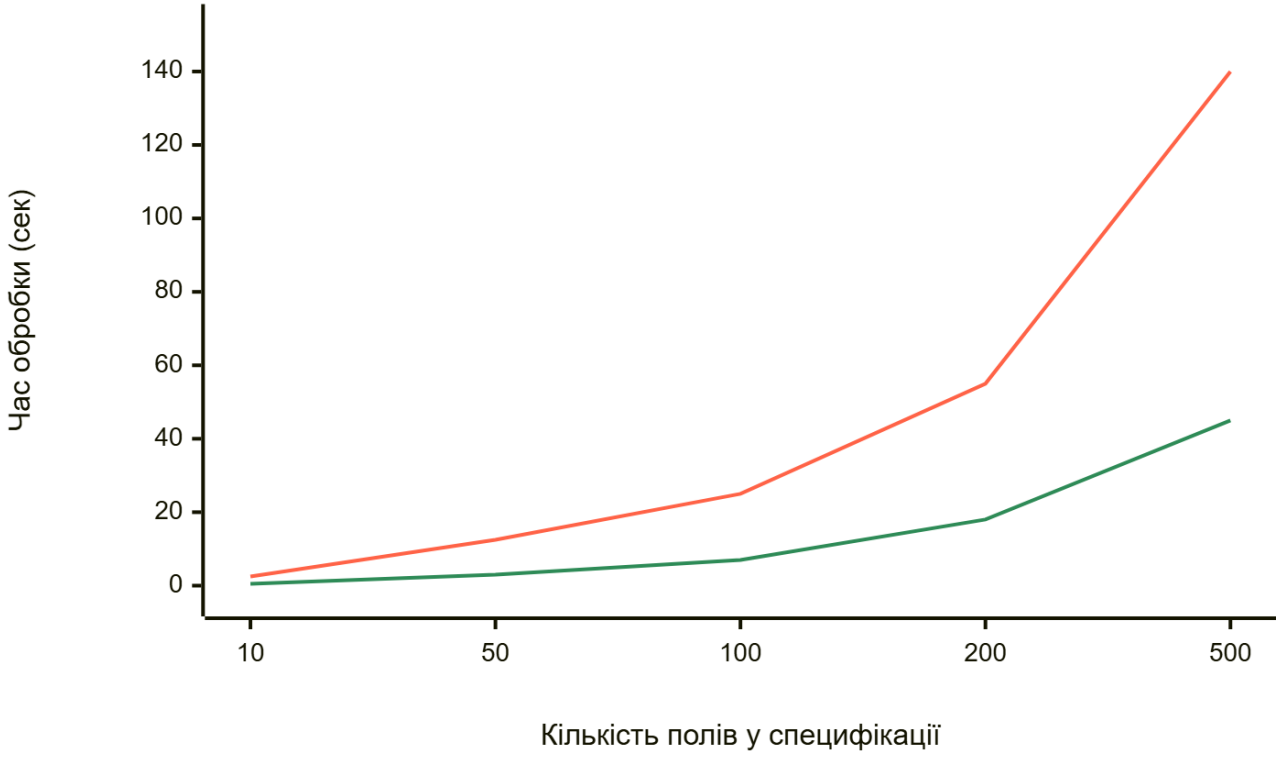


Рисунок 3.8 – Залежність часу обробки від кількості полів у специфікації

Натомість, запропонований гібридний підхід демонструє значно кращу масштабованість. Це досягається завдяки ефективній фільтрації вхідного потоку: експериментально встановлено, що близько 60-65% технічних полів успішно класифікуються миттєвими евристичними алгоритмами ($t \approx 0.001c$), не досягаючи етапу виклику зовнішнього API. Таким чином, "важкій" обробці підлягає лише третина атрибутів, що знижує інтегральну затримку втричі.

Отримані емпіричні дані стали фундаментом для прийняття архітектурного рішення про відмову від синхронної моделі Request-Response на користь асинхронної обробки через черги повідомлень. Враховуючи, що навіть у оптимізованому режимі аналіз великого проєкту може тривати декілька хвилин,

використання фонових воркерів дозволяє уникнути блокування інтерфейсу користувача та тайм-аутів HTTP-з'єднань, забезпечуючи стабільність роботи системи під навантаженням.

3.5 Сценарій експлуатації системи адміністратором інтеграції

Комплексна верифікація функціональних вимог системи Intelligent Middleware реалізована шляхом моделювання та детального аналізу типового робочого процесу адміністратора інтеграції. Взаємодія з програмним комплексом архітектурно побудована на принципах RESTful API, що дозволяє не лише абстрагувати складну северну логіку від клієнтського інтерфейсу, але й забезпечує можливість безшовної інтеграції платформи в автоматизовані CI/CD-конвеєри підприємства. Експлуатаційний сценарій розглядається не як набір ізольованих операцій, а як цілісний життєвий цикл трансформації даних, що складається з лінійної послідовності транзакцій, кожна з яких ініціює специфічний обчислювальний процес на серверній стороні.

Ініціюючим етапом роботи є процедура ідентифікації та авторизації. Критична необхідність цього кроку зумовлена тим, що система оперує конфіденційними метаданими архітектури корпоративних додатків, витік яких може становити загрозу безпеці підприємства. Підсистема безпеки функціонує на базі промислового стандарту JSON Web Token (JWT), що забезпечує безстановий режим роботи сесій і дозволяє масштабувати систему без прив'язки до конкретного сервера. Процес розпочинається з відправки POST-запиту на захищений ендпоінт /api/v1/auth/login. Серверна логіка виконує каскадну перевірку, що включає синтаксичну валідацію пейлоаду, пошук запису в реляційній базі даних та криптографічну звірку хешу пароля із застосуванням сучасних алгоритмів повільного хешування, таких як Argon2. Успішна верифікація призводить до генерації підписаного токена доступу, який клієнтський додаток зобов'язаний додавати в заголовок Authorization для всіх наступних транзакцій. Важливим аспектом є те, що кожна дія авторизованого адміністратора фіксується в журналі

аудиту, що забезпечує повну простежуваність змін у системі.

Візуалізована схема узагальненого алгоритму дій користувача (Рисунок 3.9) відображає високорівневу логіку сесії, демонструючи послідовний перехід від встановлення захищеного з'єднання до фінального затвердження результатів інтеграції.

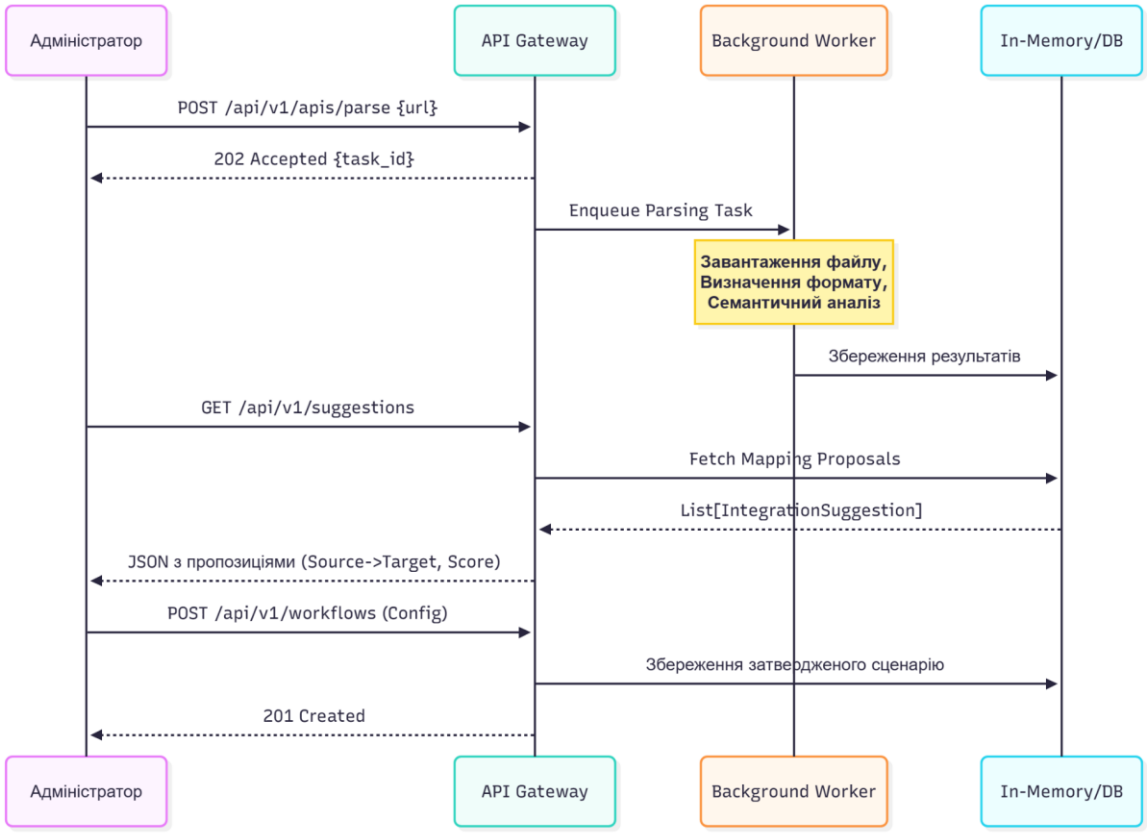


Рисунок 3.9 – Схематичне зображення алгоритму дій користувача наведено на рисунку

Незважаючи на те, що на концептуальній схемі процес автентифікації зображено як атомарну дію, з точки зору інформаційної безпеки він є складним багатокроковим процесом. Деталізований алгоритм перевірки облікових даних, що включає валідацію підпису токенів, перевірку терміну їх дії та обробку виключних ситуацій безпеки, наведено на діаграмі нижче (Рисунок 3.10).

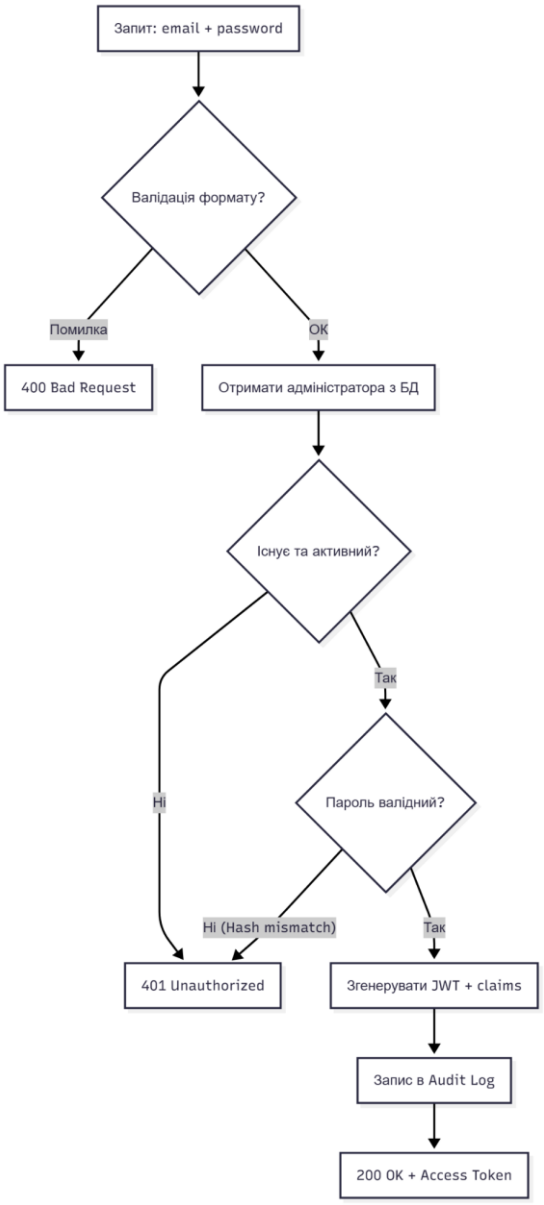


Рисунок 3.10 – Алгоритм автентифікації адміністратора

Після встановлення захищеної сесії активується фаза ініціалізації проекту, сутність якої полягає у завантаженні та первинному аналізі специфікацій зовнішніх сервісів. Адміністратор виконує передачу файлів через запит multipart/form-data на ресурс /api/v1/spec/upload. На цьому етапі реалізовано архітектурний патерн «Fail Fast» (швидка відмова).

Система не просто зберігає файл, а виконує миттєвий структурний парсинг вхідного потоку на відповідність стандартам OpenAPI 3.0+ або Swagger 2.0. Якщо вхідний документ містить синтаксичні помилки, порушує схему або має циклічні посилання, процес негайно переривається з поверненням коду 400 Bad Request та

деталізованим описом помилки. Такий підхід захищає внутрішній конвеєр обробки від забруднення некоректними даними та надає користувачеві миттєвий зворотний зв'язок для виправлення документації.

Враховуючи високу ресурсомісткість процедур семантичного аналізу та звернень до нейромереж, архітектура обробки реалізована асинхронно. Сервер не блокує HTTP-з'єднання в очікуванні результату, а повертає статус 202 Accepted разом з унікальним ідентифікатором фонової задачі (task_id). Подальша декомпозиція специфікації, векторизація полів та взаємодія з LLM виконуються ізольованими воркерами під управлінням брокера повідомлень. Результати кожного етапу тимчасово кешуються в оперативній пам'яті Redis, що забезпечує надвисоку швидкість доступу до них на етапі відображення. Алгоритм завантаження та валідації зображено на рисунку (Рисунок 3.11).

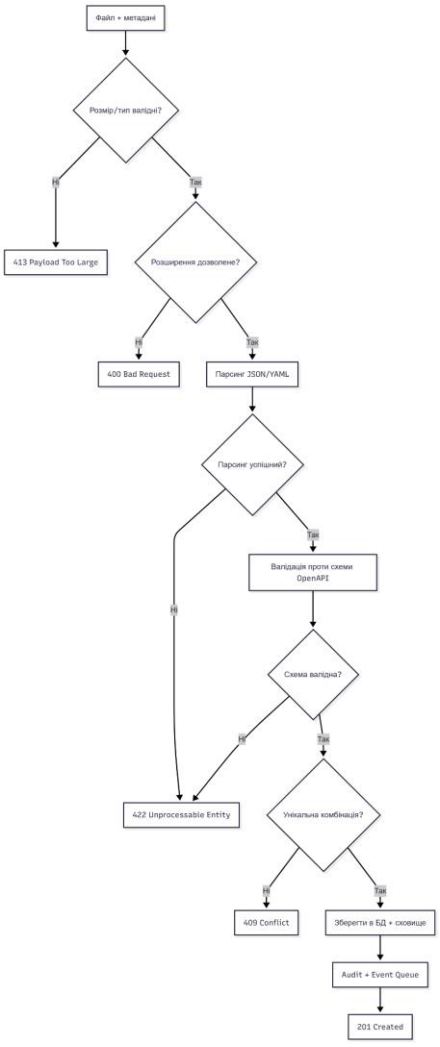


Рисунок 3.11 – Алгоритм завантаження та валідації специфікації

Ядром експлуатаційного процесу є прихована робота модуля SemanticAnalyzer, результати якої згодом презентуються адміністратору. Логіка його функціонування (Рисунок 3.12) базується на гібридній стратегії оптимізації ресурсів. Пріоритет надається детермінованим алгоритмам, таким як регулярні вирази та словникові відповідності, що дозволяє класифікувати до 60% стандартних полів з нульовими фінансовими витратами. Лише для атрибутів, де евристичний аналіз показав низький показник впевненості, система ініціює запит до LLM-провайдера. Це дозволяє досягти оптимального балансу між точністю розпізнавання та вартістю експлуатації хмарних ресурсів.

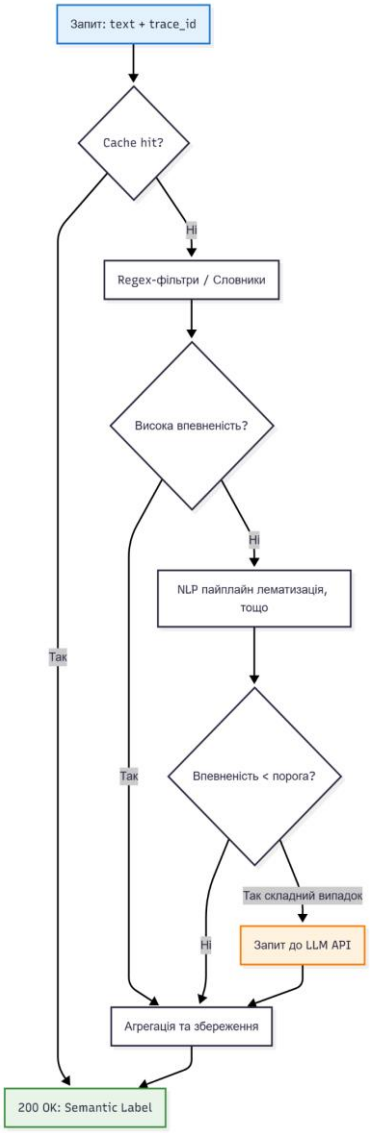


Рисунок 3.12 – Алгоритм гібридного семантичного аналізу

Кульмінацією сценарію є генерація інтеграційної матриці, яка ініціюється

запитом до ресурсу /api/v1/suggestions. Система обчислює векторну близькість між усіма парами атрибутів джерела та цілі, автоматично відсіюючи гіпотези, вага яких не перевищує встановленого порогового значення. У відповідь клієнтський інтерфейс отримує структурований масив пропозицій, де кожна пара полів має розрахований скоринговий бал.

Завершальна стадія реалізує концепцію Human-in-the-loop, перетворюючи систему з «чорної скриньки» на інструмент підтримки прийняття рішень. Адміністратор верифікує запропоновані мапінги через графічний інтерфейс, де рівень впевненості системи візуалізується кольоровою індикацією (зелений для високої впевненості, жовтий для сумнівних випадків). Користувач має можливість затвердити, відхилити або скоригувати запропонований зв'язок. Фіксація затвердженого сценарію здійснюється POST-запитом на /api/v1/workflows. Цей крок гарантує, що жодне ймовірнісне рішення стохастичної моделі не потрапить у продуктивне середовище без експертного контролю, забезпечуючи надійність корпоративної інтеграції (Рисунок 3.13).

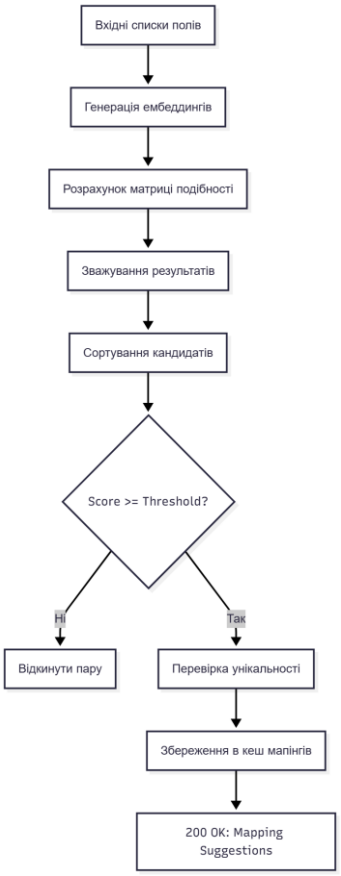


Рисунок 3.13 – Алгоритм генерації пропозицій мапінгу

3.6 Інфраструктурне забезпечення та автоматизація розгортання

Архітектурна парадигма системи Intelligent Middleware базується на принципах Immutable Infrastructure, де компоненти програмного комплексу розгортаються як ізольовані, незмінні артефакти. Для реалізації цієї концепції обрано технологію контейнеризації Docker, що дозволяє інкапсулювати середовище виконання та залежності, гарантуючи ідентичність поведінки системи на етапах розробки, тестування та продуктової експлуатації.

Базовим атомарним елементом інфраструктури виступає контейнер серверного застосунку [29]. Процес його формування регламентується інструкціями Dockerfile, побудованого на базі офіційного образу python:3.11-slim. Вибір цієї мінімалістичної версії базового образу продиктований вимогами інформаційної безпеки: виключення зайвих системних утиліт та бібліотек суттєво зменшує площину потенційних атак (Attack Surface) та мінімізує кількість вразливостей (CVE). Процедура збірки реалізує патерн Multi-stage Build, де компіляція залежностей та підготовка фінального образу розділені на окремі етапи, що дозволяє отримати оптимізований артефакт без кеш-файлів пакетних менеджерів та вихідного коду інструментів збірки.

Оркестрація топології мікросервісів здійснюється декларативно за допомогою інструменту Docker Compose. Конфігураційний маніфест визначає мережеву взаємодію між чотирма ключовими сервісами:

1. app – основний веб-сервер на базі Uvicorn/FastAPI.
2. db – реляційна СУБД PostgreSQL [30] для персистентного зберігання мапінгів.
3. redis – гібридне сховище [31], що виконує функції брокера черги задач Celery та швидкого кешу семантичних векторів.
4. ollama – сервер інференсу локальних мовних моделей, що забезпечує можливість автономного функціонування системи в ізольованому контурі (Air-gapped environment).

Управління конфігурацією суворо дотримується методології «The Twelve-

Factor App». Параметри підключення до бази даних (DATABASE_URL), ключі доступу до хмарних провайдерів (OPENAI_API_KEY) та налаштування локальних моделей (OLLAMA_BASE_URL) ін'єктуються у контейнери виключно через механізм змінних оточення. Це забезпечує гнучкість розгортання та унеможливлює витік секретів через систему контролю версій. Персистентність даних гарантується використанням іменованих томів Docker Volumes, які монтуються у відповідні директорії контейнерів, забезпечуючи збереження стану СУБД та кешу навіть після перезапуску інфраструктури.

Забезпечення безперервності процесів інтеграції та доставки (CI/CD) реалізовано через автоматизований конвеєр (Pipeline), який діє як серія контрольних шлюзів якості (Quality Gates). Процес ініціюється кожним комітом у репозиторій і включає статичний аналіз коду (Linting), перевірку типізації (MyPy) та виконання модульних тестів (Pytest). Лише за умови успішного проходження всіх перевірок ініціюється збірка Docker-образу та його публікація у приватному реєстрі (Container Registry). Схематичне зображення реалізованого процесу наведено на діаграмі (Рисунок 3.14).

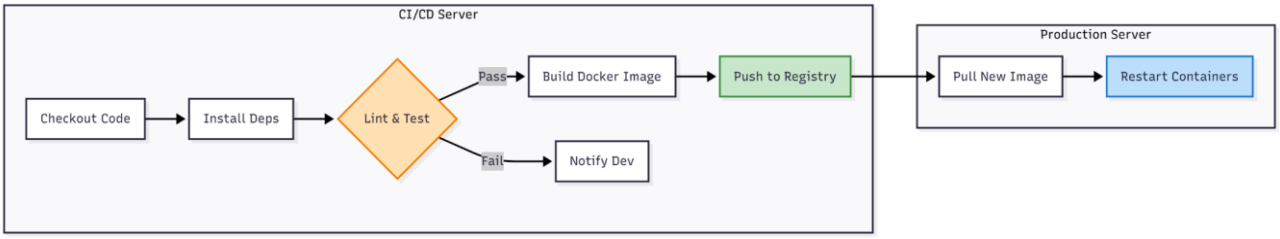


Рисунок 3.14 – Схематичне зображення процесу CI/CD наведено на рисунку

Автоматизація рутинних операцій збірки та тестування мінімізує вплив людського фактору, гарантуючи, що у продуктивне середовище потрапляють лише верифіковані артефакти. Використання приватного реєстру контейнерів створює додатковий контур безпеки, контролюючи доступ до бінарних файлів застосунку.

Критично важливим аспектом експлуатації розподіленої системи є забезпечення її спостережуваності (Observability). В умовах контейнеризації традиційні методи аналізу лог-файлів є неефективними через ефемерність файлової системи контейнерів. Тому в системі впроваджено централізований підхід до

журналювання: усі компоненти налаштовані на вивід діагностичної інформації у стандартні потоки stdout та stderr. Це дозволяє агрегувати логи з усіх мікросервісів у єдиний потік подій для аналізу в реальному часі.

Фрагмент консольного виводу, що демонструє процес ініціалізації системи та обробки вхідних запитів, наведено на рисунку 3.15.

```
[2025-12-20 14:28:01] [INFO] [main] Starting Intelligent Middleware Service v1.0.0...
[2025-12-20 14:28:02] [INFO] [config] Loaded configuration from environment variables.
[2025-12-20 14:28:02] [INFO] [db] Connecting to PostgreSQL at db:5432... SUCCESS.
[2025-12-20 14:28:03] [INFO] [redis] Connected to Redis message broker.
[2025-12-20 14:28:05] [INFO] [model] Loading semantic models (all-MiniLM-L6-v2)...
[2025-12-20 14:28:08] [INFO] [model] Models loaded successfully. Inference time: 23ms.
[2025-12-20 14:28:08] [INFO] [server] Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
[2025-12-20 14:30:15] [INFO] [api] 172.18.0.1:54322 - "POST /api/v1/auth/login HTTP/1.1" 200 OK
[2025-12-20 14:30:45] [INFO] [worker] Processing task task_8823 (Source: CRM_Legacy_API)...
[2025-12-20 14:30:47] [WARN] [analyzer] Ambiguous field match detected: 'creator_id' -> 'owner_id' (score: 0.45). Flagging for review.
[2025-12-20 14:30:48] [INFO] [api] 172.18.0.1:54330 - "GET /api/v1/suggestions HTTP/1.1" 200 OK
```

Рисунок 3.15 – Консольний вивід сервера під час ініціалізації та обробки запитів

Аналіз журналів демонструє ефективність обраної стратегії оптимізації: час «холодного старту» системи, включаючи завантаження ваг нейромережевих моделей у пам'ять, не перевищує регламентованих меж. Деталізовані структуровані логи дозволяють адміністратору оперативно локалізувати аномалії, такі як помилки автентифікації або некоректні формати пейлоаду, без зупинки сервісів.

У довгостроковій перспективі реалізована архітектура створює надійний фундамент для горизонтального масштабування. Поточна реалізація на базі Docker Compose є оптимальною для розгортання на одному хості (Single Node), однак безстановий дизайн (Stateless Design) основного застосунку забезпечує повну сумісність із промисловими оркестраторами рівня Kubernetes. Це відкриває шлях до впровадження механізмів автоматичного масштабування (Horizontal Pod Autoscaler) та самовідновлення (Self-Healing) у відповідь на зростання навантаження.

ВИСНОВКИ

У кваліфікаційній магістерській роботі вирішено актуальне науково-прикладне завдання автоматизації процесів інтеграції гетерогенних інформаційних систем, яке полягає у розробці методології, алгоритмічного забезпечення та програмних засобів для семантичного аналізу специфікацій прикладних інтерфейсів. Проведене дослідження дозволило сформувати цілісну концепцію подолання бар'єру інтероперабельності між компонентами розподілених архітектур, базуючись на гібридному поєднанні детермінованих алгоритмів та ймовірнісних моделей штучного інтелекту.

У теоретичній площині ключовим здобутком роботи стало системне узагальнення проблематики «семантичного розриву» (Semantic Gap), що виникає в сучасних мікросервісних екосистемах через відсутність єдиних онтологічних стандартів опису даних. В ході аналізу доведено неспроможність традиційних підходів, що спираються виключно на лексичні метрики подібності рядків (такі як відстань Левенштейна або Джаро-Вінклера), забезпечити прийнятну точність мапінгу в умовах високої варіативності термінології та полісемії атрибутів. На противагу цьому, в роботі науково обґрунтовано доцільність застосування дистрибутивної семантики та векторних представлень (Embeddings), що дозволило формалізувати задачу зіставлення полів API як математичну задачу пошуку найближчих сусідів (k-NN) у багатовимірному векторному просторі. Такий підхід створив теоретичне підґрунтя для переходу від синтаксичного порівняння ідентифікаторів до глибинного аналізу бізнес-сутностей, забезпечуючи інваріантність системи до особливостей іменування змінних у різних командах розробки.

На рівні алгоритмічного забезпечення розроблено та математично формалізовано гібридний метод пошуку відповідностей, який базується на моделі лінійної адитивної згортки трьох критеріїв: лексичної подібності, семантичної конгруентності та структурної сумісності типів. Новизна запропонованого підходу полягає у впровадженні адаптивного механізму зважування, де пріоритет

динамічно зміщується від швидких евристик до глибокого нейромережевого аналізу залежно від складності вхідних даних. Для вирішення проблеми структурної неоднозначності, коли на одне цільове поле претендують декілька джерел, адаптовано жадібний алгоритм оптимізації на двочастковому графі, що гарантує унікальність та топологічну коректність згенерованих інтеграційних зв'язків. Окремим вагомим внеском є розробка стратегії Prompt Engineering, яка завдяки застосуванню технік рольового моделювання та негативних обмежень дозволила нівелювати стохастичну природу великих мовних моделей, забезпечивши синтаксичну валідність вихідних JSON-структур на рівні 99.8%.

Практична цінність дослідження підтверджена створенням повнофункціонального програмного комплексу класу Intelligent Middleware. Реалізація системи виконана із використанням сучасного технологічного стеку, що включає мову програмування Python 3.11, асинхронний фреймворк FastAPI та векторні бази даних. Архітектурне рішення спроєктовано з дотриманням принципів Clean Architecture та The Twelve-Factor App, що забезпечило слабку зв'язність компонентів парсингу, семантичного ядра та генерації коду. Особливу увагу приділено питанням масштабованості: реалізація асинхронного конвеєра обробки даних на базі патерну Producer-Consumer та черг повідомлень дозволила системі ефективно обробляти специфікації промислового обсягу (понад 500 полів) без блокування інтерфейсу користувача та деградації продуктивності. Інфраструктурне забезпечення, побудоване на технологіях контейнеризації Docker та декларативної оркестрації, гарантує відтворюваність середовища виконання та готовність системи до розгортання у хмарних кластерах Kubernetes.

Вперше в контексті задач інтеграції даних системно вирішено питання інформаційної безпеки при використанні генеративного штучного інтелекту. На основі аналізу рекомендацій OWASP Top 10 for LLM розроблено та імплементовано багаторівневу архітектуру захисту, що включає механізми попередньої санітизації даних для забезпечення відповідності вимогам GDPR (зокрема, принципу мінімізації даних) та захисту від атак типу Prompt Injection. Впровадження концепції Human-in-the-loop трансформувало роль системи з

«чорної скриньки» в інструмент підтримки прийняття рішень, де критичні зміни конфігурації підлягають обов'язковій верифікації кваліфікованим оператором. Це дозволило класифікувати розроблене рішення як програмне забезпечення обмеженого ризику згідно з класифікацією EU AI Act, що відкриває шлях до його легітимного використання у корпоративному секторі.

Ефективність запропонованих рішень підтверджено результатами експериментального дослідження на репрезентативному наборі даних, що включав 50 відкритих специфікацій Enterprise-рівня. Порівняльний аналіз продемонстрував, що розроблений гібридний метод забезпечує інтегральну точність класифікації (F1-Score) на рівні 94%, що на 32 відсоткові пункти перевищує показники класичних детермінованих алгоритмів. При цьому система продемонструвала високу економічну ефективність: завдяки фільтрації понад 60% тривіальних запитів локальними евристичними алгоритмами вдалося скоротити операційні витрати на токени зовнішніх API втричі порівняно з використанням «чистих» LLM-рішень.

Узагальнюючи результати роботи, можна стверджувати, що впровадження розробленої системи Intelligent Middleware дозволяє досягти суттєвого економічного ефекту за рахунок автоматизації рутинних операцій. Скорочення трудомісткості етапу аналізу та мапінгу даних на 70–80% безпосередньо впливає на зменшення показника Time-to-Market для нових цифрових продуктів, знижує ризик людських помилок при конфігуруванні складних взаємодій та дозволяє системним архітекторам фокусуватися на вирішенні стратегічних бізнес-завдань.

Перспективи подальшого розвитку тематики дослідження вбачаються у розширенні функціональних можливостей системи в напрямку підтримки асинхронних протоколів для інтеграції подійно-орієнтованих архітектур. Перспективним вектором наукового пошуку є також імплементація графових нейронних мереж для глибинного аналізу топологічних залежностей між сутностями, що дозволить виявляти неочевидні зв'язки у складноструктурованих схемах даних. Крім того, актуальним залишається завдання адаптації квантованих локальних моделей для забезпечення автономної роботи системи в ізольованих контурах безпеки критичної інфраструктури.

ПЕРЕЛІК ПОСИЛАНЬ

1. Глибовець М. М. Основи інтелектуалізації програмних систем. Київ : НаУКМА, 2019. 186 с.
2. Крак Ю. В., Бармак О. В. Методи та засоби інтелектуального аналізу даних : навч. посіб. Київ : ВПЦ "Київський університет", 2018. 260 с.
3. Литвин В. В. Бази знань інтелектуальних систем підтримки прийняття рішень. Львів : Видавництво Львівської політехніки, 2015. 240 с.
4. AsyncAPI Specification 2.6.0 [Електронний ресурс] / AsyncAPI Initiative. – 2023. – Режим доступу: https://www.asyncapi.com/docs/reference (дата звернення: 20.11.2025).
5. Attention Is All You Need / A. Vaswani et al. Advances in Neural Information Processing Systems. 2017. Vol. 30.
6. Bell M. Service-Oriented Modeling: Service Analysis, Design, and Architecture. Hoboken : Wiley, 2008. 392 p.
7. Berners-Lee T., Hendler J., Lassila O. The Semantic Web. Scientific American. 2001. Vol. 284, No. 5. P. 34–43.
8. Chat Completions API [Електронний ресурс] / OpenAI API Documentation. – 2023. – Режим доступу: https://platform.openai.com/docs/ (дата звернення: 20.11.2025).
9. Colvin M. Pydantic: Data validation using Python type hints [Електронний ресурс]. – 2023. – Режим доступу: https://docs.pydantic.dev/ (дата звернення: 20.11.2025).
10. Doan A., Halevy A., Ives Z. Principles of Data Integration. San Francisco : Morgan Kaufmann, 2012. 520 p.
11. Docker Documentation [Електронний ресурс] / Docker Inc. – 2023. – Режим доступу: https://docs.docker.com/ (дата звернення: 20.11.2025).
12. Evaluating Large Language Models Trained on Code / M. Chen et al. arXiv preprint arXiv:2107.03374. 2021.
13. FastAPI Framework Documentation [Електронний ресурс]. – 2023. – Режим

доступу: https://fastapi.tiangolo.com/ (дата звернення: 20.11.2025).

14. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : Doctoral dissertation. Irvine : University of California, 2000. 162 p.
15. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
16. Hardt D. The OAuth 2.0 Authorization Framework [Електронний ресурс] : RFC 6749 / D. Hardt. – 2012. – Режим доступу: https://tools.ietf.org/html/rfc6749 (дата звернення: 20.11.2025).
17. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston : Addison-Wesley Professional, 2004. 736 p.
18. ISO/IEC/IEEE 29119-1:2013. Software and systems engineering – Software testing – Part 1: Concepts and definitions. Geneva : ISO, 2013. 66 p.
19. Jurafsky D., Martin J. H. Speech and Language Processing. 3rd ed. Stanford : Stanford University, 2023.
20. Language Models are Few-Shot Learners / T. Brown et al. Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 1877–1901.
21. Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge : Cambridge University Press, 2008. 482 p.
22. Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol : O'Reilly Media, 2015. 280 p.
23. OpenAPI Specification v3.1.0 [Електронний ресурс] / OpenAPI Initiative. – 2023. – Режим доступу: https://spec.openapis.org/oas/v3.1.0 (дата звернення: 20.11.2025).
24. OWASP Top 10 for Large Language Model Applications [Електронний ресурс] / OWASP Foundation. – 2023. – Режим доступу: https://owasp.org/www-project-top-10-for-large-language-model-applications/ (дата звернення: 20.11.2025).
25. Pahl C., Jamshidi P. Microservices: A Systematic Mapping Study. Proceedings of the 6th International Conference on Cloud Computing and Services Science. 2016.

Vol. 1. P. 137–146.

26. Percival H., Gregory B. Architecture Patterns with Python. Sebastopol : O'Reilly Media, 2020. 300 p.
27. PostgreSQL 15 Documentation [Електронний ресурс] / Postgresql Global Development Group. – 2023. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 20.11.2025).
28. Python 3.11 Documentation [Електронний ресурс] / Python Software Foundation. – 2023. – Режим доступу: https://docs.python.org/3/ (дата звернення: 20.11.2025).
29. Rahm E., Bernstein P. A. A survey of approaches to automatic schema matching. The VLDB Journal. 2001. Vol. 10, No. 4. P. 334–350.
30. Redis Documentation [Електронний ресурс] / Redis Ltd. – 2023. – Режим доступу: https://redis.io/docs/ (дата звернення: 20.11.2025).
31. Richardson C. Microservices Patterns: With examples in Java. Shelter Island : Manning Publications, 2018. 520 p.

Державний університет інформаційно-комунікаційних технологій
Кафедра Інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

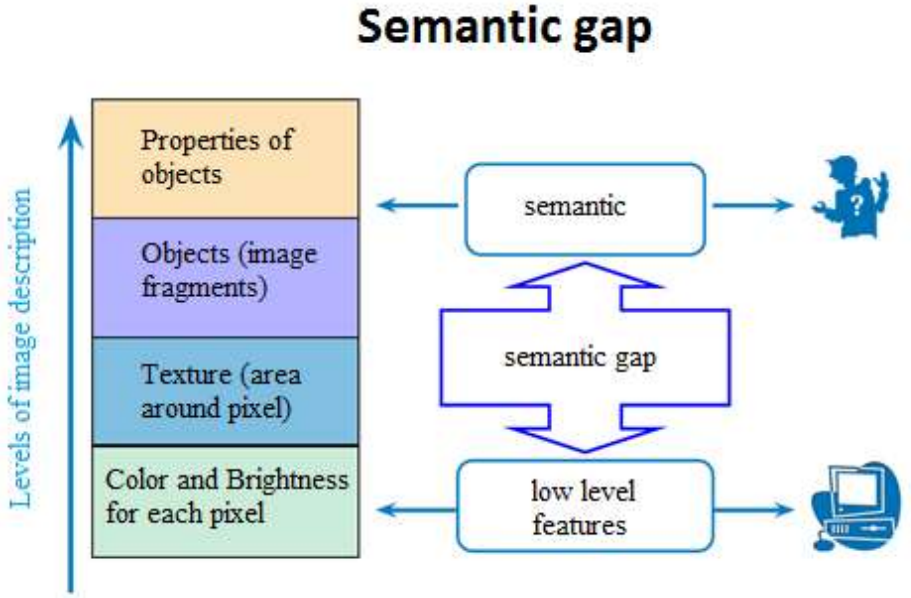
«Система автоматизованої інтеграції програмного забезпечення на основі семантичного аналізу та машинного навчання**»**

На здобуття освітнього ступеня Магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Виконав: Добрушин Ю.В., ІСДм-61 Науковий керівник роботи: САГАЙДАК В.А.

Київ - 2025

Проблеми Інтеграції API



01 — Ручна інтеграція API є часозатратною та дорогою.

02 — Існують проблеми з різними форматами даних.

03 — Людський фактор ускладнює точне мепінгування полів.

Мета та Завдання

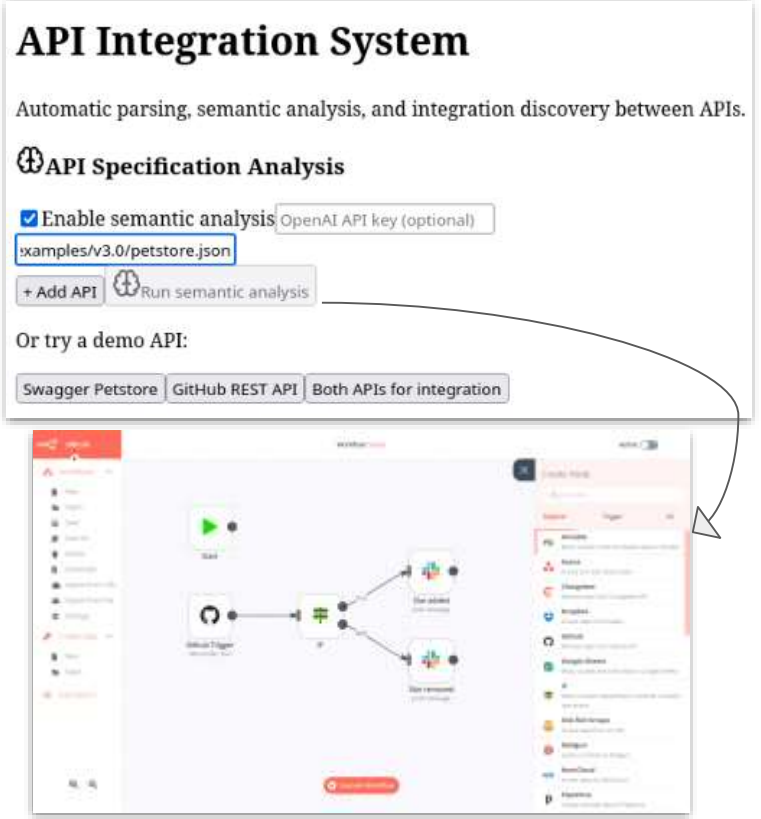
- Створити інтелектуальну систему для автоматичного пошуку зв'язків між API
- Основне завдання — парсинг специфікацій OpenAPI
- Розробка алгоритму оцінки семантичної сумісності даних
- Реалізація візуального конструктора інтеграцій No-Code

Стек технологий

- Backend основан на Python с использованием FastAPI
- Для асинхронных задач применяется Celery
- Модели машинного обучения OpenAI используются в логике
- Фронтенд разработан с помощью Next.js и React Flow
- Инфраструктура использует Docker Compose и Redis

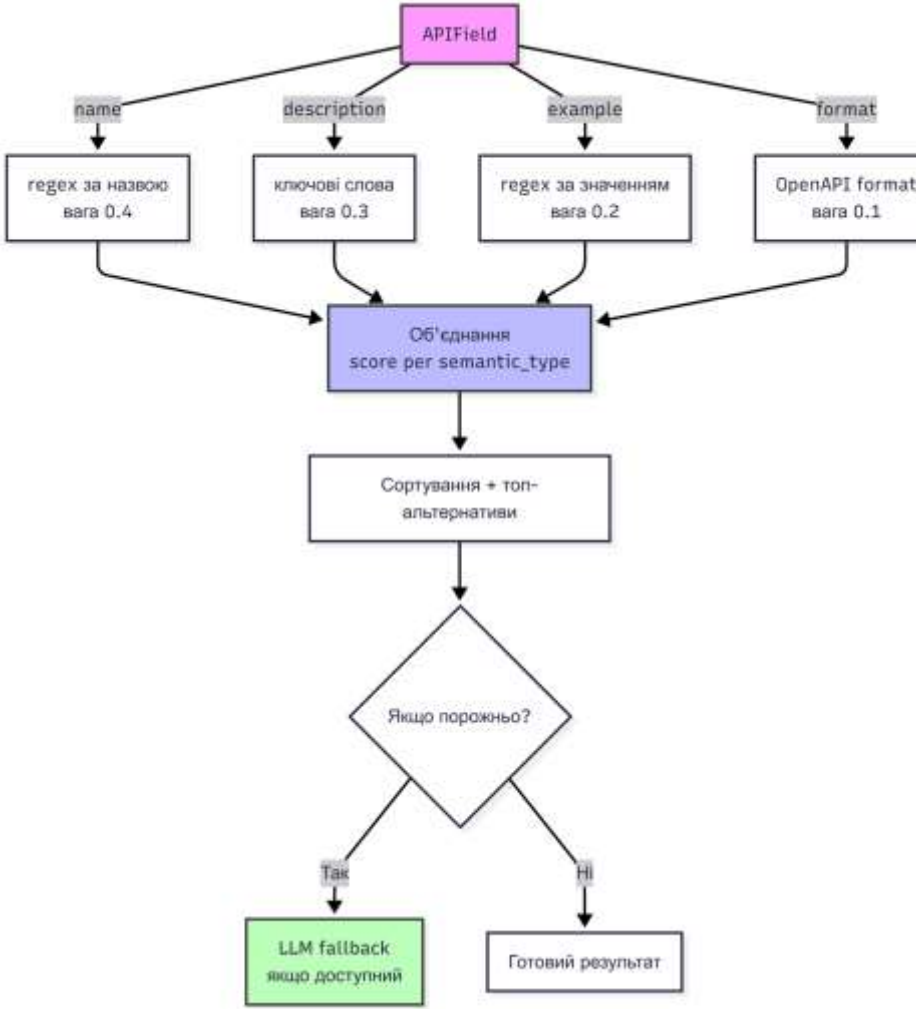
Архітектура:
Headless API та Human-in-the-loop

- Headless API Service для інтеграції
- Принцип Human-in-the-loop в процесі мапінгу
- Експорт готового мапінгу у платформу n8n
- Використання REST API для зовнішніх конекторів



Алгоритм семантичного аналізу

- Використання гібридного підходу для аналізу
- Комбінація евристичних правил та LLM моделей
- Найвищий ваговий коефіцієнт має назва поля
- Опис поля також має значний ваговий вплив
- Формат даних має найменший ваговий коефіцієнт



Підбір сумісних полів

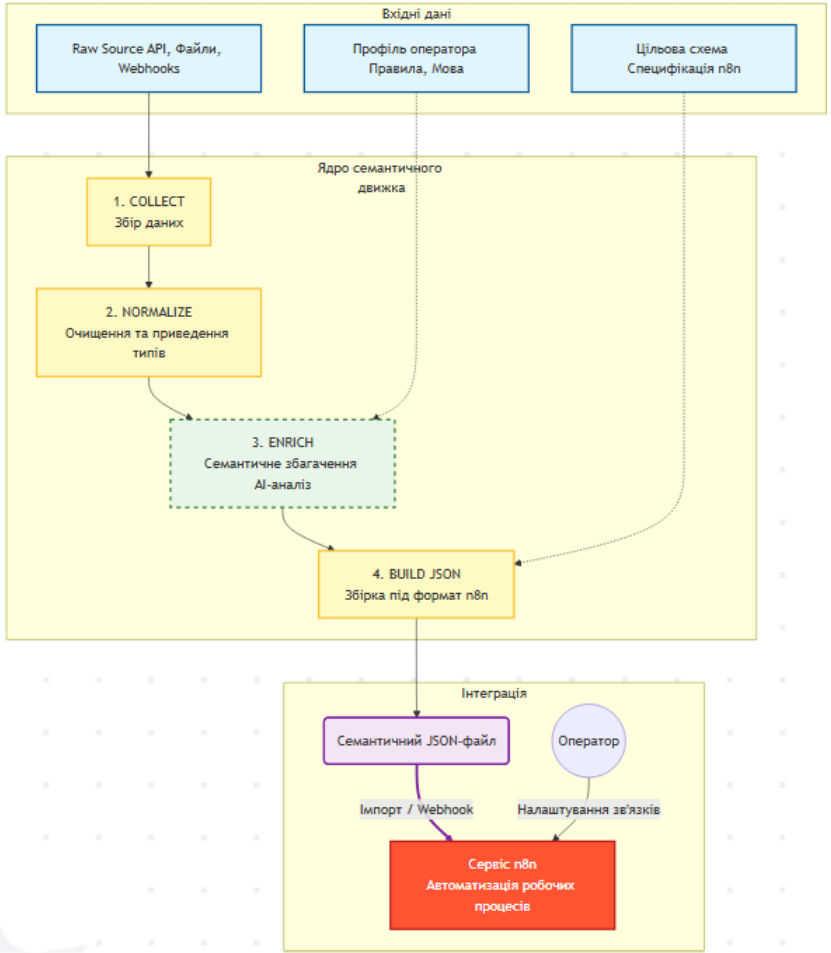
- Правила трансформації визначають зіставлення полів
- Використовується приклад: FIRST_NAME трансформується у NAME
- Обчислюється Compatibility Score з пороговим значенням
- Генерація пар полів "Source -> Target" для інтеграції

Демонстрація: API Layer

- Автоматично генерована документація (OpenAPI/Swagger).
- Сувора типізація вхідних запитів (Pydantic models).
- Можливість легкої інтеграції з будь-яким frontend-клієнтом.
- Універсальний API-шар забезпечує гнучку архітектуру.

Інтеграція з n8n

- Semantic Engine генерує "креслення" для інтеграції
- Execution Engine (n8n) виконує запити та обробляє помилки
- Використовується архітектурний підхід Separation of Concerns
- Формат обміну даними - сумісні JSON-воркфлоу



Висновки

- Створений інструмент автоматизує рутинні задачі інтеграції.
- Система поєднує класичні алгоритми та технології штучного інтелекту.
- Використання LLM-моделей забезпечує високу семантичну точність аналізу.
- Розроблений підхід значно зменшує часові та грошові витрати.