

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Центр післядипломної освіти
(повна назва)
Кафедра _____ Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

Рекомендаційна система оцінювання якості АВАР-коду на основі принципів
Clean Code із використанням великих мовних моделей
(тема)

Виконав:
здобувач _____ другого _____ року навчання,
групи _____ ІТШПз-24-1

Сергій ЯБЛУНІВСЬКИЙ
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)

Тип програми _____ освітньо-професійна

Освітня програма _____ Штучний інтелект

(повна назва освітньої програми)

Керівник _____ ст. викл. Дмитро ВОДЯНИЦЬКИЙ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ

(підпис)

Лариса ЧАЛА
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Центр післядипломної освіти _____

Кафедра _____ Штучного інтелекту _____

Рівень вищої освіти _____ перший (бакалаврський) _____

Спеціальність _____ 122 Комп'ютерні науки _____
(код і повна назва)

Тип програми _____ освітньо-професійна _____

Освітня програма _____ Штучний інтелект _____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____» _____ 20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві _____ Яблунівському Сергію Володимировичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____ Рекомендаційна система оцінювання якості ABAP-коду на основі принципів Clean Code із використанням великих мовних моделей _____

затверджена наказом університету від 18 травня 20 26 р. № 77Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 20 26 р.

3. Вихідні дані до роботи _____ науково-технічні публікації з питань якості програмного коду, документація SAP щодо ATC, Clean ABAP, ABAP AI SDK, SAP AI Core, SAP HANA Cloud Vector Engine, матеріали з оцінювання RAG-систем, приклади findings ABAP-коду _____

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі забезпечення якості ABAP-коду _____

2) Методи та засоби побудови рекомендаційної системи _____

3) Розробка архітектури, моделі даних, алгоритму роботи, сценаріїв використання та оцінювання результатів системи _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	18.05.2026 – 19.05.2026	виконано
2	Аналіз предметної галузі	19.05.2026 – 21.05.2026	виконано
3	Формування принципів Clean Code/Clean ABAP	21.05.2026 – 22.05.2026	виконано
4	Аналіз інструментів ATC, ABAP Unit, AI SDK	22.05.2026 – 23.05.2026	виконано
5	Проектування Knowledge Base	23.05.2026 – 24.05.2026	виконано
6	Проектування retrieval та recommendation pipeline	24.05.2026 – 25.05.2026	виконано
7	Розробка архітектури системи	25.05.2026 – 26.05.2026	виконано
8	Формування моделі даних	26.05.2026 – 27.05.2026	виконано
9	Перевірка на академічний плагіат	28.05.2026 – 30.05.2026	виконано
10	Нормоконтроль	01.06.2026 – 02.05.2026	виконано
11	Підготовка презентації та доповіді	02.06.2026 – 03.05.2026	виконано
12	Рецензування	03.06.2026 – 04.05.2026	виконано
13	Оформлення пояснювальної записки	04.06.2026 – 05.05.2026	виконано
14	Попередній захист роботи з керівником	07.06.2026	виконано
15	Захист перед ДЕК	16.06.2026	

Дата видачі завдання 18 травня 2026 р.

Здобувач 
(підпис)

Керівник роботи _____
(підпис)

ст. викл. Дмитро ВОДЯНИЦЬКИЙ
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 56 с., 16 рис., 17 табл., 4 дод., 22 джерела.

ЯКІСТЬ КОДУ, АВАР-КОД, АТС, CLEAN AVAR, CLEAN CODE, KNOWLEDGE BASE, LLM, RAG, SAP AI CORE, SAP HANA CLOUD VECTOR ENGINE.

Об'єкт дослідження – процеси оцінювання та підвищення якості АВАР-коду в корпоративних SAP-системах.

Предмет дослідження – методи формування рекомендацій щодо вдосконалення АВАР-коду на основі принципів Clean Code/Clean AVAR, результатів статичного аналізу та великих мовних моделей.

Мета роботи – розроблення рекомендаційної системи оцінювання якості АВАР-коду, яка на основі findings статичного аналізу та правил Clean AVAR генерує пояснювані рекомендації з використанням великих мовних моделей.

Методи дослідження – системний і порівняльний аналіз, методи статичного аналізу програмного коду, підходи Retrieval-Augmented Generation, методи інформаційного пошуку та оцінювання якості програмного забезпечення.

У межах цього дослідження сформовано модель рекомендаційної системи, що поєднує АТС як джерело фактів про порушення, Knowledge Base правил і антипатернів, retrieval-шар, recommendation engine на основі LLM та контур обов'язкової валідації через повторний запуск АТС і АВАР Unit.

ABSTRACT

Bachelor's thesis contains: 56 pp., 16 fig., 17 tabl., 4 ann., 22 references.

ABAP-CODE, ATC, CLEAN ABAP, CLEAN CODE, CODE QUALITY, KNOWLEDGE BASE, LLM, RAG, SAP AI CORE, SAP HANA CLOUD VECTOR ENGINE.

Object of study – the processes of assessing and improving the quality of ABAP code in corporate SAP systems.

Subject of study – methods for generating recommendations to improve ABAP code based on Clean Code/Clean ABAP principles, static analysis results, and large language models.

Purpose of the work – to develop a recommendation system for assessing ABAP code quality that generates explainable recommendations using large language models based on static analysis findings and Clean ABAP rules.

Research methods – systems and comparative analysis, static code analysis methods, Retrieval Augmented Generation approaches, information retrieval methods, and software quality evaluation methods.

Within this study, a model of a recommendation system has been formed that combines ATC as a source of factual information about violations, a Knowledge Base of rules and antipatterns, a retrieval layer, an LLM-based recommendation engine, and a mandatory validation loop through re-running ATC and ABAP Unit.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз предметної галузі	11
1.1 Проблема забезпечення якості АВАР-коду в сучасних SAP-системах	12
1.2 Принципи Clean Code та їх адаптація до АВАР	14
1.3 Аналіз існуючих підходів до оцінювання якості коду	15
1.4 Принципи Clean Code та їх адаптація до АВАР	18
1.5 Використання великих мовних моделей у задачах аналізу програмного коду	19
1.6 Порівняльний аналіз існуючих рішень та постановка задачі	20
2 Методи та засоби побудови рекомендаційної системи	22
2.1 Формалізація критеріїв якості АВАР-коду	22
2.2 Knowledge Base правил і антипатернів	24
2.3 Модель обробки findings: від результату АТС до рекомендації	25
2.4 Використання LLM для генерації пояснюваних рекомендацій	26
2.5 RAG - підхід та механізм зниження галюцинацій	27
2.6 Метрики оцінювання якості рекомендацій	28
2.7 Проєктування системи	30
3 Розробка рекомендаційної системи оцінювання якості АВАР-коду	31
3.1 Загальна архітектура системи	32
3.2 Модель даних та структура вхідних і вихідних об'єктів	33
3.3 Алгоритм роботи рекомендаційної системи	35
3.4 Приклади типових сценаріїв роботи системи	36
3.5 Оцінювання результатів роботи системи	37
3.6 Можливості інтеграції та подальшого розвитку	39
3.7 Опис результату та прототип інтерфейсу	39
4 Опис результату кваліфікаційної роботи	41

4.1	Склад і характеристика отриманих результатів	41
4.2	Архітектура рекомендаційної системи	42
4.3	Прототип інтерфейсу користувача: findings та рекомендації	43
4.4	Прототип інтерфейсу користувача: керування Knowledge Base	44
4.5	Приклад наскрізного сценарію та формат результату.....	44
	Висновки	47
	Перелік джерел посилання	49
	Додаток А Структуровані дані системи (Rule Cards та Prompt Contracts) ..	52
	Додаток Б Вихідний код прототипу (Python).....	54
	Додаток В Макети екранних форм прототипу	55
	Додаток Г Відомість кваліфікаційної роботи.....	56

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ABAP – Advanced Business Application Programming – поглиблене програмування бізнес-застосунків;

ADT – ABAP Development Tools – інструменти розробки ABAP;

API – Application Programming Interface – інтерфейс прикладного програмування;

ATC – ABAP Test Cockpit – інструмент тестового кокпіту ABAP (статичний аналіз якості коду);

BTP – Business Technology Platform – бізнес-технологічна платформа;

CI/CD – Continuous Integration/Continuous Delivery – безперервна інтеграція / безперервне постачання (доставка);

JSON – JavaScript Object Notation – нотація об'єктів JavaScript (текстовий формат обміну даними);

KB – Knowledge Base – база знань;

LLM – Large Language Model – велика мовна модель;

RAG – Retrieval-Augmented Generation – генерація з підкріпленням пошуком (пошуково-орієнтована генерація);

SCI – Code Inspector – інспектор коду;

SAP AI Core – SAP AI Core – сервіс SAP BTP для виконання та операційного супроводу AI-активів.

ВСТУП

У сучасних умовах цифрової трансформації підприємств прикладні рішення на платформі SAP залишаються критично важливими для автоматизації фінансових, логістичних, виробничих і кадрових процесів. Так значна частина прикладної логіки таких систем реалізується мовою АВАР, тобто якість АВАР-коду безпосередньо впливає на надійність, продуктивність, підтримуваність та вартість супроводу корпоративних інформаційних систем [1], [2], [3].

У великих SAP-ландшафтах недоліки якості коду не обмежуються синтаксичною коректністю. Натомість навіть працездатний АВАР-код може містити антипатерни, які погіршують читабельність, ускладнюють тестування, збільшують технічний борг та створюють ризики під час модернізації системи. Це, у свою чергу стимулює до принципу Clean Code, адаптовані до специфіки АВАР у вигляді Clean АВАР, та вони розглядаються як нормативна база підтримуваного й надійного коду [4].

Інструменти статичного аналізу, насамперед АВАР Test Cockpit, добре виявляють недоліки конструкції, але не завжди надають зрозуміле пояснення причин порушення, пріоритетних способів виправлення та їх зв'язку з правилами Clean АВАР [1], [3].

Інформаційні системи забезпечують зберігання, обробку, пошук, розподіл, передавання та надання інформації, що сприяє підвищенню ефективності організації, спрощенню доступу до необхідних даних і гарантуванню надійності та точності обробки інформації. Це, у свою чергу, стимулює розвиток цифрового суспільства та інноваційної економіки.

Сучасний розвиток великих мовних моделей відкриває можливість використовувати їх як інтелектуальний шар над статичним аналізом: не для встановлення факту порушення, а для формування пояснених рекомендацій, прикладів рефакторингу та чеклістів валідації. SAP уже підтримує AI-based capabilities для АВАР-розробки та АВАР AI SDK для

інтеграції generative AI у власні застосунки [5], [6], [7].

Водночас пряме використання LLM без опори на доменні правила пов'язане з ризиком галюцинацій і семантично небезпечних змін. Тому для вирішення зазначених проблем в роботі ми будемо обґрунтовувати доцільність RAG-підходу, та модель натомість отримує не лише finding і кодовий контекст, а й релевантні правила з контрольованої бази знань [8], [9], [10], [11].

У межах даної роботи досліджується розроблення та інтеграції в процес АВАР-розробки рекомендаційної системи, оснащеної модулем на основі великих мовних моделей і механізмом Retrieval-Augmented Generation (RAG), пропонується суттєво вдосконалити традиційні підходи до контролю якості коду. Особлива увага приділяється та реалізується в автоматизації рутинних етапів аналізу результатів статичної перевірки (АТС), інтерпретації finding-повідомлень і підготовки практичних кроків їх усунення відповідно до принципів Clean Code/Clean АВАР. Система виконує інтелектуальну обробку фрагментів АВАР-коду в контексті конкретного порушення, зіставляє його з правилами та прикладами з контрольованої бази знань (Knowledge Base) і формує пояснювані рекомендації щодо рефакторингу з урахуванням обмежень середовища (зокрема clean core) та ризику зміни семантики.

Отже, дана робота має значний потенціал для практичного застосування. Це дає змогу в режимі реального часу отримувати обґрунтовані підказки щодо покращення коду (наприклад, оптимізації доступу до БД, зменшення складності, покращення обробки помилок, підвищення тестованості), а також забезпечує стандартизований підхід до виправлення типових порушень у команді розробки. Додатково система формує чекліст валідації змін (повторний запуск АТС, виконання АВАР Unit-тестів та, за потреби, перевірки продуктивності), що дозволяє підвищити надійність внесених виправлень і зменшити ризики накопичення технічного боргу в корпоративних SAP-системах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

На сучасному етапі розвитку ІТ глобалізація бізнесу та перехід компаній до розподілених цифрових екосистем суттєво підвищують вимоги до якості програмного забезпечення. У світі корпоративні системи вже давно перестали бути «локальними» рішеннями: вони обслуговують міжнародні ланцюги постачання, фінансові операції, комплаєнс-процедури та інтегруються з великою кількістю зовнішніх сервісів. У таких умовах навіть невеликі дефекти коду можуть масштабуватися до значних операційних ризиків, а підтримуваність і передбачуваність змін стають критично важливими характеристиками.

Для SAP-ландшафтів ці тенденції проявляються особливо виразно: АВАР-код часто розвивається роками, підтримується різними командами в різних країнах, а проєкти модернізації вимагають швидкого й безпечного внесення змін без деградації продуктивності та стабільності. Водночас ринок розробки стикається з дефіцитом вузькопрофільної експертизи, через що зростає потреба у стандартизації практик (Clean Code/Clean АВАР) і в інструментах, які знижують «вартість» розуміння й виправлення типових порушень.

Саме тому предметна галузь рекомендаційних систем оцінювання якості АВАР-коду з використанням великих мовних моделей є актуальною: вона поєднує вимоги глобального масштабу, необхідність керованого контролю якості та прагнення до прискорення циклу внесення змін у корпоративних системах.

Зазначений підхід узгоджується з науково-освітніми напрямками кафедри, де представлені дослідження та навчальні дисципліни, пов'язані з інформаційними системами, Python-розробкою, аналізом даних, проєктуванням інтелектуальних систем, алгоритмами та інтелектуальним аналізом даних [19], [20], [21], [22].

1.1 Проблема забезпечення якості АВАР-коду в сучасних SAP-системах

Предметна галузь роботи охоплює підходи, методи та інструменти оцінювання і підвищення якості АВАР-коду в корпоративних SAP-ландшафтах, де АВАР традиційно виконує роль основної мови прикладної логіки, інтеграційних сценаріїв і розширень стандартної функціональності. На сучасному етапі розвитку ІТ глобалізація бізнесу та перехід організацій до розподілених цифрових екосистем суттєво підвищують вимоги до надійності програмного забезпечення. Корпоративні інформаційні системи перестали бути локальними рішеннями: вони забезпечують роботу міжнародних ланцюгів постачання, фінансових і регуляторних процедур, інтегруються з численними зовнішніми сервісами та працюють у режимах високої доступності. У таких умовах якість коду набуває системного значення, оскільки навіть невеликі дефекти або нераціональні реалізації можуть масштабуватися до операційних ризиків, підвищення витрат на підтримку й ускладнення процесів модернізації.

Формування проблематики забезпечення якості АВАР-коду має виразну історичну передумову. У багатьох організаціях кастомний АВАР-код накопичувався протягом тривалого життєвого циклу SAP-рішень, починаючи з епохи SAP R/3 і продовжуючи в період трансформації до SAP S/4HANA. Розширення стандартної функціональності створювалися під конкретні бізнес-сценарії, часто різними командами або підрядниками, з неоднорідними підходами до проєктування, стилю та тестування.

У результаті закономірно виникло явище технічного боргу: кодова база ускладнюється, зростає кількість дублювань, збільшується складність окремих модулів, накопичуються залежності, які важко підтримувати та безпечно змінювати. Цей ефект посилюється тим, що корпоративні системи функціонують у режимі безперервної еволюції: бізнес-вимоги змінюються,

інтеграції розширюються, а регуляторні вимоги змушують регулярно модифікувати процеси.

Проблематика предметної галузі проявляється, насамперед, у підтримуваності АВАР-коду, його продуктивності, робастності та платформній сумісності. З погляду підтримуваності критичними є великі та складні методи, надмірна вкладеність умов і циклів, змішування різних відповідальностей у межах одного модуля, а також неузгоджене іменування, що підвищує когнітивні витрати на розуміння коду. З погляду продуктивності типові ризики пов'язані з доступом до бази даних і обробкою великих обсягів бізнес-даних: неефективні патерни звернень, надлишкові вибірки або нераціональні операції над внутрішніми таблицями можуть призводити до деградації часу відповіді, нестабільності під навантаженням та зростання інфраструктурних витрат. Робастність визначається здатністю коду коректно працювати у нестандартних ситуаціях та зберігати передбачувану поведінку під час змін; слабка обробка помилок, неконсистентні контракти методів і недостатня тестова база створюють ризик прихованих дефектів і регресій. Окремого значення в сучасних SAP-ландшафтах набуває платформна сумісність, пов'язана з clean core-орієнтованими підходами: код має бути не лише “естетично” правильним, а й оновлюваним, сумісним із дозволеними механізмами розширення та використання релізнутих API, щоб не блокувати розвиток системи та її модернізацію.

Актуальність цієї предметної галузі визначається поєднанням технологічних і організаційних факторів. З одного боку, бізнес очікує прискорення циклу змін: нові вимоги мають реалізовуватися швидко, але без втрати стабільності та якості. З іншого боку, у багатьох організаціях відчувається дефіцит вузькопрофільної експертизи, що підсилює потребу у стандартизації практик розробки та уніфікації підходів до виправлення типових порушень. У цьому контексті принципи Clean Code та їх адаптація до АВАР у вигляді Clean АВАР виступають нормативною базою, однак у

реальному процесі розробки залишається складною задачею практичного застосування цих правил до конкретних ситуацій, виявлених статичним аналізом. Поява великих мовних моделей створює передумови для нового класу інструментів, здатних перетворювати формальні результати перевірок на контекстні пояснення та рекомендації, що скорочують час інтерпретації порушень і підвищують узгодженість рішень у команді. Водночас використання LLM у корпоративному середовищі потребує контрольованого підходу – опори на базу знань, механізмів retrieval (RAG) і обов'язкової валідації результатів – щоб зменшити ризик некоректних або семантично небезпечних порад. Саме ці обставини визначають актуальність теми рекомендаційних систем оцінювання якості ABAP-коду та формують логіку подальшого аналізу предметної галузі в межах роботи.

1.2 Принципи Clean Code та їх адаптація до ABAP

В даній роботі розглядається Clean Code орієнтований на створення коду, який є зрозумілим, читабельним, простим для змін і безпечним у супроводі. Для ABAP ці принципи були спеціально адаптовані у вигляді Clean ABAP – відкритого style guide від SAP, що конкретизує загальні ідеї Clean Code з урахуванням синтаксису мови та характерних патернів enterprise-розробки [4].

Загалом, одним із базових принципів Clean ABAP є зрозуміле іменування. Рекомендації вимагають використовувати назви, які передають зміст і призначення об'єктів, а не технічне кодування чи скорочення. Іншим важливим блоком є структурованість методів: small methods, fail fast, низька вкладеність і прозорий потік керування зменшують когнітивне навантаження на розробника [4].

Окрему увагу Clean ABAP приділяє роботі з internal tables, вибору правильного типу таблиці, уникненню DEFAULT KEY, використанню виразних конструкцій для перевірки існування рядків та явній обробці

помилки через класові винятки. Це показує, що адаптація Clean Code до ABAP враховує не лише стилістику, а й ресурсні та платформні особливості мови [4].

У межах цієї роботи на рисунку 1.1 розглядаємо та досліджуємо принципи Clean ABAP, як нормативна основа побудови Knowledge Base, яка дозволяє пов'язати findings статичного аналізу з конкретними правилами, антипатернами та рекомендаціями щодо рефакторингу.

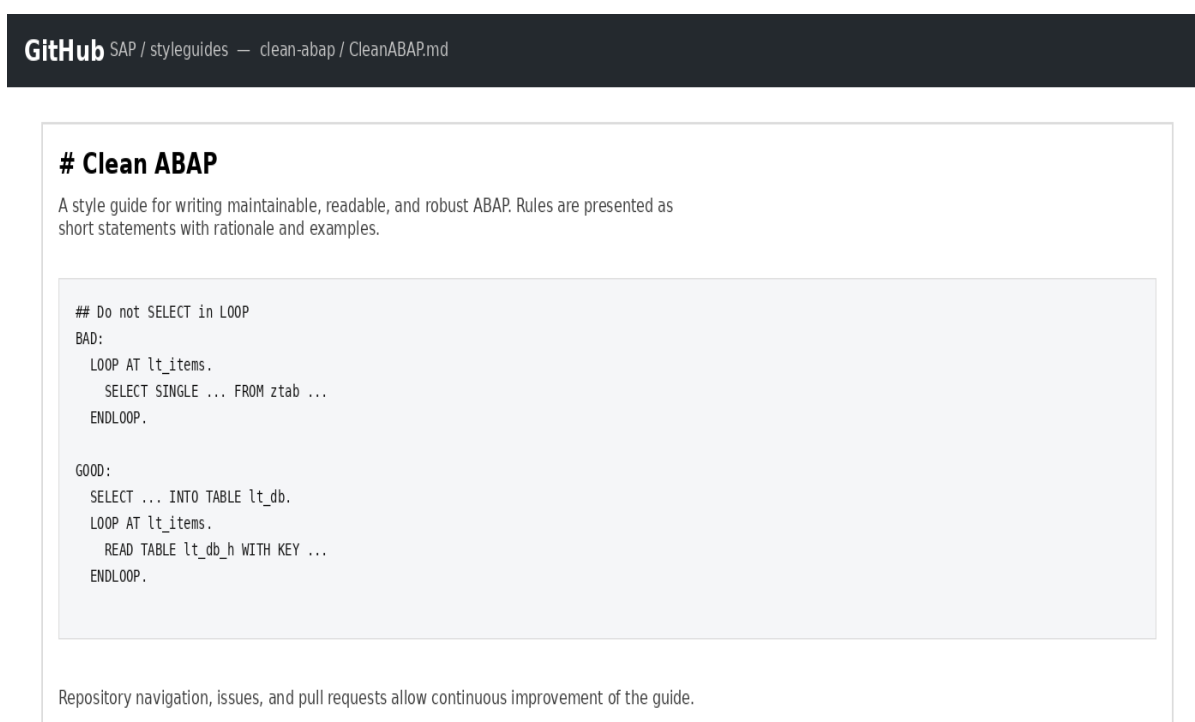


Рисунок 1.1 – Clean ABAP – нормативний style guide

Завдяки цьому LLM отримує не загальні знання «про якісний код», а конкретні правила, релевантні для ABAP.

1.3 Аналіз існуючих підходів до оцінювання якості коду

Шляхи оцінювання якості коду в сучасній інженерній практиці ґрунтуються на поєднанні кількох підходів: ручного code review, статичного аналізу, метрик складності та підтримуваності, rule-based style guides, а

також AI-assisted code analysis. Так бачимо, що Code review допомагає врахувати бізнес-сенс зміни, статичний аналіз швидко знаходить формальні порушення, метрики показують складність і дублювання, а LLM може пояснити finding людською мовою. Окремо кожен підхід має обмеження, але разом вони утворюють більш надійний процес [12], [14].

Також, ручний code review дозволяє враховувати бізнес-контекст, архітектурні обмеження та намір автора коду, але є витратним за часом і погано масштабується на великі кодові бази. На рисунку 1.2 показано, що статичний аналіз, навпаки, добре масштабується і формально виявляє порушення, але зазвичай дає розробнику «сухе» finding-повідомлення [12], [14].

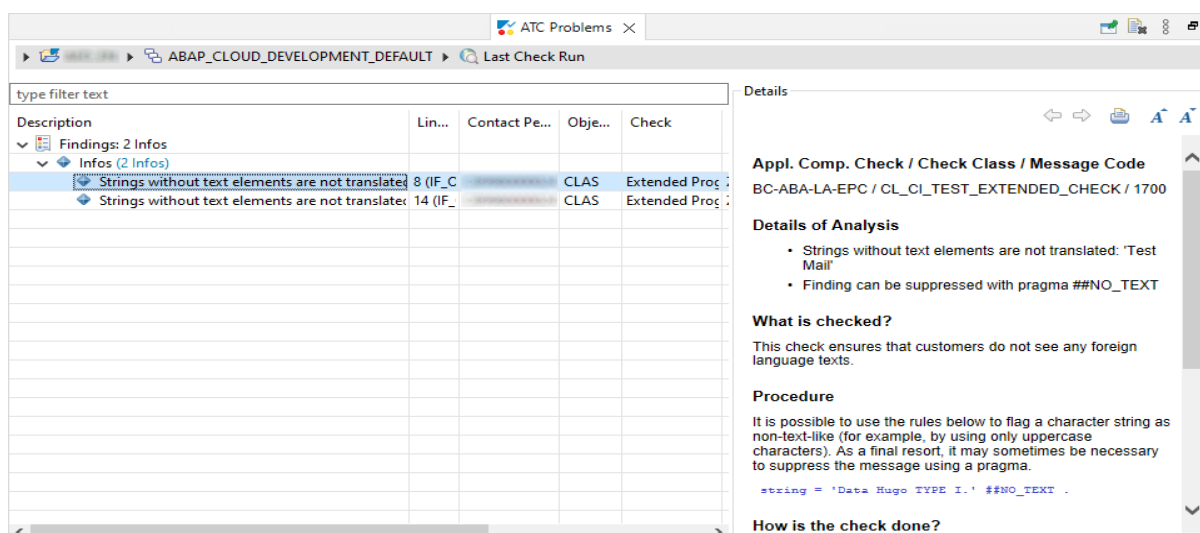


Рисунок 1.2 – ATC Problems в ADT

В роботі було досліджено метрики складності, дублювання, підтримуваності та тестового покриття корисні для ранжування проблемних зон, однак не дають прямих порад щодо безпечного виправлення конкретного порушення. LLM-підхід здатний формувати пояснювані рекомендації, але без rule grounding не забезпечує достатнього рівня довіри [12], [15], [16].

У межах цієї роботи на рисунку 1.1 розглядаємо та досліджуємо

принципи Clean ABAP, як нормативна основа побудови Knowledge Base та пояснювального шару.

У таблиці 1.1 наведено порівняння основних підходів до оцінювання якості коду.

Таблиця 1.1 – Порівняння основних підходів до оцінювання якості коду

Підхід	Перевага	Обмеження	Доцільність для ABAP
Ручний code review	Враховує бізнес-і архітектурний контекст	Високі часові витрати, залежність від експерта	Критичні зміни та фінальне рев'ю
АТС / статичний аналіз	Формальне й раннє виявлення порушень	Не завжди пояснює спосіб виправлення	Базовий quality mechanism
Code metrics	Кількісна оцінка складності й підтримуваності	Не дають конкретного рефакторингу	Пріоритезація проблемних зон
SonarQube для ABAP	Централізований моніторинг, quality gates	Потрібна попередня екстракція коду	Додатковий DevOps-шар
LLM + KB / RAG	Пояснювані поради та контекстність	Потребує grounding і валідації	Перспективна рекомендаційна надбудова

АТС забезпечує статичні перевірки та формує worklist findings. Перевага – формальні правила; обмеження – брак пояснюваних рекомендацій.

1.4 Принципи Clean Code та їх адаптація до ABAP

У сучасній SAP-практиці контроль якості ABAP-коду спирається насамперед на поєднання статичних і динамічних перевірок. SAP прямо визначає ABAP Test Cockpit як стандартний інструмент перевірки якості ABAP development objects за допомогою static checks, а ABAP Unit – як базовий механізм модульного тестування [1], [18]. ABAP Unit підтверджує коректність після змін. Як показано на рисунку 1.3, Coverage допомагає оцінити покриття тестами.

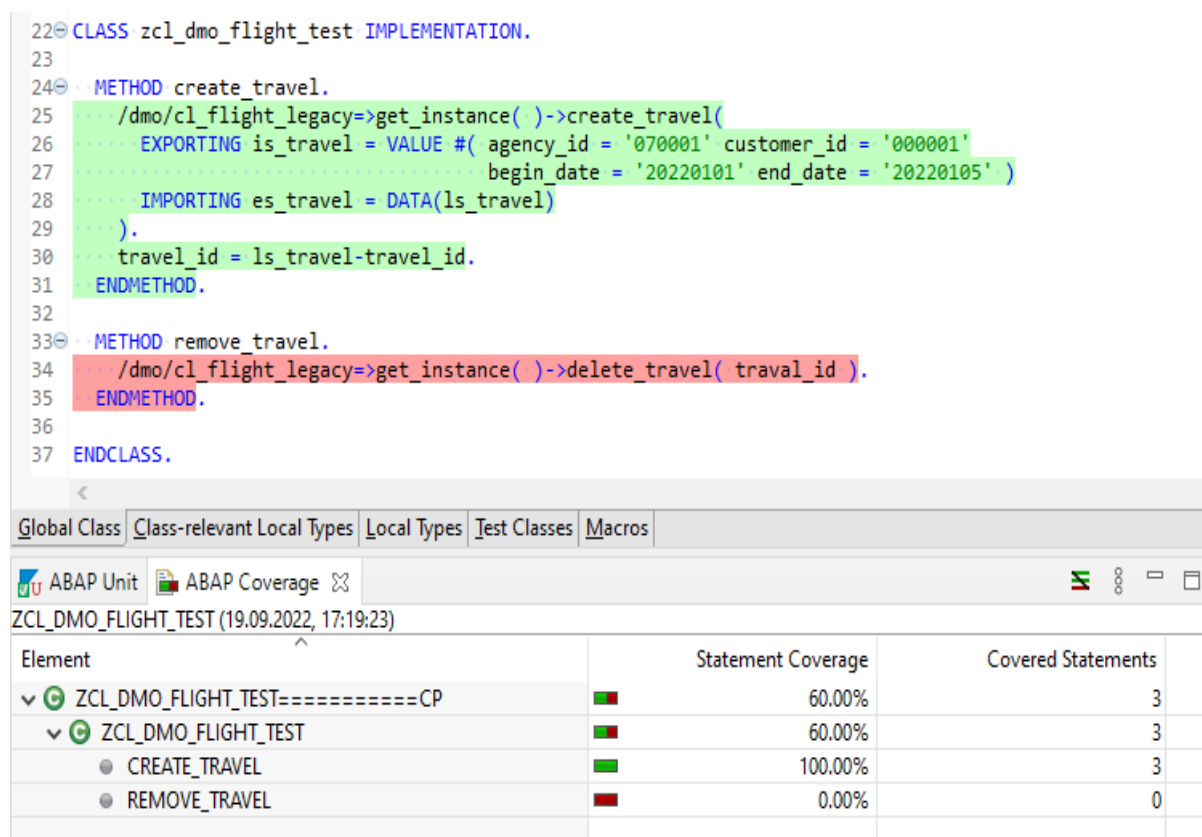


Рисунок 1.3 – ABAP Unit та Coverage

Для частини findings в ADT доступні quick fixes, які скорочують час на усунення типових проблем, але не замінюють експертної інтерпретації. Саме тому показаний в таблиці 1.2 інструментальний ландшафт ABAP quality control є міцною основою для виявлення порушень, проте потребує

рекомендаційної надбудови для пояснення та стандартизації способів виправлення [17].

Таблиця 1.2 – Основні інструменти контролю якості АВАР-коду

Інструмент	Основне призначення	Тип контролю	Роль у процесі розробки
АВАР Test Cockpit (ATC)	Статичний аналіз АВАР-об'єктів	Статичний	Виявлення findings, governance, quality gates
Code Inspector (SCI)	Rule-based check variants	Статичний	Налаштування та сумісність з АТС
АВАР Unit	Модульне тестування	Динамічний	Перевірка коректності після змін
АТС Quick Fixes	Напіваавтоматичне усунення findings	Допоміжний	Зменшення часу на типові виправлення

1.5 Використання великих мовних моделей у задачах аналізу програмного коду

При дослідженні в роботі було виявлено, що великі мовні моделі стали важливим інструментом для генерації, пояснення, трансформації та аналізу програмного коду. Дослідження показують, що їх оцінювання в задачах software engineering має враховувати не лише точність генерації, а й робастність, faithfulness, безпеку та практичну корисність [15], [16].

У SAP-екосистемі цей напрям уже підтримується через Joule for developers та АВАР AI capabilities в ADT: модель може пояснювати вибрані фрагменти коду й допомагати з генерацією АВАР Unit tests [5]. АВАР AI SDK, своєю чергою, дозволяє інтегрувати LLM у власні АВАР-застосунки [6].

Разом з тим в роботі було зроблені деякі висновки, що LLM не повинна розглядатися як самодостатній засіб контролю якості. Дослідження

показують на рисунку 1.4, що моделі краще розуміють синтаксис, ніж глибинну семантику виконання, і можуть галюцинувати при інтерпретації програмних структур [16].

Приклад: виклик LLM через ABAP AI SDK (умовний фрагмент)

```
DATA(lo_client) = zcl_ai_client_factory=>create( ).

DATA(ls_req) = VALUE zif_ai_chat=>ty_request(
  system_prompt = 'You are a code quality assistant for ABAP.'
  user_prompt   = lv_prompt_json ).

DATA(ls_resp) = lo_client->chat_completion( ls_req ).

" Parse structured JSON recommendation
DATA(ls_reco) = zcl_reco_parser=>from_json( ls_resp-content ).

" Show in UI / store for audit
zcl_reco_store=>save( iv_finding_id = lv_finding_id
                    is_recommendation = ls_reco ).
```

Рисунок 1.4 – Виклик LLM через ABAP AI SDK

Тому для задач оцінювання якості коду вони мають працювати поверх формальних finding-повідомлень і на основі контрольованої бази знань. ABAP AI SDK – інтеграційний шар для викликів LLM із ABAP.

1.6 Порівняльний аналіз існуючих рішень та постановка задачі

Використаний в роботі порівняльний аналіз показав в таблиці 1.3, що АТС є найкращим джерелом формалізованих finding-повідомлень і платформно-специфічних перевірок, але не забезпечує достатнього рівня пояснюваності. Clean ABAP є нормативною основою оцінювання якості, але не надає автоматичного зв'язку між правилом і конкретним finding. LLM здатні генерувати корисні поради, але без grounding і незалежної валідації не можуть виступати самостійним механізмом оцінювання [1], [4], [15].

Таблиця 1.3 – Порівняльний аналіз рішень для оцінювання якості АВАР-коду

Рішення/ підхід	Переваги	Обмеження	Висновок для роботи
АТС	Формалізовані перевірки, native SAP integration	Недостатня пояснюваність	Основне джерело фактів про порушення
Clean АВАР	Нормативна база якісного коду	Немає автоматичного мапінгу на finding	Основа knowledge base
Ручний review	Враховує контекст і семантику	Трудомісткий, слабка масштабованість	Механізм експертної валідації
LLM без grounding	Швидкі пояснення та приклади	Галюцинації, ризик змінити семантику	Недостатньо як самостійне рішення
LLM + RAG/KB	Поєднання правил і контекстності	Потребує retrieval та метрик	Найперспективніша основа системи

Отже, актуальною є задача розроблення рекомендаційної системи оцінювання якості АВАР-коду, яка приймає findings статичного аналізу, зіставляє їх із правилами Clean АВАР, використовує retrieval і великі мовні.

Також було зроблено висновки, що моделі для генерації пояснюваних рекомендацій та передбачає обов'язкову валідацію результату через повторний запуск АТС і тести. В роботі ставились задачі, що Мета – спроектувати рекомендаційну систему, що перетворює АТС findings на grounded рекомендації в стилі Clean АВАР із LLM і RAG та забезпечує валідацію.

2 МЕТОДИ ТА ЗАСОБИ ПОБУДОВИ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

Наразі ми розглянемо методи та інструментальні засоби, необхідні для побудови рекомендаційної системи оцінювання якості АВАР-коду на основі принципів Clean Code/Clean АВАР із використанням великих мовних моделей. Основна ідея полягає в тому, щоб поєднати формальні результати статичного аналізу (findings) із нормативними правилами якості коду та перетворити їх на пояснювані й практично застосовні рекомендації для розробника. Для цього необхідно формалізувати критерії якості, визначити структуру бази знань правил і антипатернів, описати модель обробки findings, а також обґрунтувати використання LLM у контрольованому режимі із застосуванням Retrieval Augmented Generation (RAG) та механізмів валідації результатів.

Далі послідовно розглянуто: підхід до формалізації критеріїв якості АВАР-коду, принципи побудови Knowledge Base, логіку retrieval – відбору релевантних правил, правила формування prompt-контракту для LLM, а також систему метрик, за допомогою яких оцінюється корисність, коректність і практична придатність сформованих рекомендацій.

2.1 Формалізація критеріїв якості АВАР-коду

Для продовження дослідження та побудови рекомендаційної системи необхідно перейти від загального уявлення про «якісний код» до набору формалізованих критеріїв в таблиця 2.1, які можна пов'язати з findings, правилами Clean АВАР і шаблонами рекомендацій. У роботі виділяються п'ять груп критеріїв: коректність і робастність, підтримуваність і читабельність, продуктивність, тестованість та платформна сумісність [1], [4].

Таблиця 2.1 – Формалізовані критерії якості АВАР-коду

Група критерію	Зміст критерію	Типові ознаки порушення	Джерело формалізації
Коректність і робастність	Безпечна та передбачувана поведінка	Ігнорування результатів, слабка обробка помилок	ATC findings, Clean ABAP
Підтримуваність	Простота розуміння та змін	Великі методи, погані назви, магічні значення	Clean ABAP, metrics
Продуктивність	Ефективне використання БД і internal tables	SELECT in LOOP, SELECT *, DEFAULT KEY	ATC findings, Clean ABAP
Тестованість	Придатність до АВАР Unit	Сильна зв'язаність, нечіткі контракти	Clean ABAP, ABAP Unit
Платформна сумісність	Відповідність clean core	Нерелізнуті API, platform constraints	ATC, SAP governance

Так, коректність і робастність охоплюють явну обробку помилок, перевірку результатів операцій та прозорі контракти методів. Підтримуваність і читабельність пов'язані з іменуванням, невеликими методами, низькою вкладеністю та осмисленою структурою коду. Продуктивність для АВАР особливо критична через активну взаємодію з БД та internal tables [4], [12].

В роботі видно, що тестованість відображає здатність коду до модульного тестування та безпечних змін, а платформна сумісність охоплює clean core, released API та АВАР Cloud constraints. Така формалізація створює міст між finding-повідомленням і типом майбутньої рекомендації.

2.2 Knowledge Base правил і антипатернів

Для досягнення поставлених задач, зрозуміло, що великі мовні моделі формували не лише правдоподібні, а й керовані та відтворювані рекомендації, їм необхідна опора на контрольоване джерело доменних знань. У роботі таким джерелом виступає Knowledge Base – структурована база знань, яку зображено в таблиці 2.2 та, яка містить правила Clean ABAP, типові антипатерни, приклади «поганого» і «хорошого» коду, а також зв'язки між finding-повідомленнями й відповідними рекомендаціями [4], [11].

Таблиця 2.2 – Рекомендована структура rule card у Knowledge Base

Поле	Призначення
card_id	Унікальний ідентифікатор картки
title/source/anchor	Назва правила та джерело
tags/atc_categories/hints	Пошукові та класифікаційні ознаки
problem	Опис антипатерну або проблеми
recommendation	Сутність рекомендованої зміни
examples_bad/examples_good	Мінімальні приклади before/after
validation/constraints	Чекліст перевірки й platform limits
embedding_text	Текст для векторизації та semantic search

Таким чином, базовою одиницею KB є rule card, яка містить назву правила, джерело, теги, hints до finding-класів, опис проблеми, сутність рекомендації, приклади before/after, checklist валідації, platform constraints і embedding_text для semantic retrieval. Такий формат одночасно придатний і для людини, і для retrieval-компонента [4], [10], [11].

При дослідженні було зрозуміло, що KB у цій роботі розглядається не як енциклопедія про АВАР, а як спеціалізована доменна база знань для задачі code quality recommendation. Її спеціалізація зменшує ризик нерелевантних відповідей і підвищує groundedness генерації.

2.3 Модель обробки findings: від результату ATC до рекомендації

В роботі було розглянуто модель обробки findings, яку зображено в таблиці 2.3 та будується як багатоетапний pipeline: отримання та нормалізація results of ATC check run, витяг програмного контексту, класифікація finding за групою якості, retrieval релевантних rule cards, формування структурованого prompt-запиту, генерація рекомендації LLM, post-processing та повторна валідація [1], [11].

Виходячи з цього зрозуміло, що Finding сам по собі не містить достатньої інформації для якісної поради. Лише поєднання finding, code context і Knowledge Base дозволяє визначити, чи доцільніше рекомендувати bulk select, зміну типу internal table, fail fast refactoring, введення exceptions або platform-compliant alternative.

Таблиця 2.3 – Етапи моделі обробки findings

№	Етап	Основна дія	Результат
1	Отримання findings	Витяг results of ATC	Нормалізований finding
2	Витяг контексту	Побудова code context	Snippet і технічні сигнали
3	Класифікація	Визначення групи якості	Тип рекомендації
4	Retrieval	Пошук rule cards	Top-k knowledge cards
5	Prompt building	Формування структурованого запиту	Prompt object

Продовження таблиці 2.3

№	Етап	Основна дія	Результат
6	Generation	Створення рекомендації	Structured recommendation
7	Post-processing	Перевірка повноти й обмежень	Підготовлена відповідь
8	Validation	Rerun ATC / tests	Підтвердження або відхилення

Робимо висновок, що саме модель обробки findings перетворює технічне повідомлення статичного аналізу на інженерно корисну рекомендацію з поясненням, рівнем ризику та чеклістом перевірки.

2.4 Використання LLM для генерації пояснюваних рекомендацій

В дослідженні у запропонованій системі LLM використовується не як автономний інструмент «оцінювання коду», а як керований генератор пояснюваних рекомендацій та відображено в таблиці 2.4.

Таблиця 2.4 – Рекомендована структура взаємодії з LLM

Блок	Зміст	Призначення
finding	ID перевірки, message, category	Формальна фіксація проблеми
code_context	Фрагмент коду та сигнали	Контекстуалізація finding
retrieved_rules	Rule cards із KB	Grounding і зниження галюцинацій
constraints	Обмеження та правила безпеки	Контроль семантичних ризиків
response_schema	summary, change, validation	Стандартизація виходу
validation	ATC rerun, ABAP Unit	Підтвердження коректності зміни

На вхід recommendation engine надходять finding, code context, retrieved rules та constraints; на вихід – структурована відповідь із коротким висновком, поясненням значущості порушення, посиланням на релевантні правила, suggested change і validation checklist [5], [6], [7], [8], [9].

Так показано, що ключовим методичним елементом є prompt contract. Він фіксує роль моделі, формат відповіді та обмеження: не змінювати бізнес-логіку без потреби, враховувати clean core, не пропонувати нерелізнуті API, явно формулювати припущення у разі недостатнього контексту. Такий підхід безпосередньо впливає з рекомендацій SAP щодо використання generative AI в АВАР-середовищі [6], [7].

2.5 RAG - підхід та механізм зниження галюцинацій

В роботі однією з ключових проблем застосування LLM є ризик галюцинацій в таблиці 2.5 показано механізми зниження галюцинацій – правдоподібних, але непідтверджених або нерелевантних відповідей. Для задач оцінювання якості АВАР-коду цей ризик критичний, оскільки помилкова рекомендація може або не усунути finding, або змінити семантику програми [15], [16].

Таблиця 2.5 – Основні механізми зниження галюцинацій

Механізм	Сутність	Роль у системі
Domain KB	Контрольоване джерело правил	Обмежує простір відповідей
Retrieval before generation	Пошук релевантних rule cards	Зменшує довільність генерації
Metadata filtering	Фільтрація за категорією та constraints	Підвищує точність retrieval
Semantic search	Пошук за змістом на основі embeddings	Знаходить близькі за сенсом правила

Продовження таблиці 2.5

Механізм	Сутність	Роль у системі
Structured output	Обов'язкова схема відповіді	Робить відповідь перевірюваною
Post-processing	Автоматична перевірка відповіді	Відсікає некоректні результати
Rerun ATC/tests	Незалежна перевірка після змін	Підтверджує реальну корисність

Тому в роботі щоб зменшити цей ризик, використовується Retrieval Augmented Generation. Сутність RAG полягає в тому, що перед генерацією виконується пошук релевантних зовнішніх знань, а модель формує відповідь уже на основі цієї підмножини правил. У SAP reference architecture окремо виділяються embeddings і semantic search, а SAP HANA Cloud Vector Engine підтримує векторні подання та similarity search як основу для такого пошуку [10], [11].

Механізм зниження галюцинацій реалізується на кількох рівнях: обмеження джерела знань, retrieval before generation, structured output, post-processing і mandatory rerun ATC/tests. Така багаторівнева схема є значно надійнішою, ніж спроба покладатися лише на prompt engineering.

2.6 Метрики оцінювання якості рекомендацій

Дослідження оцінювання recommendation pipeline не може обмежуватися лише суб'єктивним враженням від тексту відповіді. У сучасних оглядах RAG-систем підкреслюється необхідність окремо аналізувати retrieval-компонент і generation-компонент, використовуючи relevance, accuracy, faithfulness, retrieval precision та retrieval recall, які зображено в таблиці 2.6 [15].

Таблиця 2.6 – Основні метрики оцінювання рекомендаційної системи

Група	Метрика	Що оцінює
Retrieval	Precision@k	Частку релевантних rule cards у top-k
Retrieval	Recall@k	Повноту відбору релевантних cards
Retrieval	NDCG@k	Якість ранжування retrieved cards
Generation	Groundedness	Спирання рекомендації на KB
Generation	Unsupported claim rate	Частку непідтверджених тверджень
Generation	Schema validity rate	Відповідність відповіді формату
Utility	Acceptance rate	Частку рекомендацій, прийнятих розробником
Utility	Time-to-fix	Час до внесення коректної зміни
Validation	ATC fix rate	Частку findings, усунутих після rerun
Validation	ABAP Unit pass rate	Збереження коректної поведінки
Secondary	Complexity / maintainability delta	Вплив на структурну якість коду

У роботі метрики групуються за чотирма напрямками: retrieval-якість, якість самої рекомендації, практична корисність для розробника та валідація після внесення змін. Для retrieval використовуються Precision@k, Recall@k, NDCG@k; для generation-groundedness, unsupported claim rate і schema validity rate; для practical utility-acceptance rate і time-to-fix; для validation - ATC fix rate, delta findings і ABAP Unit pass rate [1], [15].

Такий підхід дозволяє оцінювати не лише якість тексту відповіді, а й фактичний внесок рекомендаційної системи у покращення якості ABAP-коду.

2.7 Проектування системи

Система проектується як pipeline: ATC → preprocessing → KB/retrieval → LLM → UI → validation.

На практиці це означає, що кожен finding проходить однаковий маршрут, як показано на рисунку 2.1, спочатку факт порушення, потім контекст і правила, після цього рекомендація та перевірка. Завдяки стандартизованій послідовності етапів забезпечується трасованість рішення: можна відстежити, які саме правила з KB були використані та чому запропоновано конкретний варіант виправлення.

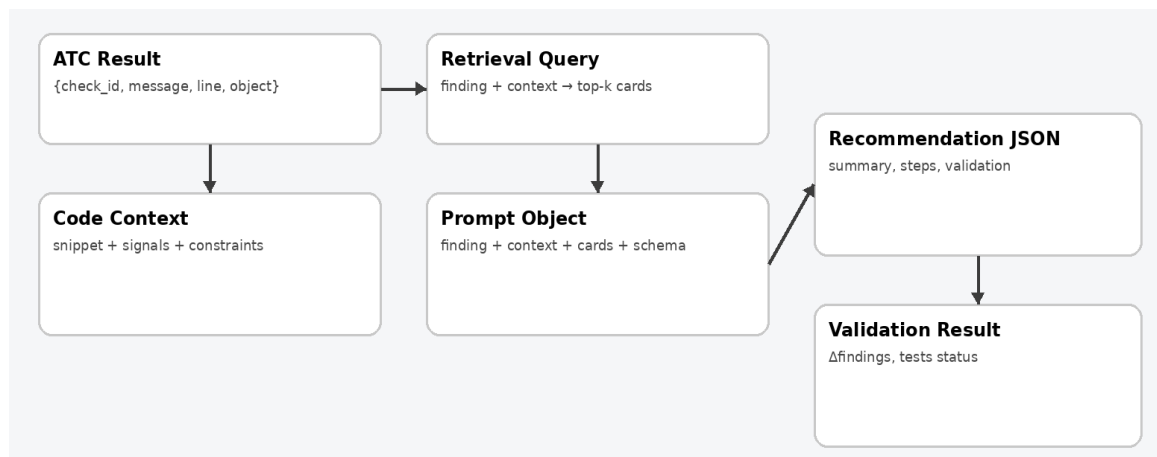


Рисунок 2.1 – Потоки даних у pipeline

Крім того, наявність обов'язкового кроку validation (повторний запуск АТС і тести) дозволяє підтверджувати коректність рекомендацій і порівнювати ефект покращення між різними запусками та версіями коду.

3 РОЗРОБКА РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ОЦІНЮ- ВАННЯ ЯКОСТІ АВАР-КОДУ

На даному етапі дослідження виконується первинна розробка основної структури рекомендаційної системи оцінювання якості АВАР-коду, яка враховує актуальні потреби корпоративних SAP-ландшафтів та спрямована на підвищення ефективності процесів контролю якості, супроводу й модернізації програмних об'єктів.

У межах роботи сформовано основне функціональне середовище, що забезпечує отримання та обробку результатів статичного аналізу (findings) АВАР Test Cockpit, витяг контексту проблемних фрагментів коду, а також ведення бази знань правил і антипатернів відповідно до принципів Clean Code/Clean АВАР. Реалізовано логіку, яка дозволяє систематизувати порушення за категоріями (продуктивність, підтримуваність, робастність, платформна сумісність), відстежувати їх пріоритетність і формувати узагальнену аналітику щодо стану якості коду.

Особливу увагу приділено розробці інтелектуального рекомендаційного модуля на основі великих мовних моделей, який інтерпретує findings у контексті конкретного АВАР-коду та генерує пояснювані рекомендації щодо виправлення й рефакторингу. Запропонований підхід поєднує Retrieval Augmented Generation (RAG) із контрольованою базою знань (Knowledge Base), що дозволяє прив'язувати рекомендації до нормативних правил Clean АВАР, зменшувати ризик некоректних порад і забезпечувати відтворюваність результатів. Система не лише автоматизує рутинні етапи аналізу повідомлень статичної перевірки, а й підтримує стандартизований підхід до усунення типових порушень, формуючи чекліст перевірки змін (повторний запуск АТС, виконання АВАР Unit-тестів та, за потреби, перевірки продуктивності).

Таким чином, розроблений підхід базується на поєднанні традиційних засобів контролю якості АВАР-коду (АТС, АВАР Unit, практики Clean

АВАР) із сучасними інструментами пошуку знань і генеративного штучного інтелекту, що дозволяє створити інтегровану систему підтримки прийняття рішень у процесі АВАР-розробки та зменшити витрати часу на інтерпретацію порушень і підготовку коректних виправлень.

3.1 Загальна архітектура системи

В роботі розглянуто архітектура рекомендаційної системи на рисунку 3.1, будується як поєднання формального контролю якості, доменної бази знань і керованої генерації рекомендацій. У загальному вигляді система включає шість логічних компонентів: джерело findings, preprocessing, Knowledge Base, retrieval-шар, LLM recommendation engine та шар подання результату й валідації [1], [8], [10], [11].

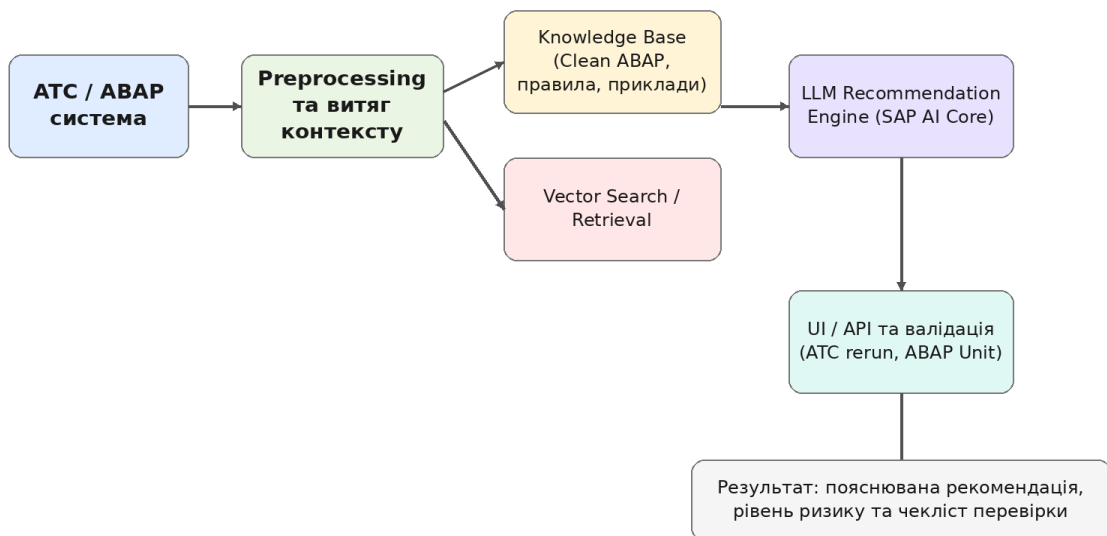


Рисунок 3.1 – Узагальнена архітектура рекомендаційної системи

З дослідження бачимо в таблиця 3.1, що АТС виступає джерелом фактів про порушення, preprocessing перетворює finding на структурований контекст, KB і retrieval забезпечують grounded generation, а LLM recommendation engine формує пояснювану рекомендацію.

Таблиця 3.1 – Основні компоненти архітектури системи

Компонент	Основна функція	Вхідні дані	Результат
АТС/джерело findings	Виявлення порушень якості	АВАР development objects	Findings
Preprocessing	Нормалізація finding і витяг контексту	Findings, код об'єкта	Structured context
Knowledge Base	Зберігання правил і прикладів	Rule cards, metadata	Retrieval-ready knowledge
Retrieval layer	Пошук релевантних rule cards	Context, hints, embeddings	Top-k карток
LLM engine	Генерація пояснюваної рекомендації	Prompt contract	Structured recommendation
UI/validation	Показ рекомендації та quality checks	Recommendation, changed code	User-facing result

Подальша валідація через повторний запуск АТС і АВАР Unit замикає цикл «finding → recommendation → validation».

3.2 Модель даних та структура вхідних і вихідних об'єктів

Під час досліджень було виявлено, що для практичної реалізації системи формалізовано сім базових сутностей: Finding, Code Context, Rule Card, Retrieval Result, LLM Prompt Object, Recommendation і Validation Result – це відображено в таблиці 3.2. Така модель даних відокремлює формальний факт порушення від кодового контексту, нормативного знання, генеративної відповіді та незалежної валідації.

Таблиця 3.2 – Основні вхідні та вихідні об'єкти системи

Сутність	Тип	Призначення
Finding	Вхідний	Подання конкретного АТС-порушення
Code Context	Проміжний	Контекстуалізація finding у коді
Rule Card	База знань	Нормативне правило або антипатерн
Retrieval Result	Проміжний	Набір релевантних knowledge cards
LLM Prompt Object	Проміжний	Контракт взаємодії з LLM
Recommendation	Вихідний	Пояснювана порада щодо виправлення
Validation Result	Вихідний	Підтвердження ефективності рекомендації

Також у структурі Finding зображеного в таблиці 3.3 зберігаються ідентифікатор перевірки, категорія, пріоритет, message, об'єкт розробки та позиція в коді. Code Context містить snippet та технічні сигнали. Rule Card формалізує правило Quality Knowledge Base.

Таблиця 3.3 – Приклад логічної структури Finding і Recommendation

Поле	Приклад значення
check_id	CI_SEL_NESTED
category	Performance
message	SELECT SINGLE on table in LOOP
suggested_change	Collect keys, bulk SELECT, read from hashed table

Recommendation є головним вихідним об'єктом із полями summary, suggested_change, validation_steps, assumptions і risk_level. Validation Result фіксує результати rerun ATC та ABAP Unit після внесення змін – це показано в таблиці 3.3.

3.3 Алгоритм роботи рекомендаційної системи

При аналізі рисунка 3.2 та детальному дослідженні прийшли до висновку, що алгоритм роботи системи включає десять кроків (таблиця 3.4): запуск ATC, нормалізацію findings, витяг контексту, класифікацію проблеми, retrieval rule cards, формування prompt, генерацію recommendation, post-processing, подання результату користувачу та повторну валідацію після внесення змін [1], [6], [8], [11].

Таблиця 3.4 – Алгоритм роботи рекомендаційної системи

№	Крок	Вхід	Вихід
1	Запуск ATC	ABAP-об'єкт або пакет	Results of check run
2	Нормалізація findings	ATC results	Structured findings
3	Витяг контексту	Finding + код	Code context
4	Класифікація проблеми	Finding + context	Якісна категорія
5	Retrieval	Finding + context + KB	Top-k rule cards
6	Формування prompt	Context + rules	Prompt object
7	Генерація	Prompt object	Structured recommendation
8	Post-processing	Recommendation	Checked recommendation
9	Подання користувачу	Checked recommendation	UI/API result
10	Валідація	Changed code	Rerun ATC/ABAP Unit

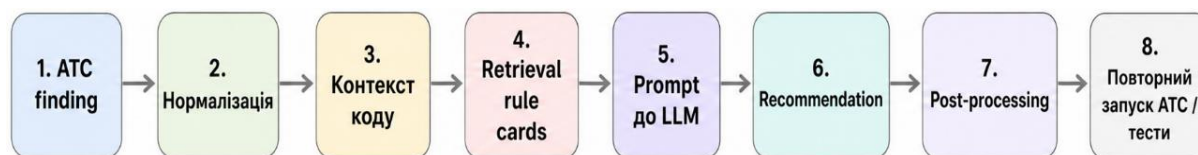


Рисунок 3.2 – Pipeline обробки finding: від ATC до рекомендації

Такий алгоритм в таблиці 3.4 поєднує формальну надійність static code analysis із пояснювальними можливостями LLM та незалежною перевіркою результату, що робить систему придатною для включення в практичний процес АВАР-розробки. Сильна сторона алгоритму полягає в тому, що він не змішує пояснення з перевіркою. LLM допомагає сформулювати пораду, але остаточне підтвердження дають інструменти SAP: повторний ATC, АВАР Unit і за потреби профілювання.

Це важливо для корпоративних систем, де помилка рефакторингу може вплинути на реальний бізнес-процес.

3.4 Приклади типових сценаріїв роботи системи

В роботі були проаналізовані типові сценарії, які зображено в таблиці 3.5 вони охоплюють проблеми продуктивності, підтримуваності, робастності та clean core. Найпоказовіший приклад – SELECT in LOOP: код працює правильно на десяти записах, але на десяти тисячах створює тисячі звернень до БД. Інший приклад – deep nesting, коли бізнес-правило заховане в кількох рівнях IF і LOOP, через що його важко перевірити тестами.

Таблиця 3.5 – Типові сценарії роботи рекомендаційної системи

Сценарій	Тип finding	Основний фокус рекомендації	Очікувана валідація
SELECT in LOOP	Performance	Bulk select, set-based access, proper table type	Rerun ATC, performance check

Продовження таблиці 3.5

Сценарій	Тип finding	Основний фокус рекомендації	Очікувана валідація
DEFAULT KEY/table access	Performance/ Maintainability	Explicit key, HASHED/SORT ED TABLE, line exists	Rerun ATC, unit tests
Deep nesting / long method	Maintainability	Guard clauses, extract methods, simplify flow	Rerun ATC, ABAP Unit
Weak error handling	Robustness	Explicit checks, exceptions, clearer contract	Rerun ATC, regression tests
Unreleased API/ clean core	Platform compatibility	Released API, wrapper, compliant extension	Rerun ATC, architecture review

Також у кожному сценарії retrieval-шар підбирає релевантні rule cards, а recommendation engine формує пояснення, suggested change і validation checklist.

Для performance-кейсів акцент робиться на set-based доступі до даних та proper table type; для maintainability – на small methods, guard clauses і simplification; для robustness – на explicit checks і exceptions; для platform findings – на released API і compliant extension strategy [3], [4], [6].

3.5 Оцінювання результатів роботи системи

Отже, оцінювання результатів системи показанов таблиці 3.6 та має здійснюватися багаторівнево: окремо для retrieval, generation, practical utility і formal validation after changes. Такий підхід дозволяє виявити, чи проблема виникла через нерелевантні rule cards, помилкову інтерпретацію retrieved rules, низьку корисність для розробника або відсутність фактичного покращення коду [15].

Таблиця 3.6 – Показники оцінювання результатів роботи системи

Група оцінювання	Метрика	Інтерпретація
Retrieval	Precision@k	Частка релевантних rule cards у top-k
Retrieval	Recall@k	Повнота відбору релевантних rule cards
Retrieval	NDCG@k	Якість ранжування результатів retrieval
Generation	Groundedness	Наскільки рекомендація спирається на KB
Generation	Unsupported claim rate	Частка непідтверджених тверджень
Generation	Schema validity rate	Відповідність виходу заданій схемі
Utility	Acceptance rate	Частка рекомендацій, прийнятих розробником
Utility	Time-to-fix	Середній час до внесення виправлення
Validation	ATC fix rate	Частка findings, усунутих після rerun ATC
Validation	ABAP Unit pass rate	Збереження коректної поведінки
Secondary	Maintainability / complexity delta	Вторинний ефект на структуру коду

Так ключовими показниками виступають Precision@k, Recall@k, groundedness, acceptance rate, time-to-fix, ATC fix rate і ABAP Unit pass rate.

Саме вони дозволяють довести, що рекомендаційна система не лише генерує переконливий текст, а й реально сприяє покращенню якості ABAP-коду [1], [15], [18].

3.6 Можливості інтеграції та подальшого розвитку

В дослідженні виявлено, що практично система може бути інтегрована в ABAP Development Tools як розширення до workflow роботи з ATC findings, у централізований quality governance процес, у code review і CI/CD сценарії. ABAP AI SDK забезпечує інтеграційний шар для роботи з LLM із ABAP-коду, SAP AI Core – керований runtime для recommendation engine, а SAP HANA Cloud Vector Engine – основу для Knowledge Base і semantic retrieval [6], [8], [9], [10].

Подальший розвиток системи доцільно пов'язувати з еволюцією Knowledge Base, накопиченням прикладів успішних виправлень, персоналізацією retrieval під політики конкретної команди, а також із генерацією допоміжних артефактів – коротких summaries, шаблонів ABAP Unit tests і безпечних quick-fix-like suggestions для простих finding-класів [5], [6].

3.7 Опис результату та прототип інтерфейсу

У результаті виконання роботи було розроблено програмний прототип (Proof of Concept), що демонструє ключові функціональні можливості запропонованої архітектури. Прототип реалізовано на мові програмування Python та включає модулі емуляції ATC, локальної Knowledge Base, механізму Retrieval (пошук за ключовими словами) та симуляції відповіді LLM (використання шаблонів на основі знайдених правил).

Основним результатом є демонстрація життєвого циклу обробки помилки:

- вхід: система отримує «Finding» (наприклад, SELECT in LOOP);
- аналіз: система аналізує ключові слова та пошукає відповідне правило в базі знань (Knowledge Base);

- генерація: на основі правила формується структурована рекомендація з описом проблеми, пропозицією щодо рефакторингу (наприклад, «Використати SELECT ... FOR ALL ENTRIES») та чек-лістом перевірки;

- вихід: користувачеві (розробнику) надається зрозумілий план дій.

Прототип інтерфейсу користувача в даній реалізації представлений у вигляді консольного виводу, що відображає всі етапи обробки інформації.

У повноцінній комерційній реалізації цей інтеграційний рівень може бути вбудований безпосередньо в середовище ABAP Development Tools (ADT) як плагін або як веб-додаток, що взаємодіє з SAP BTP.

Практична цінність отриманих результатів полягає у доведенні можливості автоматизації інтерпретації технічних повідомлень АТС, що знижує поріг входу для нових розробників та скорочує час на виправлення помилок для досвідчених фахівців.

4 ОПИС РЕЗУЛЬТАТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ

У цьому розділі узагальнемо кінцеві результати кваліфікаційної роботи та подамо опис розроблених артефактів: архітектури рекомендаційної системи, моделі даних, структури Knowledge Base, а також прототипних матеріалів інтерфейсу користувача. Опис зорієнтований на практичну перевірюваність результатів та можливість подальшої реалізації рішення у SAP-ландшафті.

4.1 Склад і характеристика отриманих результатів

Результатом роботи є узгоджений комплект проектних і прототипних матеріалів наданов таблиці 4.1, що описують рекомендаційну систему оцінювання якості ABAP-коду на основі принципів Clean Code/Clean ABAP із використанням великих мовних моделей та RAG-підходу.

Таблиця 4.1 – Склад результатів кваліфікаційної роботи

№	Назва результату (Компонент)	Коротка характеристика та зміст
1	Архітектурна модель та потоки даних	Розроблена схема архітектури, що описує рух даних від джерела (АТС) через етапи обробки (Preprocessing, Retrieval, LLM) до кінцевого користувача (UI) та валідації.
2	Логічна модель даних	Формалізована структура сутностей системи: Finding (порушення), Rule Card (картка правила), Prompt Object (запит до LLM) та Recommendation (рекомендація), що визначає інтерфейси взаємодії модулів.
3	База знань (Knowledge Base)	Опис структури та наповнення бази правил Clean ABAP. Включає механізм retrieval (пошуку) та ранжування правил для забезпечення релевантності підказок.

Продовження таблиці 4.1

№	Назва результату (Компонент)	Коротка характеристика та зміст
4	Прототипні UI-матеріали	Набір макетів (mockups) інтерфейсу, що демонструють сценарії роботи розробника з перегляду findings та адміністратора з наповнення бази знань.
5	Наскрізний сценарій та чек-ліст	Детальний опис робочого процесу (workflow) на прикладі помилки «SELECT in LOOP», включаючи формування рекомендації та чек-ліст обов'язкової валідації змін.

Комплект результатів включає: (1) архітектурну модель системи та потоки даних; (2) логічну модель даних для findings, rule cards, prompt-об'єктів і рекомендацій; (3) опис бази знань та підходу до retrieval/ранжування; (4) прототипні UI-матеріали, що демонструють роботу з findings і керування Knowledge Base; (5) приклад наскрізного сценарію формування рекомендації та чекліст валідації.

4.2 Архітектура рекомендаційної системи

В роботі показано на рисунку 4.1, що архітектура системи реалізує RAG-контур, у якому статичний аналіз (АТС) виступає джерелом фактів про порушення, а генерація рекомендацій відбувається лише після підбору релевантних правил із Knowledge Base. Preprocessing нормалізує finding і витягує контекст коду; Knowledge Base зберігає rule cards та embeddings; retrieval-шар комбінує metadata-фільтрацію і vector search; LLM engine (SAP AI Core/GenAI Hub) формує структуровану рекомендацію; Recommendation API/UI відображає результат і джерела grounding.

Цикл завершується обов'язковою валідацією через повторний запуск АТС та ABAP Unit.

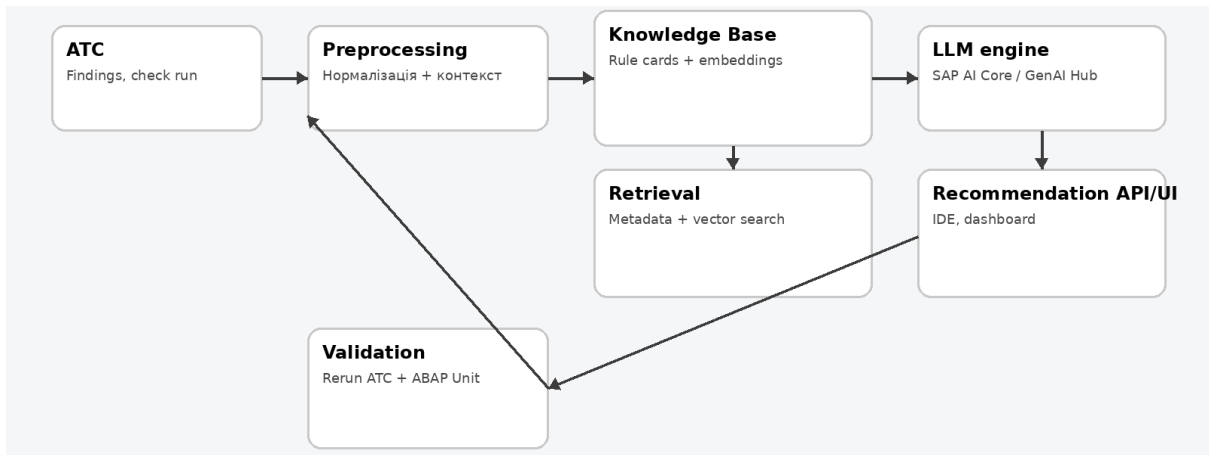


Рисунок 4.1 – Архітектура рекомендаційної системи оцінювання якості АВАР-коду (RAG + LLM)

4.3 Прототип інтерфейсу користувача: findings та рекомендації

Також показано в роботі на рисунку 4.2, що прототип екрана «Findings & Recommendation» демонструє робочий сценарій після АТС-перевірки. Ліва панель містить список findings поточного check run із фільтрацією за категорією та швидким переходом до проблемного місця. Після вибору finding у правій частині відображається рекомендація у стандартизованій формі: summary, why it matters, suggested change (steps) і validation.

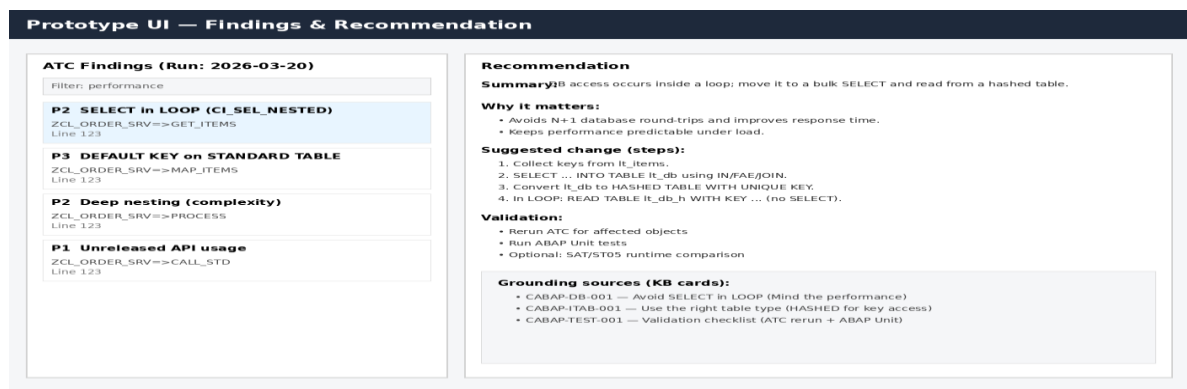


Рисунок 4.2 – UI (прототип): findings та сформована рекомендація

Блок «Grounding sources» фіксує КВ-картки, використані для формування поради, що підвищує прозорість та полегшує code review.

4.4 Прототип інтерфейсу користувача: керування Knowledge Base

В роботі під час розробки отримано на рисунку 4.3 прототип екрана «Knowledge Base Management» демонструє підтримку rule cards як керованого артефакта. Ліва панель відображає список карток і пошук за ключовими словами/тегами.

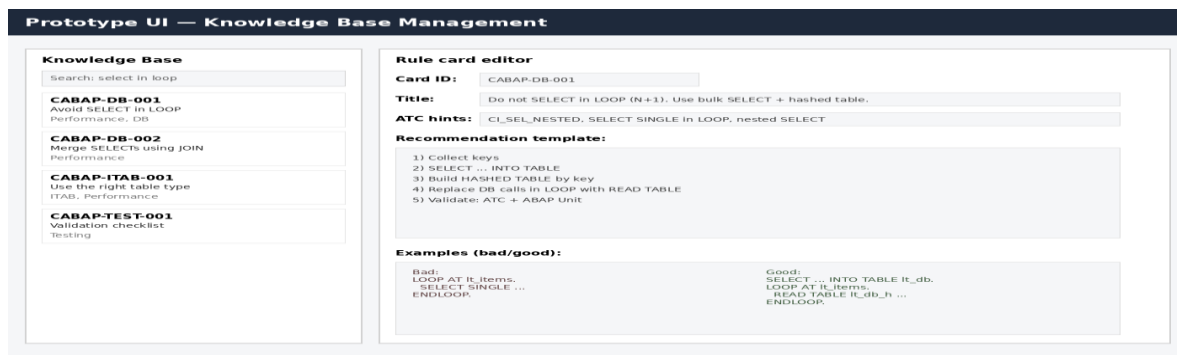


Рисунок 4.3 – UI (прототип): керування Knowledge Base (rule cards)

Права панель містить редактор картки, де фіксуються поля, критичні для retrieval і grounded generation: Card ID, title, ATC hints, шаблон рекомендації та приклади bad/good. Такий інтерфейс дозволяє доповнювати КВ team rules, підтримувати зв'язок із finding-патернами та забезпечувати відтворюваність рекомендацій.

4.5 Приклад наскрізного сценарію та формат результату

Для відтворюваного використання рекомендаційного модуля визначено структурований контракт взаємодії з LLM: вхід включає finding, code context, retrieved rules і constraints, а вихід — структуровану

рекомендацію у форматі JSON. Такий формат придатний до відображення в UI та до вимірювання метрик groundedness/acceptance/ATC-fix. Приклад скороченого виходу наведено в рисунку 4.4.

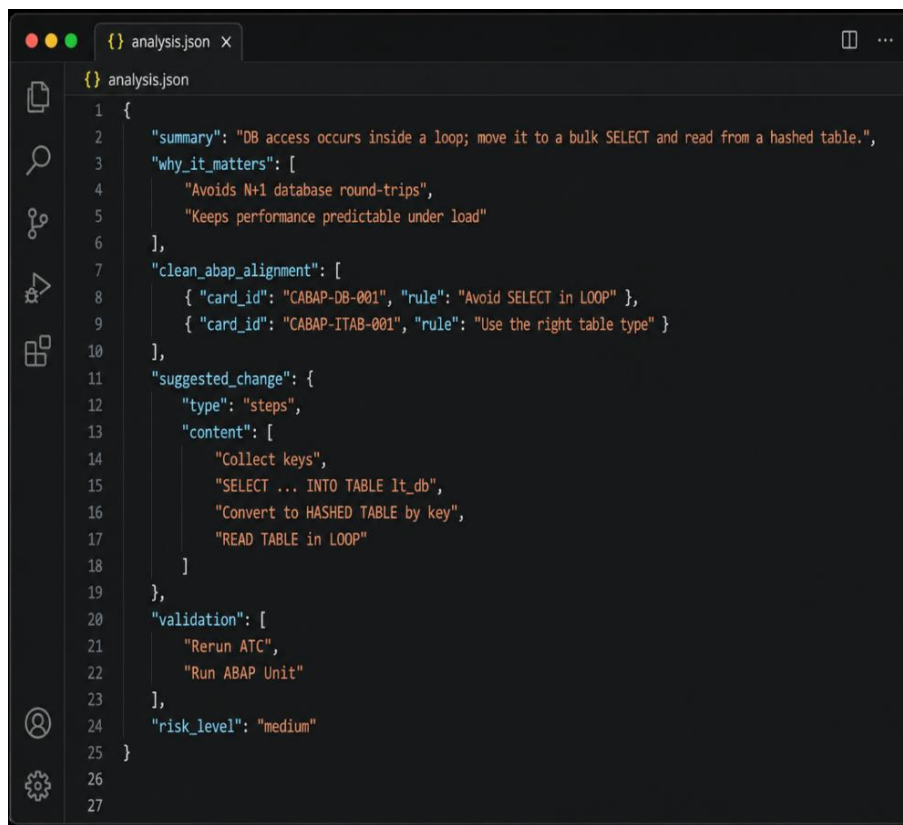


Рисунок 4.4 – Приклад структурованого виходу рекомендації (JSON)

Також в таблиці 4.2 наведено загальні дані.

Таблиця 4.2 – Загальні дані

Артефакт	Призначення	Форма подання
Архітектура системи (RAG + LLM)	Компоненти та взаємодія між ATC, preprocessing, KB, retrieval, LLM і контуром валідації.	Схема (рис. 4.1)
UI-прототип: findings → recommendation	Відображення findings і структурованої рекомендації з джерелами grounding.	Макет (рис. 4.2)

Продовження таблиці 4.2

Артефакт	Призначення	Форма подання
UI-прототип: Knowledge Base	Пошук і редагування rule cards та шаблонів рекомендацій, приклади bad/good.	Макет (рис. 4.3)
Контракт рекомендації	Структура виходу LLM і зв'язок із finding та КВ-картками.	JSON (листинг 4.1)
Приклад rule card	Еталонна картка правила для RAG-опори (problem → recommendation → examples → validation).	Додаток А

Повні приклади rule card і шаблон prompt/response наведено в додатках.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розв’язано актуальну науково-прикладну задачу побудови рекомендаційної системи оцінювання якості АВАР-коду, яка поєднує результати статичного аналізу, принципи Clean АВАР та можливості великих мовних моделей. Поставленої мети досягнуто, усі визначені завдання виконано.

В роботі було проаналізовано предметну область забезпечення якості АВАР-коду в сучасних SAP-системах; розглянуто вплив підходу Clean Core на вимоги до кастомної розробки, а також роль ATC, Code Inspector, АВАР Unit і великих мовних моделей у задачах code quality. У другому розділі формалізовано критерії якості АВАР-коду, запропоновано структуру Knowledge Base правил і антипатернів у форматі rule cards, описано модель обробки findings, RAG-механізм зниження галюцинацій та систему метрик оцінювання retrieval, generation і practical utility. У третьому розділі розроблено архітектуру системи, модель даних, алгоритм роботи, типові сценарії використання та напрями інтеграції рішення в ADT, quality governance контур, SAP AI Core і SAP HANA Cloud Vector Engine.

Основними результатами роботи є: концептуальна архітектура рекомендаційної системи, формалізована модель даних (finding → context → retrieval → recommendation → validation), структура Knowledge Base, приклади шаблонів prompt/вихідного формату рекомендацій, а також прототипні макети інтерфейсу для перегляду findings і отримання рекомендацій. Елементи новизни полягають у формалізованому мапінгу «ATC findings → правила Clean АВАР → пояснювана рекомендація LLM» із grounding через Knowledge Base та обов’язковим контуром валідації (повторний запуск ATC і АВАР Unit), а також у запропонованій системі метрик, яка дозволяє вимірювати не лише якість тексту рекомендацій, але й їх практичний ефект.

Практичне значення одержаних результатів полягає в можливості

використання запропонованого підходу як основи для AI-підсиленого інструмента підтримки АВАР-розробника, який прискорює інтерпретацію findings, стандартизує рекомендації та підвищує керованість покращення коду. Разом із тим застосування системи потребує наповнення Knowledge Base правилами конкретної організації та валідації на реальних кодових базах, оскільки рекомендації LLM можуть залежати від контексту й обмежень платформи.

Перспективами подальшої роботи є розширення Knowledge Base за рахунок історії фактичних виправлень, персоналізація retrieval під політики конкретної команди, накопичення еталонного корпусу findings для об'єктивного оцінювання (precision@k, groundedness, ATC fix rate) та глибша інтеграція в CI/CD і code review процеси.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Checking Quality of ABAP Code with ATC. *SAP Help Portal*.
URL: <https://help.sap.com/docs/abap-cloud/abap-development-tools-user-guide/checking-quality-of-abap-code-with-atc> (date of access: 21.04.2026).
2. Assuring Quality of ABAP Code. *SAP Help Portal*.
URL: <https://help.sap.com/docs/btp/sap-business-technology-platform/assuring-quality-of-abap-code> (date of access: 25.04.2026).
3. Keep Clean Core Governance with ABAP Test Cockpit (ATC). *SAP Help Portal*. URL: <https://help.sap.com/docs/btp/btp-developers-guide/keep-clean-core-governance-with-abap-test-cockpit> (date of access: 23.04.2026).
4. Clean ABAP Style Guide. *SAP*.
URL: <https://github.com/SAP/styleguides/blob/master/clean-abap/CleanABAP.md> (date of access: 23.04.2026).
5. Joule for Developers, ABAP AI Capabilities. *SAP Help Portal*.
URL: <https://help.sap.com/docs/abap-cloud/abap-development-tools-user-guide/joule-for-developers-abap-ai-capabilities> (date of access: 28.04.2026).
6. Generative AI in ABAP Cloud: ABAP AI SDK Powered by Intelligent Scenario Lifecycle Management. *SAP Help Portal*.
URL: <https://help.sap.com/docs/abap-ai/generative-ai-in-abap-cloud/abap-ai-sdk-powered-by-intelligent-scenario-lifecycle-management> (date of access: 23.04.2026).
7. Recommendations and Constraints for Generative AI in ABAP Cloud. *SAP Help Portal*. URL: <https://help.sap.com/docs/abap-cloud/abap-development-tools-user-guide/recommendations-and-constraints> (date of access: 29.04.2026).
8. What Is SAP AI Core? *SAP Help Portal*.
URL: <https://help.sap.com/docs/sap-ai-core/sap-ai-core-service-guide/what-is-sap-ai-core> (date of access: 01.05.2026).

9. Generative AI Hub in SAP AI Core. *SAP Help Portal*.
URL: <https://help.sap.com/docs/sap-ai-core/sap-ai-core-service-guide/generative-ai-hub-in-sap-ai-core> (date of access: 30.04.2026).
10. SAP HANA Cloud, SAP HANA Database Vector Engine Guide. *SAP Help Portal*. URL: <https://help.sap.com/docs/hana-cloud-database/sap-hana-cloud-sap-hana-database-vector-engine-guide/introduction> (date of access: 02.05.2026).
11. Reference Architecture: Retrieval-Augmented Generation. *SAP Architecture Learning*. URL: <https://architecture.learning.sap.com/docs/ref-arch/e5eb3b9b1d/3> (date of access: 29.04.2026).
12. Code Metrics Definitions. *SonarSource*.
URL: <https://docs.sonarsource.com/sonarqube-server/user-guide/code-metrics/metrics-definition> (date of access: 08.05.2026).
13. Analyzing ABAP Source Code in SonarQube. *SonarSource*.
URL: <https://docs.sonarsource.com/sonarqube-server/analyzing-source-code/languages/abap> (date of access: 02.05.2026).
14. Code Review Guide. *OWASP Foundation*.
URL: <https://owasp.org/www-project-code-review-guide/> (date of access: 27.04.2026).
15. Gao Y. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv preprint*.
URL: <https://arxiv.org/html/2405.07437v2> (date of access: 07.05.2026).
16. Li Y. et al. LMs: Understanding Code Syntax and Semantics for Code Analysis. *arXiv preprint*. URL: <https://arxiv.org/abs/2305.12138> (date of access: 11.05.2026).
17. Applying Quick Fixes for ATC Findings. *SAP Help Portal*.
URL: <https://help.sap.com/docs/abap-cloud/abap-development-tools-user-guide/applying-quick-fixes-for-atc-findings> (date of access: 13.05.2026).

18. ABAP Unit Tests and Quality Checking. *SAP Help Portal*.
URL: https://help.sap.com/docs/ABAP_PLATFORM_BW4HANA/c238d694b825421f940829321ffa326a/0ea379274b43db.html (date of access: 26.05.2026).
19. Лариса Ернестівна Чала - ХНУРЕ. *NURE*.
URL: <https://nure.ua/staff/larisa-ernestivna-chala> (дата звернення: 02.04.2026).
20. Золотухін Олег Вікторович - Кафедра штучного інтелекту
ХНУРЕ. *Кафедра штучного інтелекту ХНУРЕ*.
URL: <https://ai.nure.ua/employees/zolotuhin-oleg-viktorovich> (дата звернення: 02.05.2026).
21. Марина Сергіївна Кудрявцева - ХНУРЕ. *NURE*.
URL: <https://nure.ua/staff/marina-sergiyivna-kudryavtseva> (дата звернення: 20.04.2026).
22. Вадим Леонідович Шергін - ХНУРЕ. *NURE*.
URL: <https://nure.ua/staff/vadim-leonidovich-shergin> (дата звернення: 20.05.2026).