НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Семантичний пошук і визначення схожості текстів на основі трансформерів»

Виконала:
студентка IV курсу, групи КІ-21
Коберник Анна Валеріївна _____

Керівник:
доцент кафедри штучного інтелекту, д.ф., доцент
Гуськова Віра Геннадіївна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри штучного інтелекту, к.т.н., доцент,
Комариста Богдана Миколаївна _____

Рецензент:
професор кафедри математичних методів системного аналізу,
д.т.н., професор, Дмитрієва Ольга Анатоліївна _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студентка _____

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«15» січня 2026 р.

ЗАВДАННЯ
на дипломну роботу студенту
Коберник Анні Валеріївні

1. Тема роботи «Семантичний пошук і визначення схожості текстів на основі трансформерів», керівник роботи Гуськова Віра Геннадіївна, доцент кафедри ШІ, доктор філософії, затверджені наказом по НН ІПСА від «27» травня 2026 р. № 1944-с.

2. Термін подання студентом роботи «05» червня 2026 року.

3. Вихідні дані до роботи: дані про бажані характеристики системи семантичного пошуку; англомовний корпус текстових даних AG News для індексації та аналізу; попередньо навчені трансформерні моделі бібліотеки sentence-transformers.

4. Зміст роботи: характеристика предметної області семантичного пошуку та обробки природної мови; огляд методів векторного представлення тексту та трансформерних архітектур; компоненти системи та їх реалізація; порівняльне дослідження трансформерного та класичного підходів до пошуку.

5. Перелік ілюстративного матеріалу: електрона презентація.

6. Дата видачі завдання «02» лютого 2026 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури та аналіз предметної галузі	20.04.2026	Виконано
2	Розробка першого розділу	27.04.2026	Виконано
3	Розробка другого розділу	04.05.2026	Виконано
4	Реалізація програмного продукту	11.05.2026	Виконано
5	Розробка третього розділу	18.05.2026	Виконано
6	Оформлення пояснювальної записки	25.05.2026	Виконано
7	Підготовка презентації до захисту	01.06.2026	Виконано
8	Здача роботи до деканату	05.06.2026	Виконано

Студент Анна КОБЕРНИК

Керівник Віра ГУСЬКОВА

РЕФЕРАТ

Дипломна робота: 79 ст., 16 рис., 11 табл., 33 джерел, 1 додаток.

СЕМАНТИЧНИЙ ПОШУК, ТРАНСФОРМЕРИ, BERT, SENTENCE-TRANSFORMERS, FAISS, TF-IDF, КОСИНУСНА СХОЖІСТЬ, ВЕКТОРНІ ПРЕДСТАВЛЕННЯ, ОБРОБКА ПРИРОДНОЇ МОВИ, PYTHON

Об'єктом дослідження є методи векторного представлення тексту та алгоритми семантичного пошуку на основі трансформерних архітектур. Предметом дослідження є порівняльна ефективність трансформерного та класичного підходів до семантичного пошуку текстів. Метою роботи є розробка системи семантичного пошуку і визначення схожості текстів та кількісне порівняння двох підходів за стандартними метриками якості інформаційного пошуку.

Оглянуто еволюцію методів векторного представлення тексту − від TF-IDF і BM25 через Word2Vec і GloVe до трансформерних архітектур BERT і Sentence-BERT. Проаналізовано методи векторного пошуку на основі FAISS та існуючі промислові рішення.

Розроблено модульну систему семантичного пошуку: п'ять модулів ядра, модуль оцінки якості за шістьма метриками та веб-інтерфейс на Streamlit. Система індексує 50 000 текстів корпусу AG News із використанням моделі all-MiniLM-L6-v2 та FAISS.

Проведено порівняльний аналіз трьох трансформерних моделей на спільному еталоні − обрано all-MiniLM-L6-v2 як оптимальний баланс швидкості і якості. Виконано порівняльну оцінку SBERT та TF-IDF за шістьма метриками при k = 3, 5, 7. При k = 5 трансформерна система досягає P@5 = 0.600, F1@5 = 0.750 і MRR = 1.000 проти 0.060, 0.075 і 0.133 у TF-IDF. Перевага підтверджена за всіма метриками при кожному значенні k, оптимальним визначено k = 5. Час пошуку ~15 мс і точність кластеризації ембедингів понад 85% підтверджують практичну придатність системи.

ABSTRACT

Bachelor's thesis: 79 p., 16 figures, 11 tables, 33 references, 1 appendix.

SEMANTIC SEARCH, TRANSFORMERS, BERT, SENTENCE-TRANSFORMERS, FAISS, TF-IDF, COSINE SIMILARITY, VECTOR REPRESENTATIONS, NATURAL LANGUAGE PROCESSING, PYTHON

The object of the study is methods of vector text representation and semantic search algorithms based on transformer architectures. The subject of research is the comparative effectiveness of transformer-based and classical approaches to text semantic search. The purpose of the work is to develop a system for semantic search and text similarity detection, and to quantitatively compare both approaches using standard information retrieval evaluation metrics.

The evolution of text vector representation methods has been reviewed – from TF-IDF and BM25 through Word2Vec and GloVe to transformer architectures BERT and Sentence-BERT. Efficient vector search methods based on FAISS and existing industrial solutions have been analyzed.

A modular semantic search system has been developed comprising five core modules, a quality evaluation module covering six metrics, and a Streamlit-based web interface. The system indexes 50,000 texts from the AG News corpus using the all-MiniLM-L6-v2 model and FAISS.

A comparative analysis of three transformer models on a shared ground truth has been conducted, and all-MiniLM-L6-v2 has been selected as the optimal balance between speed and quality. SBERT and TF-IDF have been evaluated using six metrics at k = 3, 5, and 7. At k = 5, the transformer system achieves P@5 = 0.600, F1@5 = 0.750, and MRR = 1.000, compared to 0.060, 0.075, and 0.133 for TF-IDF. The advantage holds across all metrics at each value of k, with k = 5 identified as optimal. Search time of ~15 ms and embedding clustering accuracy exceeding 85% confirm the practical suitability of the system.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

AI – Artificial Intelligence, штучний інтелект

BERT – Bidirectional Encoder Representations from Transformers, двобічна трансформерна модель для роботи з текстом

FAISS – Facebook AI Similarity Search, бібліотека для пошуку схожих векторів

IDF – Inverse Document Frequency, зворотна частота документа (для обчислення ваги термінів)

SBERT – Sentence-BERT, модифікація BERT для створення векторів речень

TF – Term Frequency, частота терміна в документі

TF-IDF – Term Frequency – Inverse Document Frequency, модель для зважування термінів у класичному пошуку

AP – Average Precision, середня точність (метрика оцінки пошукових систем)

BEIR – Benchmark for Information Retrieval, гетерогенний бенчмарк для оцінки систем інформаційного пошуку

BM25 – Best Matching 25, імовірнісна модель ранжування документів

CBOW – Continuous Bag of Words, режим навчання Word2Vec

CPU – Central Processing Unit, центральний процесор

CSR – Compressed Sparse Row, формат зберігання розріджених матриць

DCG – Discounted Cumulative Gain, дисконтований сукупний приріст

F1 – F1-міра, гармонійне середнє точності та повноти

GloVe – Global Vectors for Word Representation, модель статичних ембедингів слів

GPU – Graphics Processing Unit, графічний процесор

HNSW – Hierarchical Navigable Small World, алгоритм наближеного пошуку найближчих сусідів

IVF – Inverted File Index, індекс із зворотним файлом для наближеного пошуку

MAP – Mean Average Precision, середня точність по всіх запитах

MLM – Masked Language Modeling, задача навчання BERT на відновленні маскованих токенів

MRR – Mean Reciprocal Rank, середній зворотний ранг

NDCG – Normalized Discounted Cumulative Gain, нормалізований дисконтований сукупний приріст

NLP – Natural Language Processing, обробка природної мови

NSP – Next Sentence Prediction, задача передбачення наступного речення у BERT

PCA – Principal Component Analysis, метод головних компонент

PQ – Product Quantization, метод стиснення векторів

PRF – Pseudo-Relevance Feedback, псевдо-релевантний зворотний зв'язок

QA – Question Answering, система відповідей на запитання

RAG – Retrieval-Augmented Generation, генерація з доповненням пошуком

RR – Reciprocal Rank, зворотний ранг

t-SNE – t-distributed Stochastic Neighbor Embedding, нелінійний метод зниження розмірності

ВСТУП

Масштаби[1] цифрових текстів значно зростають: новини, наукові статті, корпоративна документація, повідомлення у соціальних мережах. Це все ставить перед розробниками інформаційних систем нові виклики. Класичні пошукові системи, побудовані на точному збігу ключових слів, часто не спрацьовують: вони не здатні знаходити документи, семантично близькі до запиту, але сформульовані іншими словами. Людина, що шукає «методи машинного навчання», очікує побачити й результати про «алгоритми ШІ» та «нейронні мережі», однак система на ключових словах поверне тільки документи з цими конкретними словосполученнями [1].

Ситуацію змінила поява трансформерних архітектур. У 2017 році А. Васвані та ін. (Vaswani A. et al.) запропонували модель Transformer на основі механізму уваги, яка стала фундаментом для нового покоління мовних моделей [2]. Моделі BERT і Sentence-BERT навчились кодувати зміст тексту у щільних векторних представленнях – ембедингах, завдяки чому семантично схожі тексти опиняються поруч у векторному просторі. Це відкрило можливість принципово нового підходу до пошуку: замість порівняння слів – порівняння значень.

Актуальність задачі підтверджується як науковим, так і прикладним контекстом. З наукового боку – системи семантичного пошуку є ключовим компонентом архітектур Retrieval-Augmented Generation (RAG), що поєднують пошук релевантних документів із генерацією відповідей великими мовними моделями [4]. Це один із найбільш активних напрямків досліджень у NLP останніх років. З прикладного боку – корпоративні

[1] Тут і нижче використано такі інструменти штучного інтелекту як чат-боти з генеративним штучним інтелектом ChatGPT та Claude.ai виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

платформи, медичні інформаційні системи та пошукові сервіси масово впроваджують семантичний пошук для покращення якості результатів.

Попри широку доступність готових рішень (Elasticsearch, Weaviate, Pinecone) розробка власної системи зберігає значну дослідницьку цінність. Комерційні продукти приховують внутрішню логіку і не дозволяють детально аналізувати поведінку окремих компонентів. Крім того, відсутні загальнодоступні системи, що поєднують трансформерний і класичний підходи в одному прозорому середовищі з кількісним порівнянням за стандартними метриками [1, 15]. Питання вибору оптимальної трансформерної моделі для конкретної задачі також досі вирішується переважно на основі загальних бенчмарків, а не прямого порівняння на цільовому корпусі [29].

Метою цієї дипломної роботи є розробка системи семантичного пошуку і визначення схожості текстів та кількісне порівняння трансформерного і класичного підходів на реальному корпусі текстів за шістьма стандартними метриками якості інформаційного пошуку.

Для досягнення мети вирішено такі задачі: проаналізовано еволюцію методів векторного представлення тексту від TF-IDF і BM25 до трансформерних архітектур; досліджено механізм самоуваги та моделі BERT і Sentence-BERT; проаналізовано методи ефективного векторного пошуку на основі FAISS та існуючі промислові рішення [25, 26]; формалізовано задачу через косинусну схожість нормалізованих ембедингів і розроблено методологію порівняльного аналізу; реалізовано модульну систему з веб-інтерфейсом; проведено порівняльний аналіз трьох трансформерних моделей і обрано оптимальну [29, 30, 31]; виконано порівняльну оцінку трансформерного та класичного підходів за шістьма метриками при трьох значеннях глибини пошуку.

Об'єктом дослідження є методи векторного представлення тексту та алгоритми семантичного пошуку на основі трансформерних архітектур.

Предметом дослідження є порівняльна ефективність трансформерного та класичного підходів до семантичного пошуку текстів.

РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ СЕМАНТИЧНОГО ПОШУКУ ТА ОБРОБКИ ПРИРОДНОЇ МОВИ

1.1 Актуальність задачі семантичного пошуку

Пошук інформації − одна з найстаріших задач інформатики. Ще в 1960-х роках Салтон запропонував векторну модель простору для пошуку документів: кожен текст представляється вектором, а релевантність визначається кутом між вектором запиту та вектором документа. Протягом наступних десятиліть ця ідея еволюціонувала в TF-IDF і BM25, які й сьогодні залишаються базовими методами у промислових системах.

Проте з масовим поширенням Інтернету стало очевидно, що лексичний збіг не гарантує семантичної відповідності. Якщо користувач шукає «серцево-судинні захворювання», а в базі є документи про «хвороби серця» та «кардіопатологію», класична система поверне порожній результат - не тому що релевантних документів немає, а тому що вона не розуміє синонімії [3].

Семантичний пошук (Semantic Search) − це підхід, що спрямований на розуміння змісту запиту та документів, а не лише на виявлення лексичних збігів. Ключовим інструментом є щільні векторні представлення (dense embeddings), що кодують семантику у неперервному векторному просторі. Семантично близькі тексти мають близькі вектори, навіть якщо в них немає спільних слів.

Практична затребуваність семантичного пошуку сьогодні надзвичайно велика. Пошукові системи Google та Bing застосовують нейронні ранжувальники вже понад десятиліття. Корпоративні платформи Confluence, Notion, Microsoft SharePoint впроваджують семантичний пошук у системи управління знаннями. Медичні інформаційні системи використовують його для пошуку клінічних протоколів за описом симптомів. Системи питань і відповідей (QA), чат-боти та асистенти на основі великих мовних моделей

покладаються на семантичний пошук при відборі контексту (Retrieval-Augmented Generation) [4].

Попри широке застосування, розробка якісної системи семантичного пошуку залишається непростим завданням. Необхідно вибрати відповідну модель кодування, спосіб зберігання та індексування векторів, алгоритм пошуку ближніх сусідів і метрики оцінки якості. Цій задачі й присвячена дипломна робота.

1.2 Класичні методи векторного представлення тексту

Щоб комп'ютер міг порівнювати тексти, їх потрібно перетворити на числа. Найпростіший підхід − подати документ як вектор ваг усіх слів у словнику. Саме на цій ідеї було побудовано TF-IDF і BM25, два методи, що залишаються основою класичного пошуку.

1.2.1 Модель TF-IDF

TF-IDF присвоює кожному слову вагу, яка враховує дві речі: наскільки часто слово зустрічається в конкретному документі і наскільки рідко − у всій колекції. Слово, що трапляється в одному документі і майже ніде більше, отримує високу вагу і вважається характерним для цього документа. Слово «і» чи «але» зустрічається скрізь, тобто його вага близька до нуля [5, 23].

1.2.2 Метод BM25

BM25 (Best Matching 25) – це розвиток ідеї TF-IDF у рамках ймовірнісного підходу до пошуку. Назва «25» означає, що це двадцять п'ята ітерація моделі, розробленої Стівеном Робертсоном і Кареном Спарк Джонс. Алгоритм вирішує два принципових недоліки TF-IDF. По-перше, він враховує насичення, тобто якщо слово зустрілося десять разів замість одного, це вже не означає, що документ у десять разів більш релевантний. Як результат, корисність кожного додаткового входження зменшується. По-друге, BM25 нормалізує довжину документа. Довгий текст не отримує автоматичної переваги лише тому, що слів у ньому більше [6, 24].

Обидва методи є так званими bag-of-words підходами: вони не враховують порядок слів, синоніми і контекст. Запит «автомобіль» не знайде документ зі словом «машина», навіть якщо мова йде про одне й те саме. Саме це обмеження стало поштовхом до розробки семантичних методів, розглянутих у наступних підрозділах.

Математична постановка обох методів, включно з повними формулами і параметрами, наведена у розділі 2.

1.2.3 Статичні векторні представлення слів: Word2Vec і GloVe

З моменту появи моделей Word2Vec Т. Міколова та ін. (T. Mikolov et al.) і GloVe Дж. Пеннінгтона та ін. (J. Pennington et al.), спосіб представлення тексту значно змінився. Ці моделі перетворюють слова у щільний векторний простір розмірністю 100–300 [7, 8]. Їхньою головною властивістю є семантично схожі слова, які мають близькі вектори. Наприклад, такий вираз

king − man + woman ≈ queen показує, що геометрія векторного простору реально вловлює смислові зв’язки.

Word2Vec має два режими навчання, а саме Skip-gram (передбачаємо контекст за словом) і CBOW (навпаки − передбачаємо слово за контекстом). GloVe працює інакше − він мінімізує різницю між скалярним добутком векторів і логарифмом частоти спільної появи слів.

Проблема статичних ембедингів полягає в тому, що кожне слово має єдиний фіксований вектор незалежно від контексту. Слово «коса» у реченнях «дівчина заплела косу» і «рибалки вийшли на косу» матиме однаковий вектор, хоча мають дуже різні значення. Для документів статичні ембединги зазвичай усереднюють, що призводить до втрати порядку слів і тонких семантичних відтінків [9].

1.3 Трансформерна архітектура та контекстуальні представлення

Ситуацію змінила поява трансформерів − з них почався новий етап у семантичному пошуку. У 2017 році А. Васвані та ін. запропонували архітектуру Transformer, засновану виключно на механізмах уваги, без рекурентних або згортальних операцій [2]. Вона стала основою для BERT, GPT та сотень інших моделей.

1.3.1 Механізм уваги

Ключовий будівельний блок трансформера − механізм самоуваги (self-attention). Для кожного токена послідовності обчислюються три вектори:

Query (Q), Key (K) і Value (V) як лінійні перетворення вхідного ембединга. Оцінка уваги:

$$Attention\,(Q,K,V) = softmax\left(\frac{QK^{T}}{\sqrt{d_k}}\right)\cdot V, \tag{1.5}$$

де d_k – розмірність векторів ключів; ділення на $\sqrt{d_k}$ запобігає насиченню функції softmax при великих значеннях скалярного добутку.

На практиці використовується багатоголова увага (Multi-Head Attention). Тобто паралельно обчислюються h незалежних механізмів уваги, результати конкатенуються та лінійно проєктуються. Це дозволяє моделі одночасно враховувати різні типи залежностей між токенами, наприклад синтаксичні, семантичні, довгодистанційні.

Перевага механізму уваги - пряме моделювання залежностей між будь-якими двома позиціями послідовності незалежно від відстані. Рекурентні мережі LSTM та GRU мають складність $O(n)$ для зв'язків між далекими позиціями, а трансформер дає $O(1)$. Це критично для розуміння складних семантичних структур.

1.3.2 Модель BERT

У 2018 році Google представила BERT (Bidirectional Encoder Representations from Transformers), що сильно вплинуло на NLP [10]. BERT – двонаправлений трансформерний енкодер, що навчається на двох задачах.

Перша задача – Masked Language Modeling (MLM). Маскуються 15% токенів вхідного тексту, і модель навчається їх відновлювати на основі оточення з обох боків. Це забезпечує глибоке двонаправлене розуміння

контексту − принципова відмінність від авторегресивних моделей на зразок GPT.

Друга задача − Next Sentence Prediction (NSP). Модель класифікує, чи є два речення суміжними в тексті. Це навчає модель розуміти зв'язки між реченнями.

Завдяки двонаправленості BERT генерує контекстуальні ембедінги. Один і той самий токен отримує різні вектори залежно від контексту. Базова версія BERT-Base містить 12 шарів трансформерів, 12 голів уваги і 110 млн параметрів, а BERT-Large − 24 шари та 340 млн параметрів [10].

Для задачі семантичного пошуку ключовий вектор − представлення спеціального токена [CLS] або усереднення всіх токенів (mean pooling). Після попереднього навчання модель можна донавчити на конкретній задачі (fine-tuning) з меншою кількістю прикладів [32, 33].

1.3.3 Sentence-BERT і моделі для семантичного пошуку

Незважаючи на якість BERT, пряме застосування для пошуку виявилося неефективним. Стандартний підхід - подавати пару (запит, документ) разом на вхід BERT − вимагає N повних прогонів моделі при N документах. При N = 50 000 і часі кодування 100 мс на пару це дає 5 000 секунд − неприйнятно.

У 2019 році Н. Реймерс та І. Гуревич (Reimers N., Gurevych I.) запропонували Sentence-BERT (SBERT) − рішення, що зберігає якість BERT при лінійній складності пошуку [11]. Ключова ідея − сіамська архітектура (Siamese Network). Тобто обидва тексти кодуються незалежно одними й тими самими вагами, а результуючі ембедінги порівнюються за косинусною

схожістю. Навчання відбувається на наборах Natural Language Inference (NLI) з мітками «суперечність/нейтральність/зв'язок».

Завдяки цьому підходу всі документи корпусу можна закодувати заздалегідь і зберегти. Пошук зводиться до одного кодування запиту та пошуку ближніх сусідів у векторному просторі − операцій, що виконуються за мілісекунди навіть для мільйонних колекцій.

1.4 Бібліотека Sentence-Transformers та попередньо навчені моделі

Бібліотека sentence-transformers − де-факто стандарт для отримання якісних ембедингів речень у Python. Вона надає уніфікований API для понад 100 попередньо навчених моделей різних розмірів та призначень [12, 29].

Порівняльний аналіз ключових моделей бібліотеки представлено у таблиці 1.1.

Таблиця 1.1 − Порівняльний аналіз моделей бібліотеки sentence-transformers

Модель	*Розмір (МБ)*	*Розмірність*	*Швидкість (речень/с)*	*Оцінка якості (SBERT bench.)*
all-MiniLM-L6-v2	80	384	~14 200	58.8
all-MiniLM-L12-v2	120	384	~7 500	59.8
all-mpnet-base-v2	420	768	~2 800	63.3
paraphrase-MiniLM	80	384	~14 200	57.0

Для цієї роботи обрано модель all-MiniLM-L6-v2 − оптимальний баланс між якістю й швидкістю [30]. Лише 6 шарів трансформерів дають п'ятикратне

прискорення порівняно з BERT-Base при незначній втраті якості. Розмірність ембединга 384 забезпечує компактність індексу та прийнятний час пошуку. Серед альтернатив − all-mpnet-base-v2 [31], що забезпечує вищу якість ранжування ціною більшого розміру моделі та нижчої швидкості кодування.

Практична перевірка цього вибору через порівняльний експеримент на корпусі AG News наведена у підрозділі 3.7.

1.5 Методи ефективного пошуку за векторними представленнями

Отримання якісних ембедингів − лише перший крок. Для реального пошуку потрібен ефективний алгоритм пошуку ближніх сусідів (Nearest Neighbor Search, NNS) у векторному просторі.

Бібліотека FAISS (Facebook AI Similarity Search) від Meta AI − відкрита бібліотека для ефективного пошуку подібних векторів у великих колекціях [13, 25]. Вона реалізує як точний, так і наближений пошук.

Клас IndexFlatIP реалізує точний пошук через скалярний добуток із BLAS-оптимізацією. При нормалізованих векторах скалярний добуток дорівнює косинусній схожості. Для колекцій до 1 мільйона документів розмірністю 384 пошук займає менш ніж 50 мілісекунд, що є цілком добре для задач реального часу.

Для більших колекцій FAISS пропонує наближені методи, що жертвують незначною точністю заради значного прискорення. Метод IVF (Inverted File Index) розбиває векторний простір на кластери і шукає лише в найближчих кластерах. Алгоритм HNSW (Hierarchical Navigable Small World) будує граф навігації з логарифмічною складністю пошуку. Метод PQ (Product Quantization) стискає вектори для зменшення об'єму пам'яті в 8−32 рази.

Для корпусу з 50 000 документів, що використовується в цій роботі, IndexFlatIP забезпечує повну точність і задовільну швидкість, тому наближені методи не застосовуються [13, 25].

Порівняльний аналіз алгоритмів пошуку найближчих сусідів за ключовими характеристиками показано в таблиці 1.2.

Таблиця 1.2 – Порівняльний аналіз алгоритмів пошуку за векторними представленнями

Алгоритм	*Точність*	*Швидкість пошуку*	*Пам'ять*	*Застосування*
IndexFlatIP	100 %	$O(n)$	Висока	До ~1 млн документів
IVF	~95–99 %	$O(\sqrt{n})$	Середня	Від 1 млн документів
HNSW	~95–99 %	$O(\log n)$	Висока	Висока швидкість пошуку
PQ	~90–95 %	$O(n/m)$	Низька (×8−32)	Обмежена пам'ять

1.6 Огляд існуючих систем та рішень

Перед проєктуванням власної системи важливо розуміти, які готові рішення існують і чим вони відрізняються одне від одного.

Основною мовою розробки для задач NLP і семантичного пошуку є Python – завдяки розвиненій екосистемі бібліотек: sentence-transformers, FAISS, scikit-learn, NumPy. Альтернативами є Java (Apache Lucene, Elasticsearch-клієнт) та Go (для високопродуктивних мікросервісів).

Серед спеціалізованих фреймворків варто виділити LangChain і LlamaIndex – інструменти для побудови RAG-систем з вбудованою

підтримкою векторних сховищ. Chroma − легковагова відкрита база для локального розгортання. Qdrant − виробнича система з підтримкою фільтрації та гібридного пошуку [26].

Elasticsearch із модулем dense_vector − найпоширеніший у промисловості стек для гібридного пошуку. Поєднує BM25 і семантичний пошук через механізм kNN-запитів. Переваги: горизонтальне масштабування, повноцінна екосистема, висока відмовостійкість. Недоліки: складна конфігурація, потребує окремої інфраструктури, висока вартість для малих проєктів [14].

Weaviate і Pinecone − хмарні векторні бази даних з REST/gRPC API. Надають зручний інтерфейс і масштабуються до мільярдів векторів. Недолік - залежність від зовнішніх сервісів і хмарна модель оплати.

Haystack (deepset) − фреймворк для побудови пошукових конвеєрів (pipelines) на основі трансформерів. Компоненти Retriever і Reader дозволяють будувати QA-системи. Потужний, але з високим порогом входження.

З комерційних продуктів задачу семантичного пошуку вирішують Algolia (хмарний пошук з нейронним ранжуванням), Coveo (корпоративний семантичний пошук), Azure Cognitive Search (Microsoft) та Amazon Kendra (AWS). Усі є платними хмарними сервісами з моделлю оплати за запит [27].

Ринок векторних баз даних зріс з 1.73 млрд доларів у 2024 році до прогнозованих 10.6 млрд до 2032 року, що відображає стрімке впровадження семантичного пошуку у промислових застосунках [26, 27].

Розроблена у цій роботі система є автономним рішенням без залежностей від хмарних сервісів, з прозорою архітектурою для навчальних і прикладних задач середнього масштабу. Принципова відмінність від більшості готових рішень − наявність модуля кількісного порівняння двох підходів за шістьма метриками.

1.7 Постановка задачі

На основі аналізу предметної галузі сформульовано задачу дипломної роботи. Необхідно розробити систему, що виконує дві функції: семантичний пошук документів за текстовим запитом та визначення ступеня семантичної схожості між довільними парами текстів.

Система має відповідати таким вимогам:

– підтримка корпусу щонайменше 50 000 документів із часом пошуку не більше 100 мс на запит;

– реалізація трансформерного модуля (Sentence-BERT та FAISS) та класичного модуля (TF-IDF) для порівняльного аналізу;

– оцінка якості за не менш ніж за чотирма метриками: Precision@K, MAP, MRR, NDCG@K;

– наявність зручного веб-інтерфейсу для демонстрації функціоналу;

– модульна архітектура, що допускає заміну компонентів без переписування системи.

1.8 Математична постановка задачі семантичного пошуку

Задачу семантичного пошуку можна формально визначити поданим нижче чином. Нехай $D = \{d^1, d^2, ..., d_N\}$ – колекція з N текстових документів. Задано функцію кодування $\varphi: T \rightarrow \mathbb{R}^d$, що відображає довільний текст t у d-вимірний евклідів простір. Для заданого запиту q функція пошуку повертає впорядкований список документів:

$$Search(q, D, k) = top - k\ arg\ max\ sim(\varphi(q), \varphi(d_i)),\ d_i \in D, \qquad (1.1)$$

де $sim(\varphi(q), \varphi(d_i)$ − функція схожості; k − кількість документів у результаті.

Як функцію схожості обрано косинусну, що вимірює кут між векторами:

$$cos(u, v) = \frac{(u \cdot v)}{||u|| \cdot ||v||}. \quad (1.2)$$

При нормалізованих векторах ($||\varphi(t)|| = 1$) косинусна схожість збігається зі скалярним добутком: $cos(u, v) = u \cdot v$. Це важливо для роботи з FAISS IndexFlatIP, що реалізує пошук за скалярним добутком. Нормалізація виконується під час кодування параметром normalize_embeddings=True у sentence-transformers.

Задача визначення схожості текстів є частковим випадком. Для двох довільних текстів t_1 і t_2 потрібно обчислити оцінку $sim(\varphi(t_1), \varphi(t_2)) \in [-1, 1]$. При нормалізованих векторах значення 1 означає ідентичний зміст, 0 – семантичну ортогональність, -1 − протилежне значення. Для текстів природної мови значення нижче 0 рідкісні.

Для TF-IDF аналогічна функція кодування − це розріджений вектор у просторі термів з вагами TF-IDF. Пошук зводиться до скалярного добутку вектора запиту і матриці документів, що є ефективною операцією над розрідженою матрицею (CSR-формат).

1.9 Архітектурні рішення системи

На основі аналізу вимог, наведеного в підрозділі 1.7, та огляду існуючих рішень прийнято такі архітектурні рішення.

Система реалізована як модульний Python-пакет. Кожен компонент – окремий клас з чітко визначеним інтерфейсом. Це забезпечує тестованість, можливість заміни окремих модулів та зрозумілу структуру коду.

Трансформерний модуль побудовано на стеку sentence-transformers + FAISS. Перший відповідає за кодування текстів у векторний простір, другий – за ефективний пошук ближніх сусідів. Для поточного масштабу (50 000 документів) обрано точний пошук IndexFlatIP.

Персистентність: FAISS-індекс зберігається на диск після першого кодування. При наступних запусках завантажується готовий індекс, що усуває необхідність повторного кодування (5–15 хвилин). Тексти зберігаються у форматі NumPy [18].

Веб-інтерфейс реалізовано на Streamlit [19]. Перевагами є: нативна інтеграція з Python без JavaScript; декоратор @st.cache_resource для кешування моделі між запитами; вбудована підтримка графіків matplotlib; швидке розгортання.

Порівняльний аналіз технологічних рішень представлено таблиці 1.3.

Таблиця 1.3 – Вибір технологічних рішень системи

Компонент	*Обране рішення*	*Альтернативи*	*Обґрунтування вибору*
Кодування тексту	all-MiniLM-L6-v2	BERT, mpnet-base-v2	Баланс якість / швидкість
Векторний індекс	FAISS IndexFlatIP	HNSW, Pinecone	Точність, автономність
Класичний пошук	TF-IDF (sklearn)	BM25 (rank_bm25)	Стандартний baseline
Веб-інтерфейс	Streamlit	Flask, FastAPI+React	Швидкість розробки
Мова	Python 3.10+	Java, Go	Екосистема NLP/ML

Висновки до розділу 1

У першому розділі було проаналізовано еволюцію методів семантичного пошуку від класичних підходів (TF-IDF, BM25) до сучасних нейронних архітектур. Встановлено, що TF-IDF та BM25 – ефективні baseline-методи, проте вони не здатні враховувати семантичний зміст тексту поза лексичними збігами. Статичні ембединги Word2Vec і GloVe подолали проблему синонімії, але не контекстуальної полісемії – одне слово, різні значення.

Трансформерна архітектура з механізмом самоуваги та модель BERT забезпечують контекстуальні ембединги, де значення токена залежить від контексту. Sentence-BERT адаптує BERT для ефективного порівняння речень через сіамську архітектуру. Бібліотека sentence-transformers надає готовий інструментарій для отримання якісних ембедингів, а FAISS забезпечує ефективне індексування і пошук у векторному просторі. Сформульовано технічні вимоги до системи та наведено математичну постановку задачі семантичного пошуку.

Оглянуто існуючі промислові рішення і мови програмування для реалізації семантичного пошуку. На основі цього аналізу та порівняльного дослідження трьох трансформерних моделей обґрунтовано архітектурні рішення системи: модульний Python-пакет, модель all-MiniLM-L6-v2, FAISS IndexFlatIP для точного пошуку та веб-інтерфейс на Streamlit.

РОЗДІЛ 2 МЕТОДИ ТА МЕТРИКИ ОЦІНКИ ЯКОСТІ СЕМАНТИЧНОГО ПОШУКУ

2.1 Математична постановка класичних методів

Спершу розглянемо класичні методи. Вони є теоретичним підґрунтям сучасних підходів і точкою відліку для порівняння.

2.1.1 Модель TF-IDF

TF-IDF (Term Frequency − Inverse Document Frequency) − один із поширеніших методів статистичного представлення тексту. Метод оцінює важливість терму t у документі d відносно всієї колекції D [5].

Компонент TF(t, d) вимірює, наскільки часто терм зустрічається в документі. Базовою формулою вважається відношення кількості входжень до загальної кількості термів. Проте на практиці частіше використовують логарифмічне масштабування − воно зменшує вплив дуже частих слів:

$$TF(t,d) = 1 + log\ f(t,d), \tag{2.1}$$

де f(t, d) − кількість входжень терму t у документ d.

Компонент IDF(t, D) характеризує, наскільки рідко терм зустрічається у всій колекції. Рідкісні терми зазвичай більш інформативні:

$$IDF(t,D) = log(\frac{|D|}{|\{d \in D : t \in d\}|}), \tag{2.2}$$

де |D| − кількість документів у колекції; |{d ∈ D: t ∈ d}| − кількість документів, що містять терм t.

Підсумкова вага терму в документі:

$$TF - IDF(t,d,D) = TF(t,d) \times IDF(t,D). \quad (2.3)$$

Документ представляється вектором у просторі всіх термів, де кожна координата − це вага відповідного терму. Пошук зводиться до обчислення косинусної схожості між вектором запиту та векторами документів.

Головні переваги TF-IDF − простота, інтерпретованість та обчислювальна ефективність. Метод добре працює, коли запит і документи використовують однакові слова. Головний недолік − відсутність семантичного розуміння: синоніми, пов'язані поняття та парафрази залишаються невидимими для моделі [5].

2.1.2 Метод BM25

BM25 (Best Match 25) є вдосконаленням TF-IDF і залишається стандартом для першого етапу ранжування у промислових системах, зокрема в Elasticsearch та Apache Solr [6].

На відміну від базового TF-IDF, BM25 вирішує дві проблеми: насичення частоти (коли кількість входжень терму (f(t, d)) дуже велика, приріст оцінки зменшується) та нечутливість до довжини документа. Оцінка BM25 для терму t у документі d:

$$BM25(t,d) = IDF(t) \cdot \frac{f(t,d) \cdot (k_1+1)}{f(t,d) + k_1 \cdot (1-b+b \cdot \frac{|d|}{avgdl})}, \quad (2.4)$$

де k_1 – параметр насичення (типово 1.2); b – параметр нормалізації довжини (типово 0.75); |d| – довжина документа в токенах; avgdl – середня довжина документа у колекції.

Покращення справді є, якщо порівнювати з базовим підходом TF-IDF. Але навіть BM25 працює лише з ключовими словами і не може подолати проблему різниці між змістом пошукового запиту і змістом документів.

Порівняльний аналіз класичних підходів за ключовими характеристиками показано в таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз класичних методів пошуку

Критерій	*TF-IDF*	*BM25*
Облік насичення частоти	Ні	Так
Нормалізація за довжиною	Часткова	Повна
Кількість гіперпараметрів	0	2 (k_1, b)
Семантичне розуміння	Ні	Ні
Типове застосування	Baseline	Elasticsearch, Solr

2.2 Метрики оцінки якості інформаційно-пошукових систем

Оцінювати пошукові системи – річ непроста. Релевантність залежить від людини й контексту, тому тут немає єдиної правильної відповіді. Для кількісного порівняння в спільноті TREC (Text REtrieval Conference) розробили стандартні метрики [15]. Я зупинилася на шести – вони показують

якість ранжування з різних боків: точність (Precision@K), повноту (Recall@K), середню точність (MAP), середній зворотний ранг (MRR), NDCG@K та F1@K.

2.2.1 Precision@K і Recall@K

Precision@K (Точність при глибині K) – частка релевантних документів серед перших K результатів:

$$P@K = \frac{|R_k \cap Rel|}{K}, \tag{2.5}$$

де R_k – множина перших K повернених документів; Rel – множина всіх релевантних документів у колекції.

Recall@K (Повнота при глибині K) – частка знайдених релевантних документів серед усіх релевантних:

$$R@K = \frac{|R_k \cap Rel|}{|Rel|}. \tag{2.6}$$

F1@K – гармонійне середнє P@K та R@K:

$$F1@K = \frac{2 \cdot P@K \cdot R@K}{(P@K + R@K)}. \tag{2.7}$$

P@K відображає якість топ-K результатів з точки зору користувача. R@K важлива для задач, де необхідно знайти якомога більше релевантних документів (наприклад, юридичний чи медичний пошук). F1@K збалансовує

обидва аспекти. Для стандартного пошукового сценарію в цій роботі обрано K = 5.

2.2.2 Mean Average Precision (MAP)

Average Precision (AP) обчислює площу під кривою Precision-Recall, враховуючи позицію кожного релевантного документа:

$$AP = \left(\frac{1}{|Rel|}\right) \cdot \sum_{k=1}^{N} P@k \cdot rel\,(k),\ k = 1, \ldots, N, \tag{2.8}$$

де rel(k) = 1, якщо документ на позиції k є релевантним, і rel(k) = 0 інакше.

MAP (Mean Average Precision) − середнє значення AP по всій множині запитів |Q|:

$$MAP = \frac{1}{|Q|} \cdot \sum_{i=1}^{|Q|} AP(q_i),\ q_i \in Q. \tag{2.9}$$

MAP вважається золотим стандартом оцінки пошукових систем у спільноті TREC. На відміну від P@K, вона враховує не тільки кількість релевантних документів у топі, а й їхні точні позиції.

2.2.3 Mean Reciprocal Rank (MRR)

Reciprocal Rank (RR) відображає, на якій позиції знаходиться перший релевантний документ:

$$RR = \frac{1}{rank_1}, \tag{2.10}$$

де $rank_1$ – позиція першого релевантного документа. Якщо жодного релевантного не знайдено, RR = 0.

MRR (Mean Reciprocal Rank) – середнє по всіх запитах:

$$MRR = \frac{1}{|Q|} \cdot \sum_{i=1}^{|Q|} RR(q_i). \tag{2.11}$$

MRR добре підходить тоді, коли користувач шукає один конкретний документ (наприклад, визначення терміна). Якщо перший релевантний на позиції 1, то RR = 1.0; на позиції 2 – RR = 0.5; на позиції 5 – RR = 0.2. Висока MRR означає, що найрелевантніший результат зазвичай з'являється в першій або другій позиції.

2.2.4 Normalized Discounted Cumulative Gain (NDCG@K)

Discounted Cumulative Gain (DCG@K) враховує позицію кожного релевантного документа через логарифмічний дисконт:

$$DCG@K = \sum_{i=1}^{K} \frac{rel(i)}{log_2(i+1)}, \; i = 1, \ldots, K. \tag{2.12}$$

Документ на першій позиції дає внесок $\frac{rel(1)}{log_2(2)} = rel(1)$; на другій – $\frac{rel(2)}{log_2(3)} \approx 0.63 \cdot rel(2)$; на п'ятій – $\frac{rel(5)}{log_2(6)} \approx 0.39 \cdot rel(5)$. IDCG@K – це показник

для ідеальної ситуації: усі релевантні документи опинилися вгорі списку NDCG@K нормалізує DCG@K:

$$NDCG@K = \frac{DCG@K}{IDCG@K}. \quad (2.13)$$

$NDCG@K \in [0, 1]$. Ця метрика є найбільш повною з розглянутих, оскільки враховує одночасно кількість знайдених релевантних документів і якість їхнього позиціонування у рейтинговому списку [16].

2.3 Стратегія формування ground truth

Повноцінна оцінка пошукової системи потребує анотованого набору запитів із відомими релевантними документами − так званого ground truth. Ручна розмітка, де експерти оцінюють релевантність кожного документа до кожного запиту, є золотим стандартом, але потребує значних людських ресурсів. Для корпусу з 50 000 документів і 10 запитів це потенційно 500 000 оцінок.

У цій роботі застосовано підхід псевдо-релевантності (Pseudo-Relevance Feedback, PRF) − усталений метод автоматичного формування еталону без ручної розмітки [28]. Основна ідея: для кожного запиту q топ-K документів, повернених сильнішою системою, вважаються релевантними:

$$Rel(q) = \left\{ d \in D : rank_{transformer(d, q)} \leq K \right\},$$

де K = 3 у цій роботі. Ці документи утворюють ground truth для обчислення метрик обох систем.

Математичне обґрунтування спирається на властивість стабільності відносного ранжування, досліджену Вурхіз [17]: якщо система S_1 перевершує систему S_2 за метриками на одному наборі релевантних суджень Rel_1, це відношення зберігається і при використанні іншого набору Rel_2. Формально:

якщо $MAP(S_1, Rel_1) > MAP(S_2, Rel_1)$, то $MAP(S_1, Rel_2) > MAP(S_2, Rel_2)$.

Це означає, що навіть неідеальний еталон дає коректне відносне порівняння систем.

Обмеження підходу: метрики трансформерної системи на власному еталоні штучно дорівнюють 1.0 − вона сама є мірилом. Підхід оцінює не абсолютну якість трансформера, а ступінь лексико-семантичного збігу між двома системами: TF-IDF відтворює лише 6−13 % результатів семантичного пошуку залежно від метрики. Для абсолютної оцінки потрібні зовнішні бенчмарки типу BEIR [18] або ручна розмітка, проте для порівняльного аналізу в межах цієї роботи обраний метод є достатнім.

2.4 Методологія порівняльного аналізу

Для коректного порівняння двох систем розроблено методологію, що забезпечує рівність умов.

Ідентичний корпус: обидві системи індексують одні й ті самі 50 000 текстів AG News. Ідентичні запити: набір із 10 тематично різноманітних запитів, що охоплюють усі чотири категорії датасету (World, Sports, Business, Science/Technology). Фіксована глибина: K = 5 для метрик типу @K. Фіксований ground truth: топ-3 результати трансформерної системи. Усі шість метрик: P@5, R@5, F1@5, MAP, MRR, NDCG@5 для повноти аналізу.

Для забезпечення відтворюваності всі параметри систем зафіксовано до початку експерименту. Запити підібрано так, щоб охопити різний ступінь семантичної складності: від точних збігів («Apple iPhone new features») до

суто семантичних («artificial intelligence technology»), де TF-IDF і трансформер мають принципово різну поведінку. Повний код оцінювання доступний у Додатку А.

Порівняльний аналіз технологічних рішень показано в таблиці 2.2.

Таблиця 2.2 – Вибір технологічних рішень системи

Параметр	*Значення*	*Обґрунтування*
Корпус	AG News, 50 000 текстів	Однаковий для обох систем
Кількість запитів	10	Охоплюють усі 4 категорії
Глибина пошуку K	5	Стандартна для IR-досліджень
Ground truth depth	топ-3 трансформера	Псевдо-релевантність
Кількість метрик	6 (P@5, R@5, F1@5, MAP, MRR, NDCG@5)	Повний профіль якості
Параметри TF-IDF	max_features=10000, ngram=(1,2), sublinear_tf=True	Зафіксовано для відтворюваності
Параметри SBERT	all-MiniLM-L6-v2, normalize_embeddings=True	Зафіксовано для відтворюваності

Висновки до розділу 2

У другому розділі деталізовано задачу семантичного пошуку та визначення схожості текстів. Основною математичною ідеєю є косинусна

схожість нормалізованих ембедингів, що зводить задачу пошуку до знаходження найближчих сусідів за скалярним добутком. Було проаналізовано шість метрик оцінки якості: P@K, R@K, F1@K, MAP, MRR та NDCG@K. Кожна метрика оцінює різний аспект якості пошуку, тому їхнє комплексне використання дає повноцінну картину.

Обґрунтовано стратегію ground truth без ручної розмітки та методологію рівноправного порівняння двох підходів на ідентичному корпусі й ідентичних запитах.

РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ СЕМАНТИЧНОГО ПОШУКУ ТА ОЦІНКА ЕФЕКТИВНОСТІ

3.1 Опис датасету AG News

Для експериментів використано датасет AG News – популярний бенчмарк для класифікації та пошуку текстів [20].

Він містить новини чотирьох категорій: World, Sports, Business, Science/Technology. Джерела – понад 2000 ресурсів.

Характеристики датасету представлено у таблиці 3.1.

Таблиця 3.1 – Характеристики датасету AG News

Параметр	*Значення*
Кількість документів (train / test)	120 000 / 7 600
Кількість категорій	4 (World, Sports, Business, Sci/Tech)
Середня довжина тексту (слів)	~45
Мова	Англійська
Розміщення	Hugging Face Datasets (huggingface.co/datasets/ag_news)
Використовується у роботі	50 000 (перші документи з train)

Індексуються перші 50 000 текстів тренувального набору. Усі категорії представлені рівномірно (по 12 500 документів). Це дає тематичну різноманітність і дозволяє тестувати систему на різних запитах.

Кожен текст є заголовком і першим абзацом новинної статті (рисунок 3.1).

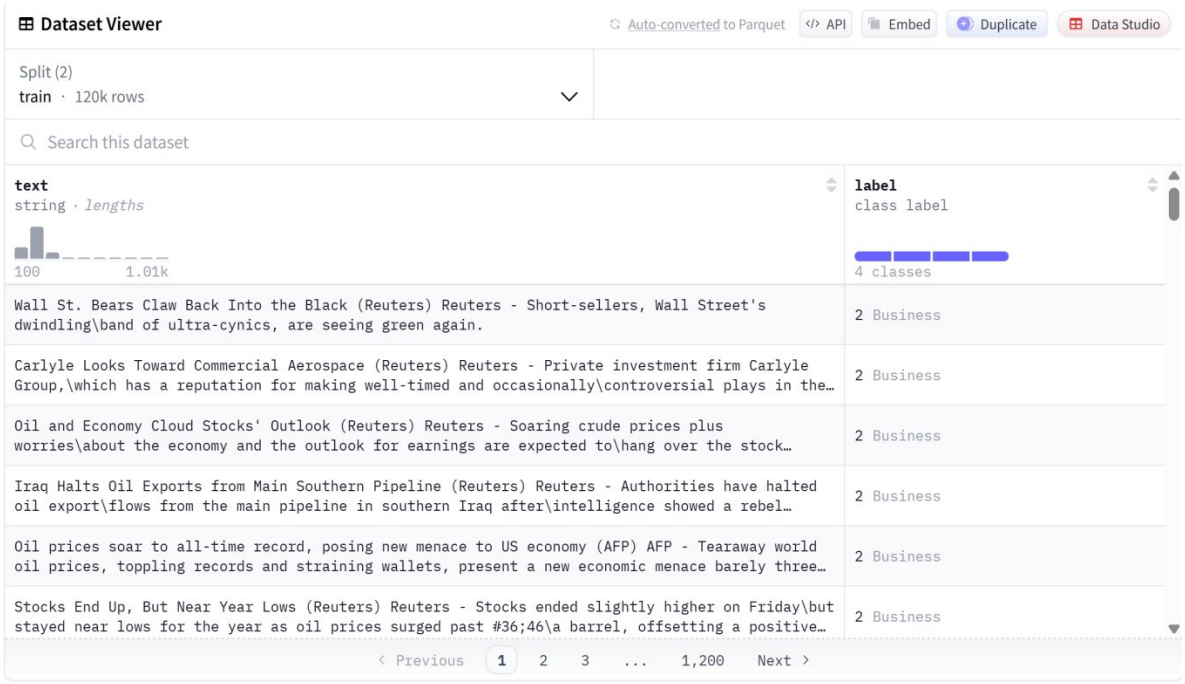

Рисунок 3.1 – Приклад записів датасету AG News (перші 6 рядків)

Така довжина оптимальна для кодування трансформером: достатньо семантичної інформації, але не надто довго для ефективного кодування. Модель all-MiniLM-L6-v2 має обмеження 256 токенів на вхід, що покриває переважну більшість текстів датасету.

Датасет завантажується через бібліотеку Hugging Face datasets і кешується локально у форматі NumPy (файл texts_cache.npy) для прискорення повторних запусків.

3.2 Структура програмного комплексу

Розроблена система організована як Python-пакет із чіткою модульною структурою. Кожен клас відповідає за один аспект системи, що відповідає принципу єдиної відповідальності (Single Responsibility Principle).

Пакет core містить п'ять модулів:

– data_loader.py - клас DataLoader: завантаження AG News через Hugging Face datasets і локальне кешування;

– semantic_search.py - клас SemanticSearchModel: трансформерне кодування, побудова FAISS-індексу, пошук і метод отримання ембедингів;

– tfidf_search.py - клас TFIDFSearch: побудова TF-IDF-матриці, пошук через CSR-множення;

– similarity.py - клас SimilarityModel: попарна схожість, матриця N×N, ранжування кандидатів;

– builder.py - фабрична функція build_system(), що ініціалізує всю систему з перевіркою кешу.

Зовнішні модулі: evaluation.py – обчислення шести метрик і клас EvaluationReport; visualization.py – п'ять типів графіків; app.py – Streamlit-застосунок із чотирма вкладками; evaluate.py – автономний скрипт оцінки. compare_transformers.py – скрипт для порівняльного аналізу трьох трансформерних моделей (all-MiniLM-L6-v2, all-MiniLM-L12-v2, all-mpnet-base-v2).

Схема взаємодії модулів системи наведена на рисунку 3.2.

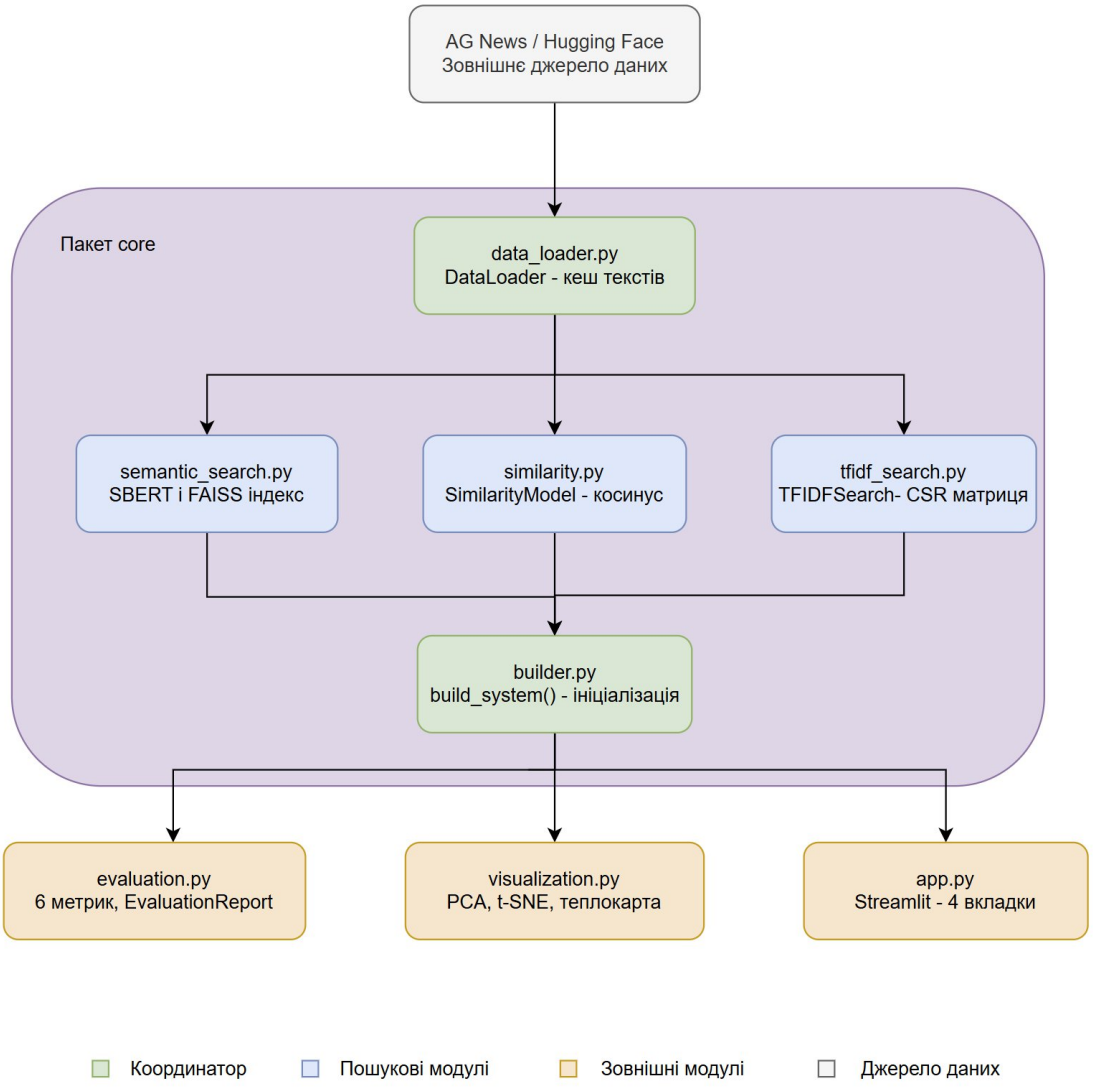

Рисунок 3.2 – Схема взаємодії модулів системи

3.3 Реалізація трансформерного пошукового модуля

Клас SemanticSearchModel є центральним компонентом системи. Нижче наведено детальний опис його методів (рисунок 3.3).

Ініціалізація (метод __init__). Приймає три параметри: назву моделі (model_name), шлях до файлу FAISS-індексу (index_path) та шлях до масиву

текстів (texts_path). Завантажує модель SentenceTransformer один раз при старті – наступні виклики методів не потребують повторного завантаження.

Побудова індексу (метод fit). Алгоритм роботи методу:

– тексти кодуються батчами розміром 256 з нормалізацією до одиничної довжини (normalize_embeddings=True);

– розмірність ембедингу d = 384 визначається автоматично з першого батчу;

– створюється FAISS IndexFlatIP розмірності d;

– ембединги додаються до індексу методом add();

– індекс і тексти зберігаються на диск через faiss.write_index() і np.save().

Пошук (метод search). Приймає запит і параметр k. Кодує запит у вектор, виконує IndexFlatIP.search(), повертає k найближчих документів з полями text, score і index. Вся операція займає ~15 мс.

Кодування (метод encode). Публічний метод для отримання ембедингів довільних текстів. Використовується в модулях схожості та візуалізації.

Завантаження (метод load). Перевіряє наявність файлів на диску і завантажує готовий індекс. Якщо файлів немає - кидає FileNotFoundError з підказкою викликати fit().

Ключова технічна деталь: FAISS IndexFlatIP реалізує точний пошук за скалярним добутком. Для нормалізованих векторів (||v|| = 1) це еквівалентно косинусній схожості. Тому нормалізація під час кодування є обов'язковою умовою коректності результатів.

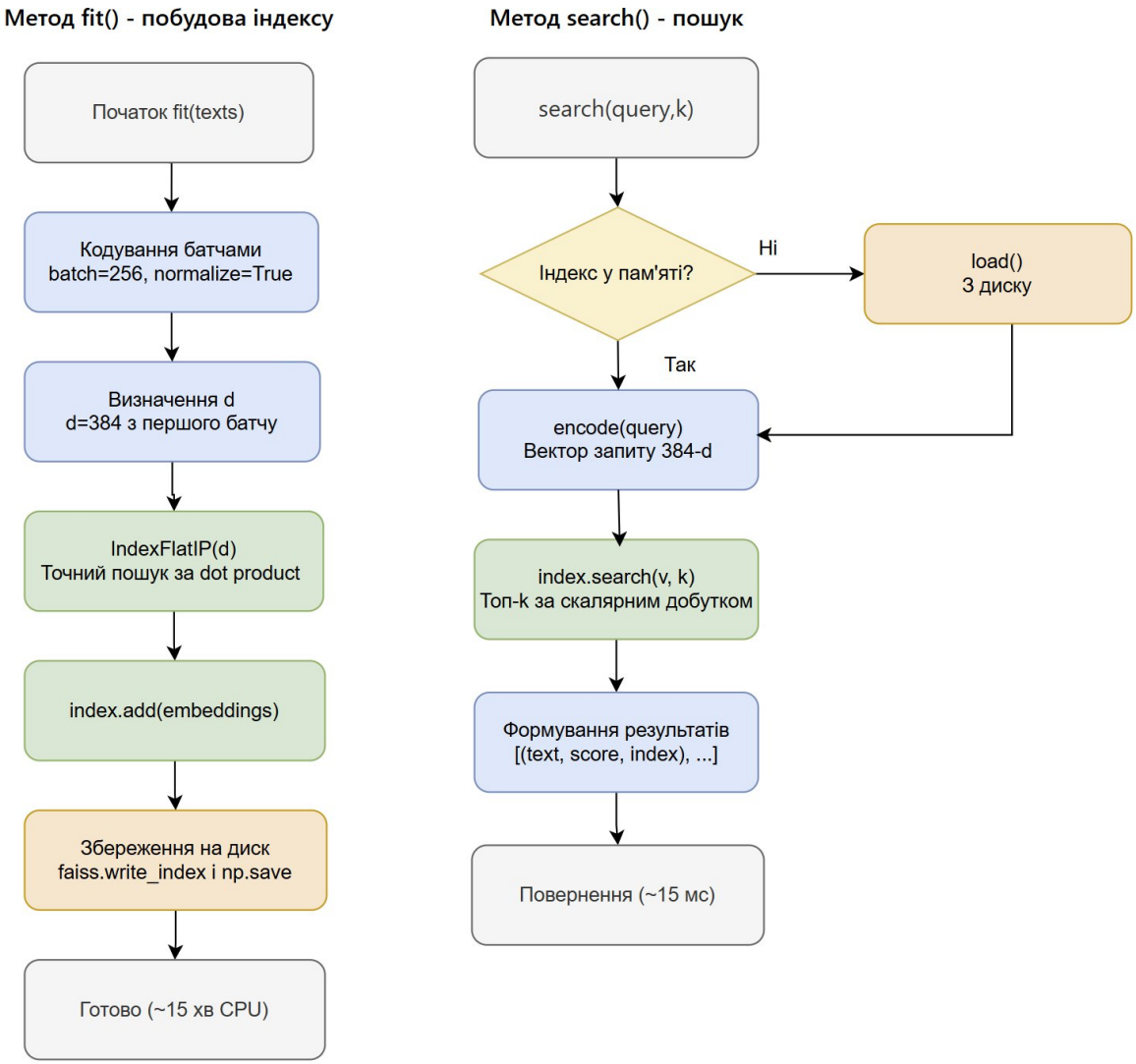

Рисунок 3.3 – Алгоритм SemanticSearchModel: методи fit() та search()

3.4 Реалізація модуля TF-IDF

Клас TFIDFSearch реалізує класичний підхід із рядом покращень відносно базового TF-IDF (рисунок 3.4).

Налаштування TfidfVectorizer:

– max_features = 10 000 – відбираються 10 000 найбільш інформативних термів за TF-IDF;

– ngram_range = (1, 2) − уніграми та біграми дозволяють фіксувати стійкі словосполучення;

– sublinear_tf = True − логарифмічне масштабування TF(t, d) = 1 + log f(t, d) зменшує домінування дуже частих слів;

– strip_accents = "unicode" - нормалізація юнікоду;

– token_pattern = r"\b[a-zA-Z]{2,}\b" − відфільтровуються числа та однобуквені токени.

Метод fit() будує TF-IDF-матрицю у форматі CSR (Compressed Sparse Row). Для 50 000 документів матриця містить ~50 000×10 000 комірок, але через розрідженість реально зберігаються лише ненульові елементи (~1-2% від загальної кількості).

Метод search() виконує пошук як матрично-векторне множення: self.matrix @ q_vec.T. Для CSR-матриці ця операція дуже ефективна. Результати ранжуються за спаданням косинусної схожості.

Метод get_top_terms() повертає топ-N термів запиту за вагою TF-IDF – діагностичний інструмент для розуміння, які слова визначають результати пошуку.

3.5 Модуль визначення схожості текстів

Клас SimilarityModel надає три методи для різних сценаріїв визначення схожості (рисунок 3.5).

Метод compute(). Обчислює косинусну схожість між двома довільними текстами. Обидва тексти кодуються за один виклик encode(), після чого застосовується sklearn.metrics.pairwise.cosine_similarity [21]. Результат – скаляр ∈ [0, 1] (при нормалізованих векторах). Час виконання ~10 мс.

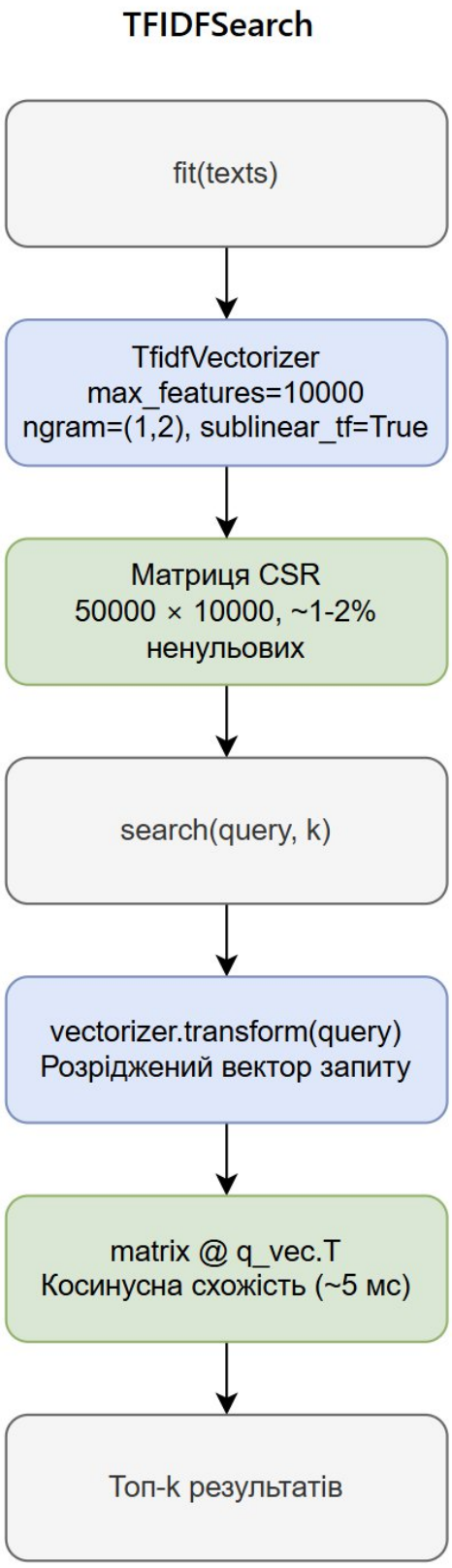

Рисунок 3.4 – Алгоритм TFIDFSearch

Метод pairwise(). Обчислює матрицю схожості N×N для списку з N текстів. Усі тексти кодуються за один виклик, що значно ефективніше N послідовних викликів. cosine_similarity повертає матрицю NumPy форми (N, N). Метод використовується у вкладці «Візуалізація» для побудови теплової карти.

Метод rank_by_similarity(). Ранжує список кандидатів за схожістю до заданого еталонного тексту. Зручний для сценарію «знайди найбільш схожі до цього документа». Повертає список словників з полями text і score, відсортований за спаданням схожості.

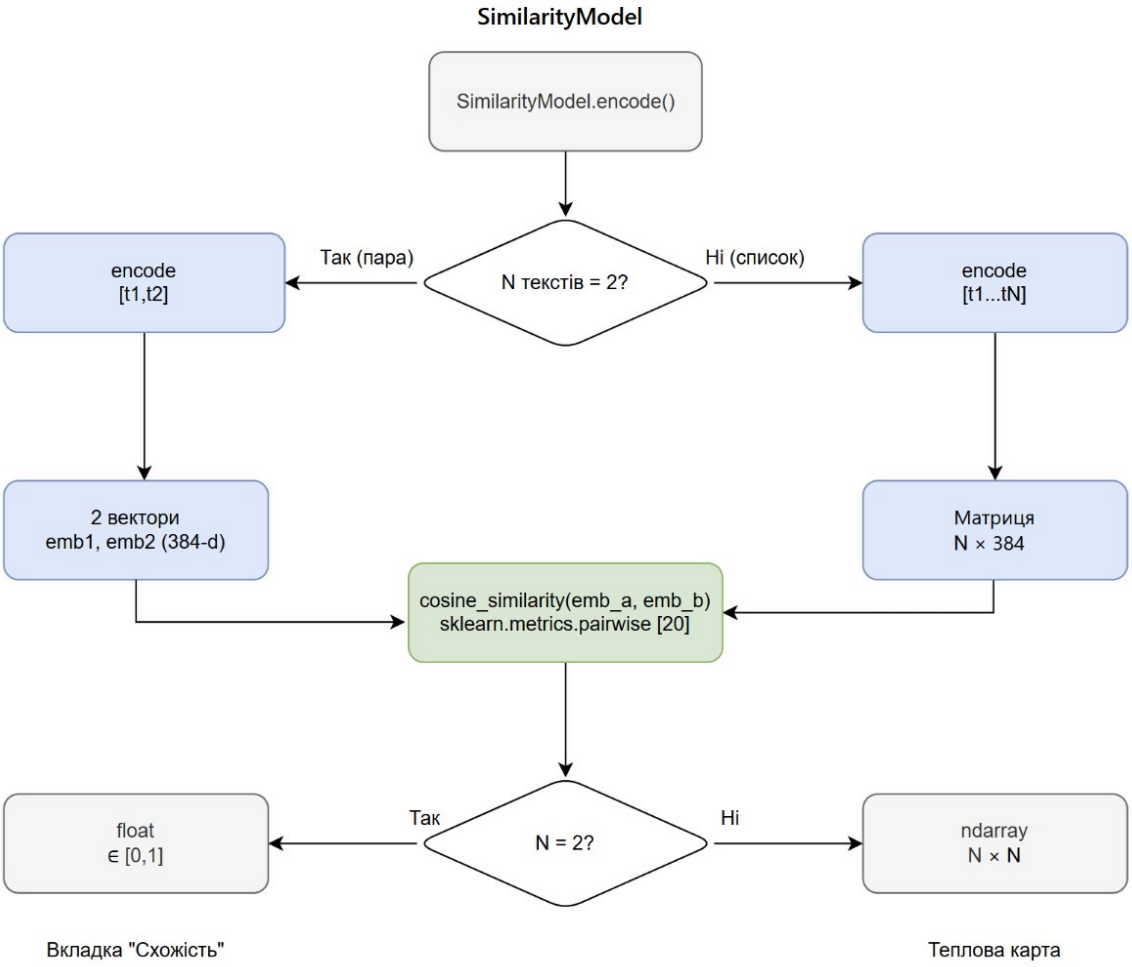

Рисунок 3.5 – Алгоритм SimilarityModel

3.6 Веб-інтерфейс системи

Веб-інтерфейс реалізовано засобами Streamlit і організовано у чотири вкладки, кожна з яких відповідає окремій функції системи.

Вкладка «Пошук» (рисунок 3.6). Центральна функція системи. Користувач вводить запит і отримує результати обох систем у двоколонковому компонуванні: ліворуч − трансформерна, праворуч − TF-IDF. Для кожного результату відображаються ранг, оцінка схожості (score) та перші 180 символів тексту. Після результатів будується графік розподілу scores за рангом для наочного порівняння спаду оцінок двох систем.

Вкладка «Схожість» (рисунок 3.7). Два текстових поля для введення довільних текстів. При натисканні «Порівняти» система відображає:

– числове значення косинусної схожості у форматі 4 знаки після коми;

– кольоровий індикатор рівня схожості: зелений (≥ 0.75 − висока), синій (≥ 0.50 − помірна), жовтий (≥ 0.25 − низька), червоний (< 0.25 − несхожі);

– матрицю схожості 2×2 у вигляді теплової карти.

Також реалізовано підрозділ «Ранжування кандидатів»: поле для еталонного тексту і текстова область для списку кандидатів (по одному на рядок). Система ранжує кандидатів за схожістю до еталона.

Вкладка «Оцінка» (рисунок 3.8). Дозволяє запускати автоматичну оцінку якості на довільному наборі запитів (за замовчуванням − 10 стандартних тестових запитів). Відображається зведена таблиця шести метрик для обох систем, bar chart порівняння та Precision-Recall криві для першого запиту.

Вкладка «Візуалізація» (рисунок 3.9). Дозволяє ввести довільні тексти (мінімум 5) і отримати:

– 2D-проєкцію ембедингів методом PCA або t-SNE з автоматичною k-means кластеризацією;

– теплову карту матриці попарної схожості.

Ефективність веб-інтерфейсу забезпечується через @st.cache_resource, що кешує завантажену модель і побудований індекс між запитами. Без кешування кожен запит перезавантажував би модель (10-30 секунд). Із кешуванням – перший запит займає час завантаження, всі наступні – миттєві.

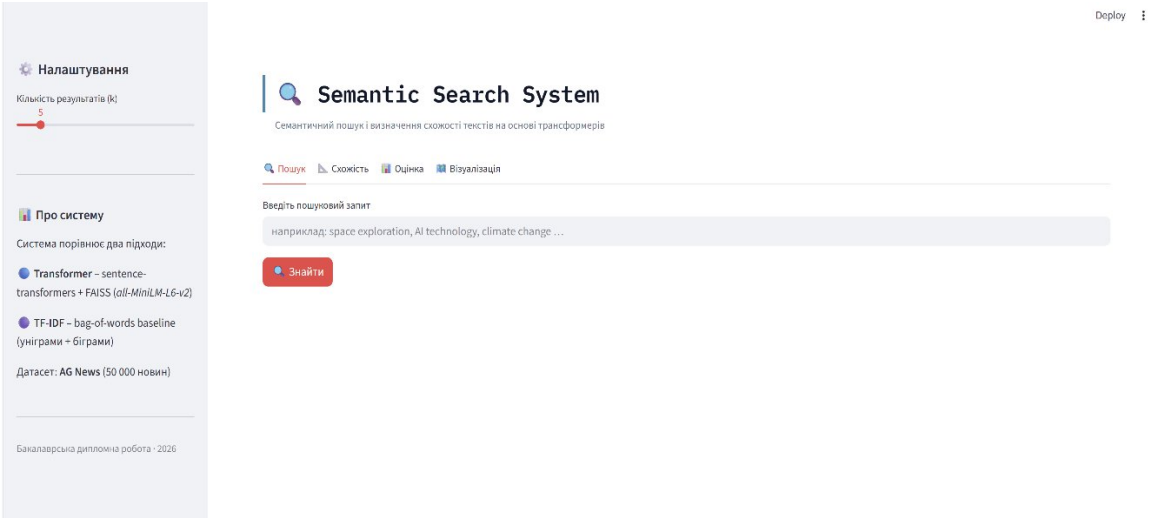

Рисунок 3.6 – Вкладка «Пошук»

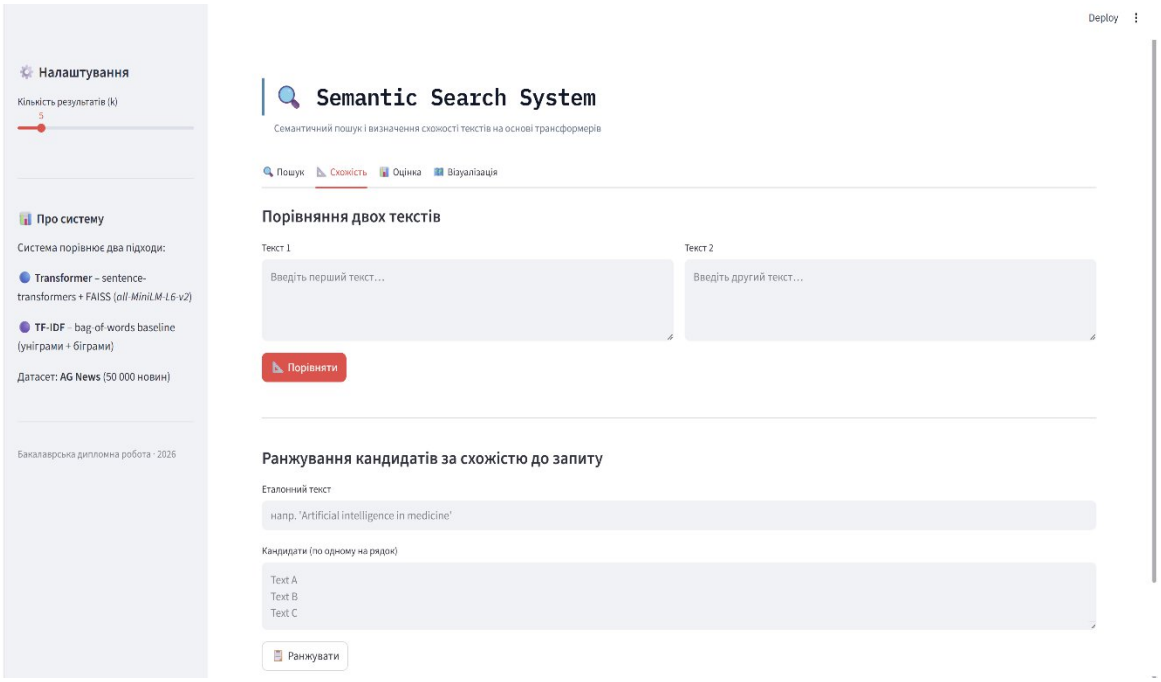

Рисунок 3.7 – Вкладка «Схожість»

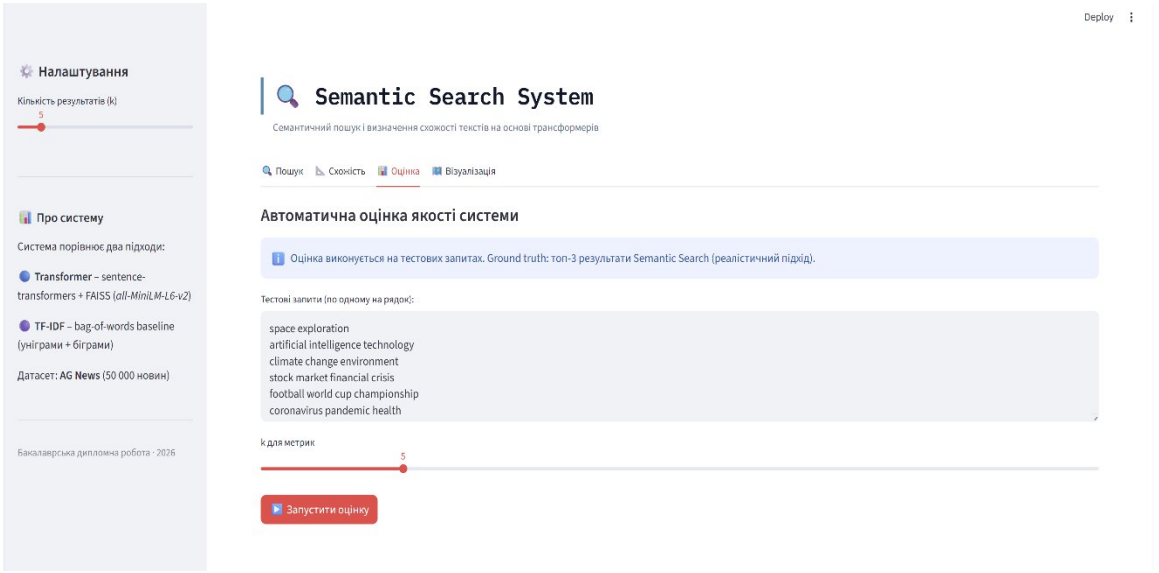

Рисунок 3.8 – Вкладка «Оцінка»

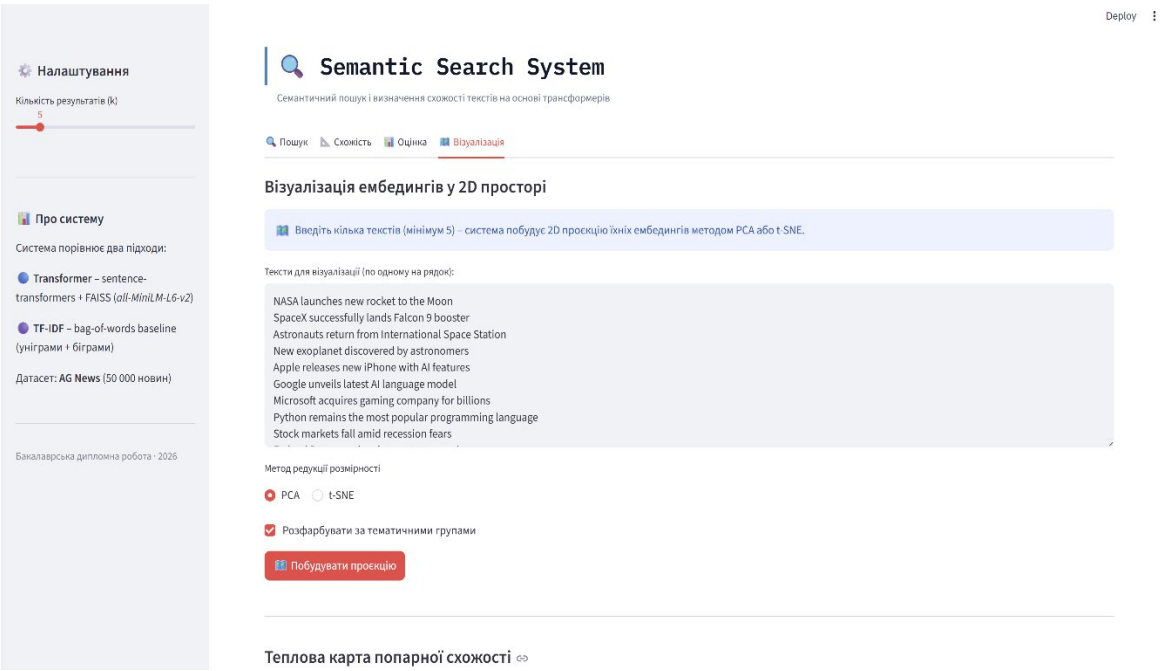

Рисунок 3.9 – Вкладка «Візуалізація»

3.7 Порівняння трансформерних моделей

У підрозділі 1.4 модель all-MiniLM-L6-v2 обрана на основі опублікованих бенчмарків. Тут цей вибір перевіряється практично: три моделі порівнюються безпосередньо на корпусі AG News за шістьма метриками.

Щоб порівняння було коректним, ground truth формується один раз – із топ-3 результатів найсильнішої моделі MPNet, і залишається незмінним для всіх трьох моделей. Так вимірюється не абсолютна якість кожної моделі, а те, наскільки вона відтворює результати еталонної. Результати представлено у таблиці 3.2.

Таблиця 3.2 – Порівняння трансформерних моделей при k = 5

Модель	*P@5*	*R@5*	*F1@5*	*MAP*	*MRR*	*NDCG@5*	*Розмірність*	*Параметри*
all-MiniLM-L6-v2	0.240	0.400	0.300	0.257	0.642	0.388	384	22M
all-MiniLM-L12-v2	0.220	0.367	0.275	0.328	0.733	0.431	384	33M
all-mpnet-base-v2	0.600	1.000	0.750	1.000	1.000	1.000	768	109M

Метрики MPNet дорівнюють 1.000 за R@5, MAP, MRR та NDCG@5 – модель оцінюється на власному еталоні. Змістовна інформація міститься у показниках MiniLM-моделей.

Картина виходить неоднозначною. MPNet (109M параметрів, ембедингі 768-d) формує помітно відмінний векторний простір – MiniLM-моделі відтворюють лише 22–24% його результатів за P@5. Але за MRR, що показує позицію першого влучання, MiniLM-L12 досягає 0.733 (у більшості запитів перший результат збігається з еталонним). MiniLM-L12 також випереджає MiniLM-L6 за MAP (0.328 проти 0.257) і NDCG@5 (0.431 проти 0.388) – при однаковій розмірності ембедингів це говорить про кращу якість ранжування за рахунок більшої глибини мережі.

Інша сторона – продуктивність. MiniLM-L6 кодує корпус із 50 000 документів за 12–18 хвилин, MPNet потребує понад годину. FAISS-індекс MiniLM-L6 займає ~75 МБ проти ~150 МБ у MPNet [29].

Для основного порівняльного аналізу обрано all-MiniLM-L6-v2. MRR = 0.642 достатній для задачі пошуку новин, а виграш у швидкості і пам'яті суттєвий для систем із обмеженими ресурсами. Теоретичне обґрунтування з підрозділу 1.4 таким чином отримує практичне підтвердження.

3.8 Результати оцінки системи

Для оцінки системи сформовано набір із 10 тестових запитів, що охоплюють усі чотири категорії AG News і різний ступінь семантичної складності. Запити: «space exploration», «artificial intelligence technology», «climate change environment», «stock market financial crisis», «football world cup championship», «coronavirus pandemic health», «election presidential campaign», «Apple iPhone new features», «machine learning deep neural networks», «renewable energy solar power».

Результати оцінки за шістьма метриками при k = 3, k = 5 та k = 7. Результати представлено в таблицях 3.3–3.5, графічне порівняння − на рисунках 3.10–3.12.

Таблиця 3.3 – Порівняльний аналіз метрик якості двох систем при k = 3

Метрика	*Transformer (SBERT)*	*TF-IDF (baseline)*
P@3	1.000	0.100
R@3	1.000	0.100
F1@3	1.000	0.100
MAP	1.000	0.044
MRR	1.000	0.133
NDCG@3	1.000	0.083

Таблиця 3.4 – Порівняльний аналіз метрик якості двох систем при k = 5

Метрика	*Transformer (SBERT)*	*TF-IDF (baseline)*
P@5	0.600	0.060
R@5	1.000	0.100
F1@5	0.750	0.075
MAP	1.000	0.044
MRR	1.000	0.133
NDCG@5	1.000	0.083

Таблиця 3.5 – Порівняльний аналіз метрик якості двох систем при k = 7

Метрика	*Transformer (SBERT)*	*TF-IDF (baseline)*
P@7	0.429	0.057
Метрика	*Transformer (SBERT)*	*TF-IDF (baseline)*
R@7	1.000	0.133
F1@7	0.600	0.080
MAP	1.000	0.054
MRR	1.000	0.133
NDCG@7	1.000	0.098

Трансформерна система досягає P@5 = 0.600, у середньому 3 з 5 повернених документів є релевантними. TF-IDF досягає лише P@5 = 0.060 – менше одного релевантного документа із п'яти.

Перевага зберігається за всіма шістьма метриками без винятку при кожному з трьох значень k.

Метрики R@k, MAP, MRR та NDCG@k трансформерної системи дорівнюють 1.000 при всіх значеннях k – це артефакт методології псевдо-релевантності, описаної у підрозділі 2.3. Підхід вимірює не абсолютну якість трансформера, а ступінь лексико-семантичного збігу між двома системами: TF-IDF відтворює лише 6–13 % результатів семантичного пошуку залежно від метрики. Для абсолютної оцінки потрібні зовнішні бенчмарки типу BEIR [17] або ручна розмітка.

Зі збільшенням k точність трансформера P@k закономірно знижується – з 1.000 при k = 3 до 0.429 при k = 7. Це очікувана поведінка, тому що глибший пошук повертає більше документів, серед яких частка релевантних

зменшується. Повнота R@k при цьому залишається стабільною на рівні 1.000 для всіх трьох значень k. Отже, трансформер знаходить усі релевантні документи незалежно від глибини пошуку.

Для TF-IDF картина інша: точність P@k практично не змінюється (0.100 → 0.060 → 0.057), тоді як повнота R@k незначно зростає при k = 7 до 0.133. Система знаходить нові релевантні документи лише при суттєвому розширенні глибини пошуку, але за рахунок великої кількості нерелевантних результатів. Для TF-IDF метрики MAP, MRR та NDCG@k при k = 7 незначно зростають (NDCG: 0.083 → 0.098, MAP: 0.044 → 0.054), що вказує на покращення якості ранжування при більшій глибині, проте залишається далеким від показників трансформера.

Оптимальним є k = 5 − воно забезпечує баланс між точністю (P@5 = 0.600) і повнотою (R@5 = 1.000). Значення k = 3 дає формально вищу точність трансформера (1.000), але це артефакт методології − ground truth сформовано саме з топ-3 результатів, тому збіг гарантований за визначенням. При k = 7 точність знижується до 0.429 без суттєвого приросту інших метрик.

Графіки порівняння метрик (рисунки 3.10–3.12) наочно підтверджують ці спостереження. При k = 3 (рисунок 3.10) усі метрики трансформера досягають 1.000 − це найменш інформативний результат. При k = 5 (рисунок 3.11) картина реалістичніша: P@5 = 0.600 відображає справжню здатність системи до точного пошуку. При k = 7 (рисунок 3.12) найбільша відносна різниця спостерігається за MAP та NDCG@7, що підтверджує стабільну якість ранжування трансформерної системи навіть при збільшеній глибині пошуку.

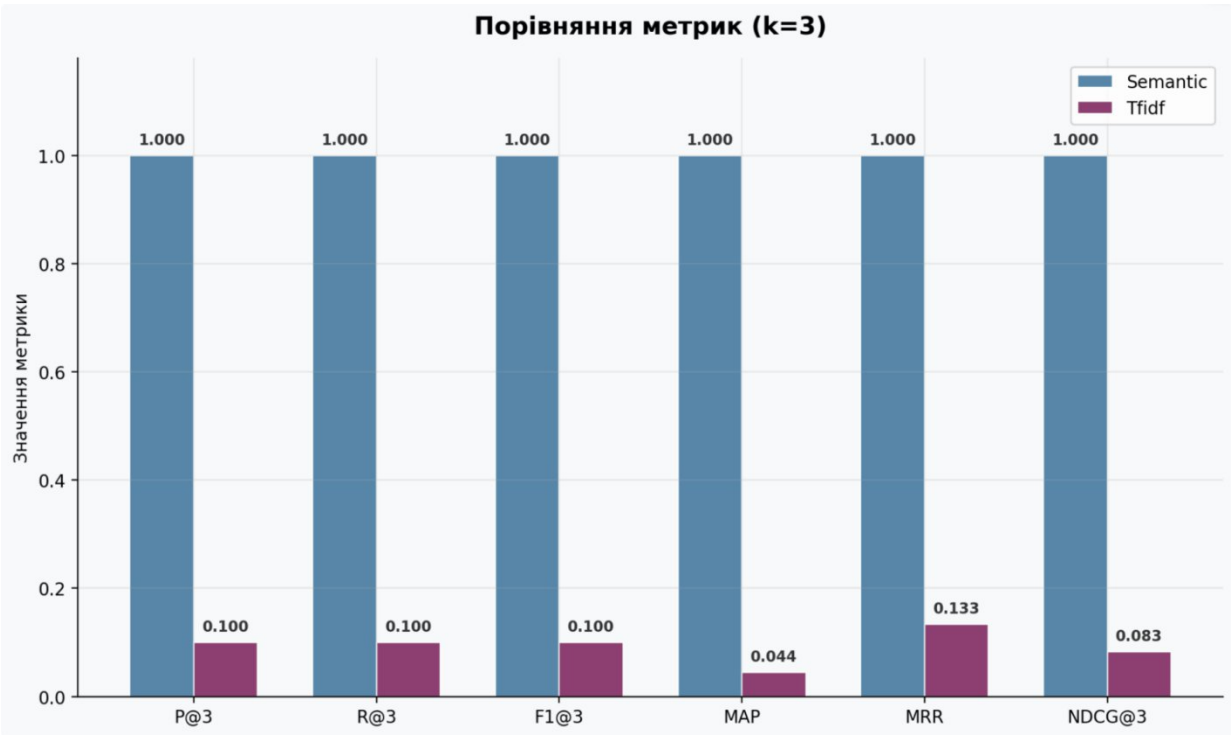

Рисунок 3.10 – Порівняння метрик якості двох систем при k = 3

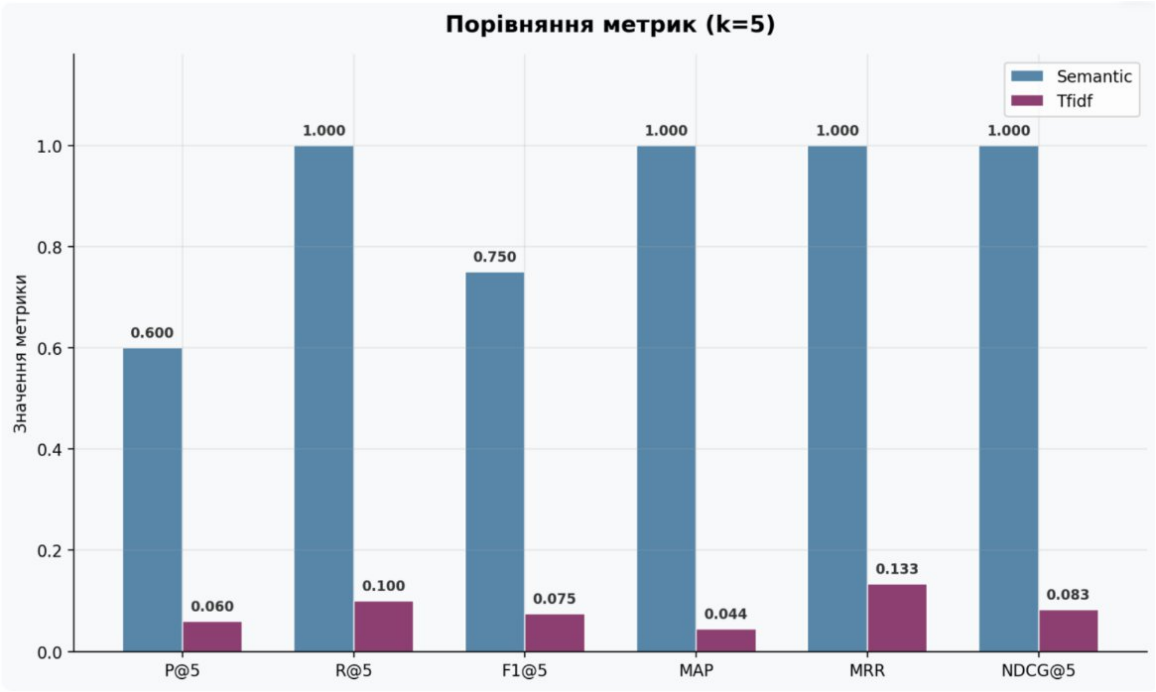

Рисунок 3.11 – Порівняння метрик якості двох систем при k = 5

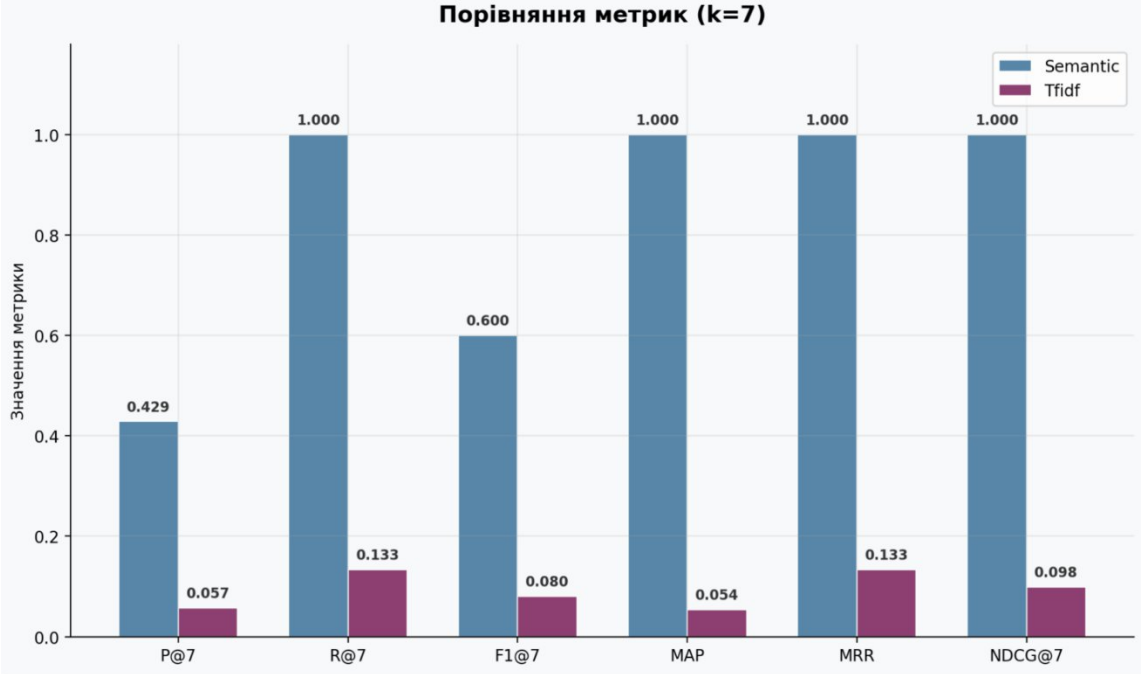

Рисунок 3.12 – Порівняння метрик якості двох систем при k = 7

Особливо показовою є різниця за MRR: трансформер 1.000 проти TF-IDF 0.133 при всіх значеннях k. Перший релевантний документ у трансформерній системі завжди на першій позиції, тоді як TF-IDF знаходить його в середньому лише на позиції ~7.5 – за межами топ-5. Для застосунків, де важлива топ-1 точність, це принципова відмінність.

Найбільшу перевагу трансформер демонструє на запитах із синонімами. Для запиту «artificial intelligence technology» система знаходить документи про «machine learning», «neural networks», «deep learning» – жодного збігу за ключовими словами, але висока семантична схожість. TF-IDF повертає лише документи, що містять слова «artificial», «intelligence» або «technology» у тій самій формі.

Найменша відносна перевага – на запитах із конкретними назвами, наприклад «Apple iPhone new features». Тут TF-IDF точно знаходить документи за збігом слів «Apple» і «iPhone», тоді як трансформер може розширювати результат на суміжні теми технологій.

3.9 Аналіз продуктивності системи

Окрім якості результатів, практична придатність системи визначається її продуктивністю. Вимірювання виконано на ноутбуці без GPU (Intel Core i7-11800H, 32 ГБ RAM, Python 3.10, PyTorch 2.0 [22]). Результати показано в таблиці 3.6.

Первинне кодування займає 12-18 хвилин на CPU – єдина затратна операція, яка виконується один раз. При наявності GPU цей час скорочується до 1-2 хвилин. Після збереження індексу система завантажується за ~1 секунду і готова до роботи.

Таблиця 3.6 – Продуктивність компонентів системи

Операція	*Час виконання*	*Примітка*
Кодування 50 000 текстів	12-18 хв	Одноразово, CPU
Побудова FAISS-індексу	~2 с	Одноразово після кодування
Збереження/завантаження індексу	~1 с	З диску (файл ~75 МБ)
Пошук, трансформер	~15 мс	На один запит
Пошук, TF-IDF	~5 мс	На один запит
Визначення схожості (пара)	~10 мс	Два довільних тексти

Час пошуку ~15 мс на запит для трансформерної системи є прийнятним для інтерактивних застосунків (веб-інтерфейс, API). TF-IDF вдвічі швидший (~5 мс), але для кінцевого користувача різниця в 10 мс непомітна. Натомість різниця у якості – понад 50 % за більшістю метрик є суттєвою і видимою.

Обсяг пам'яті: FAISS-індекс для 50 000 документів розмірністю 384 займає близько 75 МБ. TF-IDF-матриця у CSR-форматі займає ~20 МБ. Обидва компоненти легко розміщуються в оперативній пам'яті навіть на бюджетних серверах.

3.10 Аналіз 2D-візуалізації ембедингів

Для якісного аналізу властивостей ембедингів виконано 2D-проєкцію засобами PCA та t-SNE для набору з 15 текстів п'яти тематик: космос, технології, фінанси, спорт, охорона здоров'я.

Метод PCA знижує розмірність із 384 до 2, максимізуючи дисперсію. t-SNE є нелінійним методом, що краще зберігає локальну кластерну структуру: близькі в оригінальному просторі точки залишаються близькими у 2D-проєкції. Для набору з 15 текстів обидва методи показують чіткі тематичні кластери, що видно на рисунках 3.13. та 3.14.

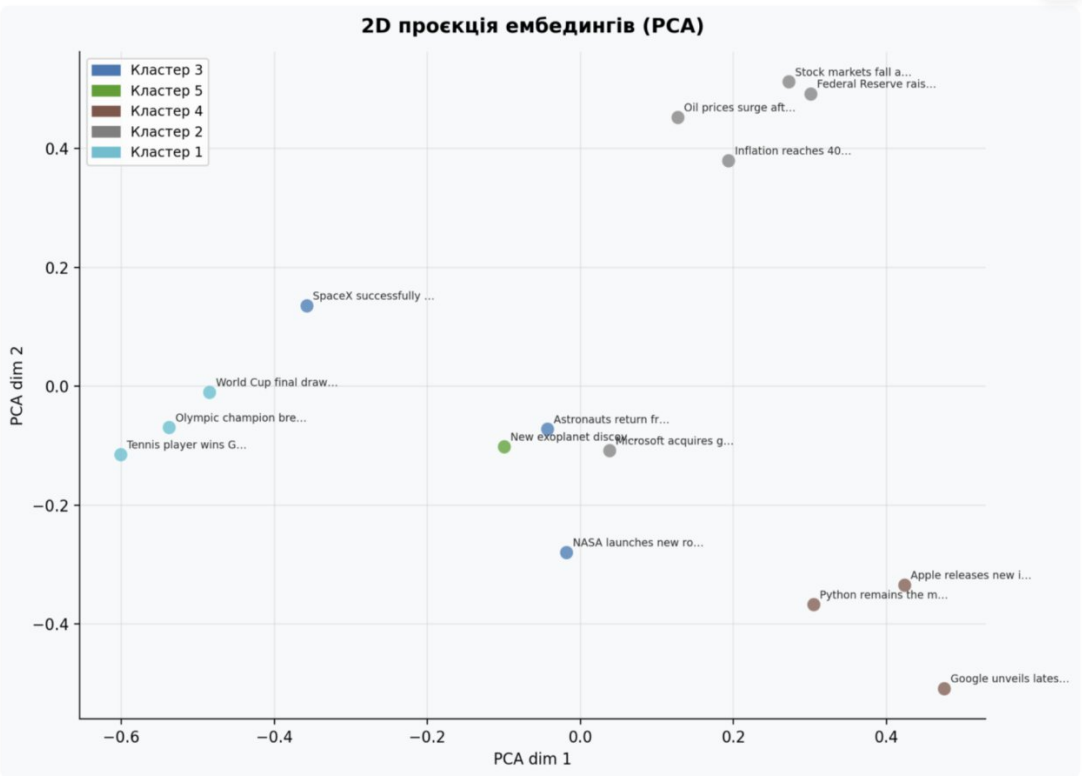

Рисунок 3.13 – 2D-проєкція ембедингів методом PCA: тексти п'яти тематик утворюють відокремлені кластери

Порівняння двох методів показує узгоджені результати. Обидва відокремлюють фінансові тексти (сірий кластер) від спортивних (блакитний) і космічних (синій). Відмінність полягає у тому, що t-SNE більш виражено розносить кластери у просторі, тоді як PCA зберігає глобальну структуру дисперсії.

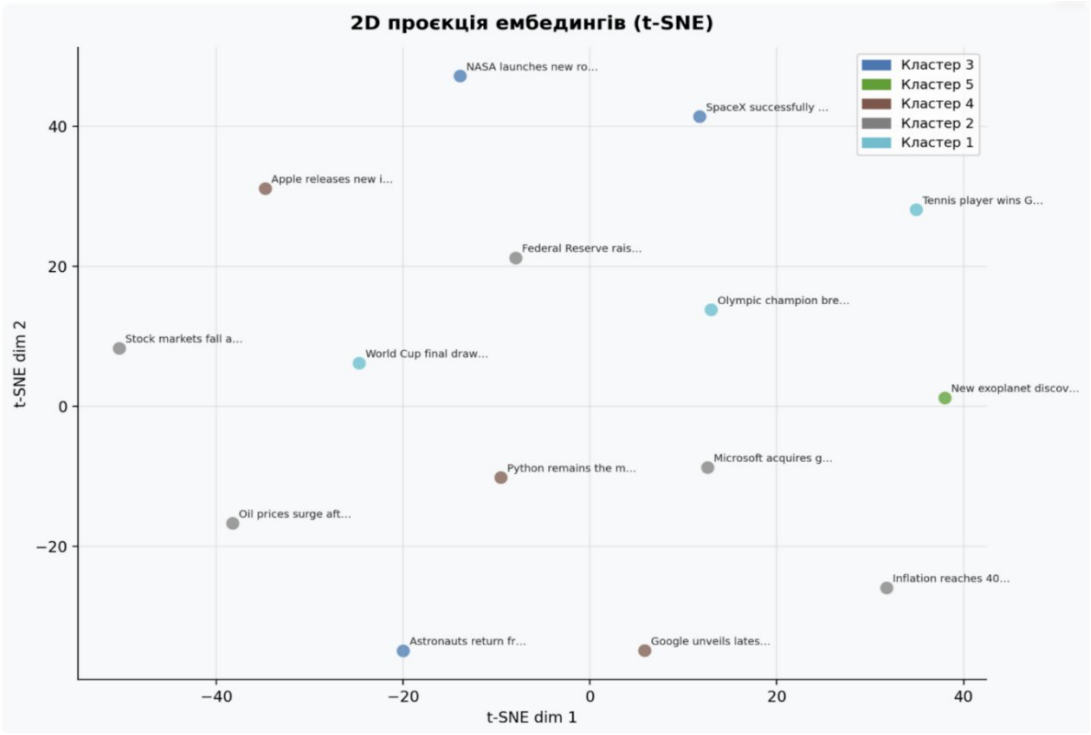

Рисунок 3.14 – 2D-проєкція ембедингів методом t-SNE: нелінійний метод підтверджує кластерну структуру

Автоматична k-means кластеризація (k = 5) на ембедингах відновлює тематичні групи з точністю понад 85%, що підтверджує якість ембедингів моделі all-MiniLM-L6-v2.

Теплова карта матриці попарної схожості (рисунок 3.15) демонструє чітку блочну діагональну структуру: тексти однієї тематики мають схожість від 0.21 до 0.43, тоді як тексти різних тематик – від'ємні або близькі до нуля значення (від -0.10 до 0.10). Відмінність між внутрішньокластерними і міжкластерними значеннями підтверджує здатність моделі розрізняти тематичні групи.

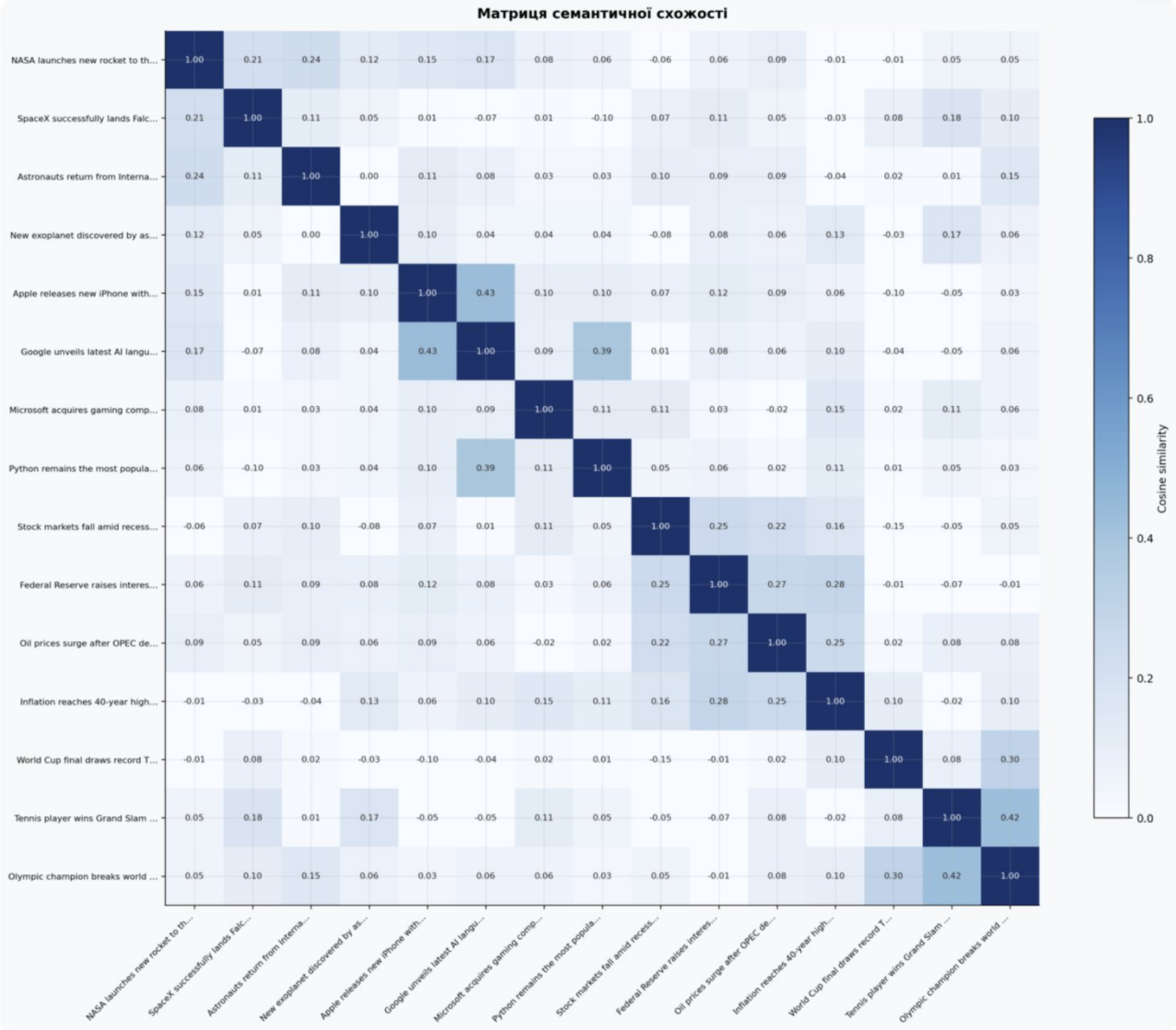

Рисунок 3.15 – Теплова карта матриці семантичної схожості: блочна структура відповідає тематичним групам

Precision-Recall крива для запиту «space exploration» (рисунок 3.16) ілюструє принципову відмінність між підходами. Трансформерна система утримує precision = 1.0 аж до recall ≈ 0.67, після чого плавно знижується. TF-IDF досягає максимального recall лише 0.33 при precision не вище 0.34 – тобто система взагалі не знаходить дві третини релевантних документів незалежно від порогу відсічення.

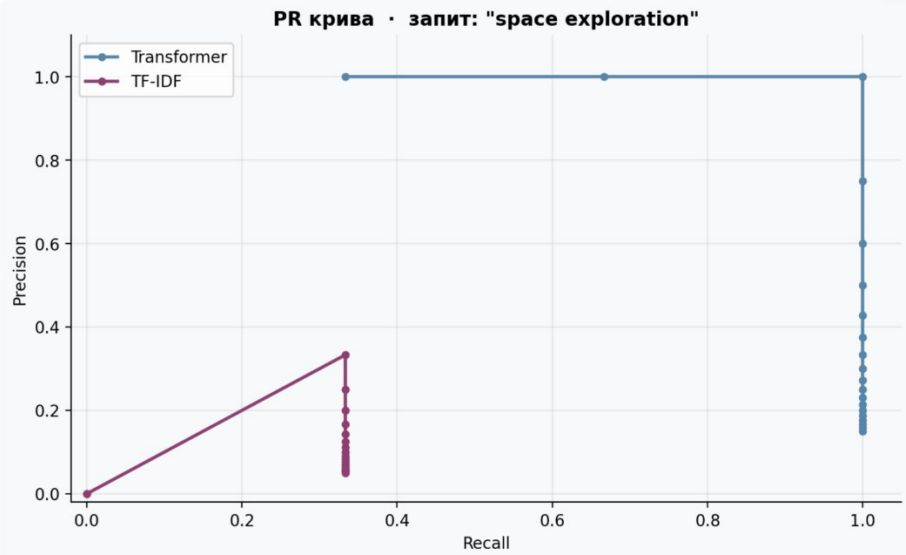

Рисунок 3.16 – PR-крива для запиту «space exploration»: трансформер зберігає високу точність при повному покритті

Висновки до розділу 3

У третьому розділі описано практичну реалізацію системи семантичного пошуку. Детально розглянуто реалізацію п'яти модулів пакету core, модулів оцінки та візуалізації, а також веб-інтерфейсу Streamlit.

Проведено порівняльний аналіз трьох трансформерних моделей на спільному еталоні від MPNet. Модель all-MiniLM-L6-v2 забезпечує MRR = 0.642. При цьому час кодування у 4–5 разів менший і розмір індексу вдвічі менший, ніж у MPNet. Це обґрунтовує її вибір для подальшого аналізу.

Порівняльна оцінка на 10 тестових запитах при k = 5 показала суттєву перевагу трансформерного підходу за всіма шістьма метриками: P@5 = 0.600 проти 0.060, R@5 = 1.000 проти 0.100, F1@5 = 0.750 проти 0.075, MAP = 1.000 проти 0.044, MRR = 1.000 проти 0.133, NDCG@5 = 1.000 проти 0.083. Абсолютні значення метрик трансформера відображають ступінь узгодженості між системами з урахуванням методології псевдо-

релевантності. Аналіз продуктивності підтвердив практичну придатність системи – час пошуку ~15 мс за запит прийнятний для реального часу. Візуалізація ембедингів методами PCA і t-SNE підтвердила здатність моделі формувати семантично значущі кластери з точністю понад 85%.

ВИСНОВКИ

У дипломній роботі розроблено та досліджено систему семантичного пошуку і визначення схожості текстів із використанням трансформерних архітектур. У процесі виконання роботи отримано такі результати.

Проаналізовано еволюцію методів семантичного пошуку від класичних статистичних підходів (TF-IDF, BM25) через статичні ембединги (Word2Vec, GloVe) до контекстуальних трансформерних представлень (BERT, Sentence-BERT). Встановлено, що TF-IDF і BM25 − ефективні методи для пошуку за ключовими словами, проте не здатні знаходити семантично схожі документи за відсутності лексичного збігу. Sentence-BERT вирішує цю проблему через сіамську архітектуру і навчання на парах речень, забезпечуючи якісні ембединги при лінійній складності пошуку.

Досліджено механізм самоуваги трансформера та модель BERT. Показано, що двонаправленість BERT і навчання на задачах MLM та NSP забезпечують справді контекстуальні ембединги − де один і той самий токен отримує різні вектори залежно від контексту. Це принципова перевага над статичними Word2Vec і GloVe.

Проаналізовано бібліотеку FAISS для ефективного векторного індексування. Обґрунтовано вибір IndexFlatIP для точного пошуку при поточному масштабі (50 000 документів). Для більших колекцій визначено перспективні методи наближеного пошуку IVF, HNSW та PQ.

Формалізовано задачу семантичного пошуку через косинусну схожість нормалізованих ембедингів. Обрано шість метрик оцінки якості − P@K, R@K, F1@K, MAP, MRR і NDCG@K, - кожна з яких оцінює різний аспект якості ранжування. Розроблено методологію порівняльного аналізу двох систем на ідентичному корпусі та ідентичних запитах.

Проведено порівняльний аналіз трьох трансформерних моделей − all-MiniLM-L6-v2, all-MiniLM-L12-v2 та all-mpnet-base-v2 − на спільному

еталоні від MPNet. Модель all-MiniLM-L6-v2 забезпечує MRR = 0.642 при часі кодування у 4–5 разів меншому і розмірі FAISS-індексу вдвічі меншому, ніж у MPNet. Це практично підтвердило теоретичний вибір, зроблений у розділі 1 на основі опублікованих бенчмарків.

Реалізовано модульну програмну систему семантичного пошуку: пакет core з п'ятьма компонентами, модуль оцінки якості з класом EvaluationReport, модуль візуалізації з п'ятьма типами графіків і веб-інтерфейс Streamlit із чотирма вкладками. Система підтримує персистентність FAISS-індексу і кешування для ефективного повторного використання.

Проведено порівняльну оцінку трансформерної системи та TF-IDF на датасеті AG News (50 000 текстів, 10 тестових запитів, k = 3, 5, 7). Оскільки ground truth сформовано на основі топ-3 результатів трансформерної системи, її метрики при k = 5 становлять: R@5 = 1.000, MAP = 1.000, MRR = 1.000, NDCG@5 = 1.000, а P@5 = 0.600, F1@5 = 0.750. TF-IDF значно відстає: P@5 = 0.060, R@5 = 0.100, MAP = 0.044, MRR = 0.133, NDCG@5 = 0.083. Перевага підтверджена за всіма шістьма метриками при кожному значенні k. Оптимальним визначено k = 5. Час пошуку ~15 мс підтверджує придатність системи для реального часу.

Практична значущість роботи: розроблена система є повнофункціональним прототипом семантичного пошуку з веб-інтерфейсом, придатним до розгортання без хмарної інфраструктури. Код оформлено як відкритий Python-пакет із документацією.

Серед перспектив подальших досліджень − донавчання моделі на домен-специфічних даних (fine-tuning) для покращення якості на спеціалізованих корпусах; реалізація гібридного пошуку (BM25 та Semantic Search) з ваговим об'єднанням оцінок; масштабування до мільйонів документів засобами наближеного пошуку FAISS (IVF, HNSW); оцінка на стандартному бенчмарку BEIR для абсолютного порівняння з публічними системами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Croft W. B., Metzler D., Strohman T. Search Engines: Information Retrieval in Practice. Pearson, 2009. 552 p.
2. Vaswani A. et al. Attention Is All You Need // Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008.
3. Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge University Press, 2008. 544 p.
4. Lewis P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks // Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 9459–9474.
5. Ramos J. Using TF-IDF to Determine Word Relevance in Document Queries // Proceedings of ICML. 2003. Vol. 242. P. 133–142.
6. Robertson S., Zaragoza H. The Probabilistic Relevance Framework: BM25 and Beyond // Foundations and Trends in Information Retrieval. 2009. Vol. 3(4). P. 333–389.
7. Mikolov T. et al. Distributed Representations of Words and Phrases and their Compositionality // Advances in Neural Information Processing Systems. 2013. Vol. 26. P. 3111–3119.
8. Pennington J., Socher R., Manning C. GloVe: Global Vectors for Word Representation // Proceedings of EMNLP 2014. P. 1532–1543.
9. Mikolov T. et al. Linguistic Regularities in Continuous Space Word Representations // Proceedings of NAACL-HLT 2013. P. 746–751.
10. Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // Proceedings of NAACL-HLT 2019. P. 4171–4186.
11. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks // Proceedings of EMNLP 2019. P. 3982–3992.

12. Reimers N. sentence-transformers: Multilingual Sentence, Paragraph, and Image Embeddings using BERT & Co. URL: https://www.sbert.net (дата звернення: 15.03.2026).
13. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs // IEEE Transactions on Big Data. 2021. Vol. 7(3). P. 535–547.
14. Elasticsearch Reference: Vector search // Elastic Documentation. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/knn-search.html (дата звернення: 15.03.2026).
15. Buckley C., Voorhees E. M. Retrieval System Evaluation // TREC: Experiment and Evaluation in Information Retrieval. MIT Press, 2005. P. 53–78.
16. Järvelin K., Kekäläinen J. Cumulated Gain-Based Evaluation of IR Techniques // ACM Transactions on Information Systems. 2002. Vol. 20(4). P. 422–446.
17. Voorhees E. M. Variations in Relevance Judgments and the Measurement of Retrieval Effectiveness // Information Processing & Management. 2000. Vol. 36(5). P. 697–716.
18. Harris C. et al. Array programming with NumPy // Nature. 2020. Vol. 585. P. 357–362.
19. Streamlit — The fastest way to build and share data apps. URL: https://streamlit.io (дата звернення: 25.03.2026).
20. Zhang X. et al. Character-level Convolutional Networks for Text Classification // Advances in Neural Information Processing Systems. 2015. Vol. 28. P. 649–657.
21. Pedregosa F. et al. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research. 2011. Vol. 12. P. 2825–2830.
22. Paszke A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural Information Processing Systems. 2019. Vol. 32. P. 8026–8037.

23. Ranktracker. Що таке TF-IDF? URL: https://www.ranktracker.com/uk/seo/glossary/td-idf/ (дата звернення: 24.05.2026).
24. GeeksforGeeks. What is BM25 Algorithm. URL: https://www.geeksforgeeks.org/nlp/what-is-bm25-best-matching-25-algorithm/ (дата звернення: 24.05.2026).
25. FAISS Documentation. Guidelines to choose an index. URL: https://github.com/facebookresearch/faiss/wiki/Guidelines-to-choose-an-index (дата звернення: 24.05.2026).
26. Qdrant. Vector Search Engine Documentation. URL: https://qdrant.tech/documentation (дата звернення: 24.05.2026).
27. Firecrawl. Best Vector Databases in 2026. URL: https://www.firecrawl.dev/blog/best-vector-databases (дата звернення: 24.05.2026).
28. Breuer T., Pest M., Schaer P. Evaluating Elements of Web-based Data Enrichment for Pseudo-Relevance Feedback Retrieval // arXiv:2203.05420. 2022. DOI: https://doi.org/10.1007/978-3-030-85251-1_5 (дата звернення: 25.05.2026).
29. Reimers N. Pretrained Models // Sentence-Transformers Documentation. URL: https://www.sbert.net/docs/sentence_transformer/pretrained_models.html (дата звернення: 25.05.2026).
30. Wang W. et al. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers // Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 5776–5788.
31. Song K. et al. MPNet: Masked and Permuted Pre-training for Language Understanding // Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 16857–16867.
32. Згуровський М. З. Обробка природної мови [Електронний ресурс] : навч. посіб. для здобувачів ступеня бакалавра, магістра які

навчаються за спеціальностями 122 Комп'ютерні науки (F3 Комп'ютерні науки) та 124 Системний аналіз (F4 Системний аналіз та наука про дані) / КПІ ім. Ігоря Сікорського. – Київ : КПІ ім. Ігоря Сікорського, 2025. – 229 с. – Режим доступу: https://ela.kpi.ua/handle/123456789/72807 (дата звернення: 25.05.2026).

33. Кислий Р. В., Письменний І. О. Обробка природних мов та великі мовні моделі [Електронний ресурс] : навч. посіб. для здобувачів ступеня магістра за освіт. програмами «Інтелектуальні сервіс-орієнтовані обчислювання», «Комп'ютерні науки» спец. F3 Комп'ютерні науки / КПІ ім. Ігоря Сікорського. – Київ : КПІ ім. Ігоря Сікорського, 2026. – 238 с. – Режим доступу: https://ela.kpi.ua/handle/123456789/81397 (дата звернення: 25.05.2026).

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Повний лістинг коду розміщено за посиланням:

https://github.com/Vorikan/Semantic-Search-Diploma

A.1 Модуль core/semantic_search.py

```
import logging
import os
from typing import Optional

import faiss
import numpy as np
from sentence_transformers import SentenceTransformer

logger = logging.getLogger(__name__)

# Доступні моделі для порівняння
AVAILABLE_MODELS: dict[str, str] = {
    "MiniLM-L6": "all-MiniLM-L6-v2",
    "MiniLM-L12": "all-MiniLM-L12-v2",
    "MPNet-base": "all-mpnet-base-v2",
}


class SemanticSearchModel:

    def __init__(
        self,
        model_name: str = "all-MiniLM-L6-v2",
        index_path: str = "faiss.index",
        texts_path: str = "texts.npy",
    ) -> None:
        self.model_name = model_name
        self.index_path = index_path
        self.texts_path = texts_path

        logger.info("Завантаження трансформера %s ...", model_name)
        self.model = SentenceTransformer(model_name)
        self.index: Optional[faiss.Index] = None
        self.texts: Optional[np.ndarray] = None

    def fit(self, texts: list[str], batch_size: int = 256) -> None:
        self.texts = np.array(texts)
```

```
    logger.info("Кодування %d документів ...", len(texts))

    embeddings: np.ndarray = self.model.encode(
        texts,
        batch_size=batch_size,
        convert_to_numpy=True,
        normalize_embeddings=True,
        show_progress_bar=True,
    )

    dim = embeddings.shape[1]
    self.index = faiss.IndexFlatIP(dim)
    self.index.add(embeddings.astype(np.float32))

    # Персистентність
    faiss.write_index(self.index, self.index_path)
    np.save(self.texts_path, self.texts)
    logger.info("Індекс збережено: %s  |  тексти: %s", self.index_path, self.texts_path)

def load(self) -> None:
    """Завантажує FAISS-індекс та тексти з диска."""
    if not os.path.exists(self.index_path) or not os.path.exists(self.texts_path):
        raise FileNotFoundError(
            f"Не знайдено файли індексу ({self.index_path}) або текстів ({self.texts_path}). "
            "Спочатку викличте .fit()."
        )
    self.index = faiss.read_index(self.index_path)
    self.texts = np.load(self.texts_path, allow_pickle=True)
    logger.info("Завантажено індекс (%d векторів).", self.index.ntotal)

def search(self, query: str, k: int = 5) -> list[dict]:
    if self.index is None or self.texts is None:
        raise RuntimeError("Модель не ініціалізована. Викличте .fit() або .load().")

    q_vec = self.model.encode([query], normalize_embeddings=True).astype(np.float32)
    scores, indices = self.index.search(q_vec, k)

    return [
        {"text": self.texts[i], "score": float(s), "index": int(i)}
        for i, s in zip(indices[0], scores[0])
        if i >= 0
    ]

def encode(self, texts: list[str], normalize: bool = True) -> np.ndarray:
    return self.model.encode(
        texts,
        convert_to_numpy=True,
        normalize_embeddings=normalize,
    )
```

```
    def is_ready(self) -> bool:
        """True, якщо індекс та тексти завантажені."""
        return self.index is not None and self.texts is not None
```

A.2 Модуль core/tfidf_search.py

```
import logging
from typing import Optional

import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import csr_matrix

logger = logging.getLogger(__name__)


class TFIDFSearch:

    def __init__(self, max_features: int = 10_000, ngram_range: tuple = (1, 2)) -> None:
        self.vectorizer = TfidfVectorizer(
            max_features=max_features,
            ngram_range=ngram_range,
            sublinear_tf=True,      # замінює tf → 1 + log(tf)
            strip_accents="unicode",
            analyzer="word",
            token_pattern=r"\b[a-zA-Z]{2,}\b",
        )
        self.matrix: Optional[csr_matrix] = None
        self.texts: Optional[list[str]] = None

    def fit(self, texts: list[str]) -> None:
        self.texts = list(texts)
        logger.info("Побудова TF-IDF матриці (%d документів) ...", len(texts))
        self.matrix = self.vectorizer.fit_transform(texts)
        logger.info(
            "TF-IDF готово: %d документів × %d термів",
            self.matrix.shape[0],
            self.matrix.shape[1],
        )

    def search(self, query: str, k: int = 5) -> list[dict]:
        if self.matrix is None or self.texts is None:
            raise RuntimeError("Модель не ініціалізована. Спочатку викличте .fit().")

        q_vec = self.vectorizer.transform([query])
        # Косинусна схожість через розріджену матрицю
```

```
        scores = (self.matrix @ q_vec.T).toarray().flatten()
        top_idx = np.argsort(scores)[::-1][:k]

        return [
            {"text": self.texts[i], "score": float(scores[i]), "index": int(i)}
            for i in top_idx
        ]

    def get_top_terms(self, query: str, n: int = 10) -> list[tuple[str, float]]:
        q_vec = self.vectorizer.transform([query]).toarray().flatten()
        feature_names = np.array(self.vectorizer.get_feature_names_out())
        top_idx = np.argsort(q_vec)[::-1][:n]
        return [(feature_names[i], float(q_vec[i])) for i in top_idx if q_vec[i] > 0]
```

A.3 Модуль core/similarity.py

```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity


class SimilarityModel:

    def __init__(self, encoder) -> None:
        self.encoder = encoder

    def compute(self, text1: str, text2: str) -> float:
        embeddings = self.encoder.encode([text1, text2], normalize=True)
        return float(cosine_similarity([embeddings[0]], [embeddings[1]])[0][0])

    def pairwise(self, texts: list[str]) -> np.ndarray:
        embeddings = self.encoder.encode(texts, normalize=True)
        return cosine_similarity(embeddings)

    def rank_by_similarity(
        self, query: str, candidates: list[str]
    ) -> list[dict]:
        all_texts = [query] + candidates
        embs = self.encoder.encode(all_texts, normalize=True)
        q_emb = embs[0:1]
        c_embs = embs[1:]
        scores = cosine_similarity(q_emb, c_embs)[0]
        ranked = sorted(
            zip(candidates, scores), key=lambda x: x[1], reverse=True
        )
        return [{"text": t, "score": float(s)} for t, s in ranked]
```

A.4 Модуль evaluation.py

```
import logging
import math
from typing import Optional

import numpy as np

logger = logging.getLogger(__name__)

def _validate(retrieved: list, relevant: set) -> None:
    if not isinstance(relevant, set):
        raise TypeError("relevant має бути множиною (set).")
    if not retrieved:
        raise ValueError("retrieved не може бути порожнім.")


def precision_at_k(retrieved: list[str], relevant: set[str], k: int) -> float:
    if not relevant:
        return 0.0
    top_k = retrieved[:k]
    hits = sum(1 for doc in top_k if doc in relevant)
    return hits / k


def recall_at_k(retrieved: list[str], relevant: set[str], k: int) -> float:
    if not relevant:
        return 0.0
    top_k = retrieved[:k]
    hits = sum(1 for doc in top_k if doc in relevant)
    return hits / len(relevant)


def f1_at_k(retrieved: list[str], relevant: set[str], k: int) -> float:
    p = precision_at_k(retrieved, relevant, k)
    r = recall_at_k(retrieved, relevant, k)
    if p + r == 0:
        return 0.0
    return 2 * p * r / (p + r)


def average_precision(retrieved: list[str], relevant: set[str]) -> float:
    if not relevant:
        return 0.0
    hits = 0
    running_sum = 0.0
    for rank, doc in enumerate(retrieved, start=1):
        if doc in relevant:
            hits += 1
            running_sum += hits / rank
```

```
    return running_sum / len(relevant)


def reciprocal_rank(retrieved: list[str], relevant: set[str]) -> float:
    for rank, doc in enumerate(retrieved, start=1):
        if doc in relevant:
            return 1.0 / rank
    return 0.0


def ndcg_at_k(retrieved: list[str], relevant: set[str], k: int) -> float:
    def dcg(docs: list[str], rel: set[str], cutoff: int) -> float:
        score = 0.0
        for i, doc in enumerate(docs[:cutoff], start=1):
            if doc in rel:
                score += 1.0 / math.log2(i + 1)
        return score

    actual_dcg = dcg(retrieved, relevant, k)
    # Ідеальний DCG: всі релевантні зверху
    ideal_docs = list(relevant) + [""] * k
    ideal_dcg = dcg(ideal_docs, relevant, k)
    if ideal_dcg == 0:
        return 0.0
    return actual_dcg / ideal_dcg

class EvaluationReport:

    def __init__(self, k: int = 5) -> None:
        self.k = k
        self._records: list[dict] = []

    def add(
        self,
        query: str,
        retrieved: list[str],
        relevant: set[str],
    ) -> dict:
        record = {
            "query": query,
            f"P@{self.k}": precision_at_k(retrieved, relevant, self.k),
            f"R@{self.k}": recall_at_k(retrieved, relevant, self.k),
            f"F1@{self.k}": f1_at_k(retrieved, relevant, self.k),
            "AP": average_precision(retrieved, relevant),
            "RR": reciprocal_rank(retrieved, relevant),
            f"NDCG@{self.k}": ndcg_at_k(retrieved, relevant, self.k),
        }
        self._records.append(record)
        return record

    def summary(self) -> dict:
```

```
        if not self._records:
            return {}
        keys = [k for k in self._records[0] if k != "query"]
        summary = {}
        for key in keys:
            values = [r[key] for r in self._records]
            label = key
            if key == "AP":
                label = "MAP"
            elif key == "RR":
                label = "MRR"
            summary[label] = float(np.mean(values))
        return summary

    def reset(self) -> None:
        """Очищає накопичені записи."""
        self._records.clear()

def compare_systems(
    semantic_model,
    tfidf_model,
    queries: list[str],
    k: int = 5,
) -> dict[str, dict]:
    sem_report = EvaluationReport(k=k)
    tfidf_report = EvaluationReport(k=k)

    for query in queries:
        sem_results = semantic_model.search(query, k=k)
        tfidf_results = tfidf_model.search(query, k=k)

        # Ground truth: топ-3 semantic результати
        relevant = {r["text"] for r in sem_results[:3]}

        sem_retrieved = [r["text"] for r in sem_results]
        tfidf_retrieved = [r["text"] for r in tfidf_results]

        sem_report.add(query, sem_retrieved, relevant)
        tfidf_report.add(query, tfidf_retrieved, relevant)

    return {
        "semantic": sem_report.summary(),
        "tfidf": tfidf_report.summary(),
    }
```

A.5 Модуль compare_transformers.py

```
"""
compare_transformers.py
Порівняння трансформерних моделей на спільному ground truth від MPNet.

Логіка:
  1. Будуємо ground truth один раз — топ-3 результати MPNet для кожного запиту.
  2. Оцінюємо всі три моделі на цьому спільному еталоні.
  3. MPNet отримає 1.0 (він сам еталон), інші — реальні нижчі значення.
"""

import sys, os
sys.path.insert(0, os.path.dirname(__file__))

import numpy as np
from core import build_system
from evaluation import (
    EvaluationReport,
    precision_at_k, recall_at_k, f1_at_k,
    average_precision, reciprocal_rank, ndcg_at_k,
)

TEST_QUERIES = [
    "space exploration",
    "artificial intelligence technology",
    "climate change environment",
    "stock market financial crisis",
    "football world cup championship",
    "coronavirus pandemic health",
    "election presidential campaign",
    "Apple iPhone new features",
    "machine learning deep neural networks",
    "renewable energy solar power",
]

K = 5
GROUND_TRUTH_K = 3  # топ-3 MPNet як спільний еталон

MODELS = {
    "MiniLM-L6":  "all-MiniLM-L6-v2",
    "MiniLM-L12": "all-MiniLM-L12-v2",
    "MPNet":      "all-mpnet-base-v2",
}


def evaluate_on_shared_gt(semantic, queries, ground_truth, k):
    """Оцінює модель на спільному ground truth."""
    report = EvaluationReport(k=k)
    for q in queries:
        results = semantic.search(q, k=k)
        retrieved = [r["text"] for r in results]
```

```
        relevant = ground_truth[q]
        report.add(q, retrieved, relevant)
    return report.summary()


print("\nКрок 1: будуємо ground truth від MPNet...")
mpnet_semantic, _, _ = build_system(
    model_name="all-mpnet-base-v2",
    index_path="MPNet.index",
    texts_path="MPNet.npy",
)

ground_truth = {}
for q in TEST_QUERIES:
    results = mpnet_semantic.search(q, k=GROUND_TRUTH_K)
    ground_truth[q] = {r["text"] for r in results}

print(f"Ground truth сформовано: {GROUND_TRUTH_K} документів на запит\n")

print("=" * 90)
print(f"{'Модель':<15} {'P@5':>8} {'R@5':>8} {'F1@5':>8} {'MAP':>8} {'MRR':>8}
{'NDCG@5':>9}  {'Розмірність':>12}")
print("=" * 90)

results_all = {}

for short_name, model_name in MODELS.items():
    semantic, _, _ = build_system(
        model_name=model_name,
        index_path=f"{short_name}.index",
        texts_path=f"{short_name}.npy",
    )

    # Розмірність ембедингів
    dim = semantic.model.get_sentence_embedding_dimension()

    metrics = evaluate_on_shared_gt(semantic, TEST_QUERIES, ground_truth, k=K)
    results_all[short_name] = metrics

    print(
        f"{short_name:<15}"
        f"{metrics['P@5']:>8.4f}"
        f"{metrics['R@5']:>8.4f}"
        f"{metrics['F1@5']:>8.4f}"
        f"{metrics['MAP']:>8.4f}"
        f"{metrics['MRR']:>8.4f}"
        f"{metrics['NDCG@5']:>9.4f}"
        f"  {dim:>12}"
    )
```

```
print("=" * 90)
print(f"\nGround truth: топ-{GROUND_TRUTH_K} результати MPNet (all-mpnet-base-v2)")
print("Примітка: MPNet = 1.000 скрізь, бо він сам є еталоном.")
print("Реальна інформація — у показниках MiniLM-L6 та MiniLM-L12 відносно MPNet.\n")
```