Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«17» травня 2024 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*

освітньо-наукова програма *Кібербезпека*
(назва освітньої програми)

на тему: «Механізми категоризації та пошуку медіа контенту для проведення OSINT аналізу»

Виконавець: студент ІІ курсу, групи КБм-22

_____ **Юрій СОКИРАН**
(підпис) (Ім'я, ПРІЗВИЩЕ)

	Ім'я, ПРІЗВИЩЕ	**Підпис**
Науковий керівник	Іван ПАРХОМЕНКО	
Нормоконтроль	Лариса МИРУТЕНКО	

Київ 2024

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«17» листопада 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*

Здобувача(ки) КБм-22 (група) Сокирана Юрія Юрійовича (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Механізми категоризації та пошуку медіа контенту для проведення OSINT аналізу

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 5 від 15.11.2023 р.

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень	Процес побудови системи для категоризації медіа-контенту.
Предмет досліджень	Механізм застосування технології вбудовувань для порівняння медіа-даних, його застосування та етапи.
Мета	Реалізація механізму категоризації та пошуку медіа контенту для проведення OSINT аналізу.
Вихідні дані для проведення роботи	Стратегії розвідки на основі відкритих даних, методи автоматизації процесу пошуку даних, технології комп'ютерного зору.

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна	удосконалено процес аналізу, збирання зображень, кадрів відео із використанням вбудовувань для числового представлення контенту та подальшого порівняння.
Практична цінність	покращення роботи дослідника великої бази даних зображень за допомогою системи пошуку.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Робота виконана у повному обсязі відповідно до теми.

5. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Уточнення постановки задачі	17.11.2023 – 29.01.2024
Аналіз літературних джерел	30.01.2024 – 12.02.2024
Ознайомлення з основними характеристиками розвідки на основі відкритих джерел	13.02.2024 – 21.02.2024
Розгляд методів пришвидшення та автоматизації процесу збору даних	22.02.2024 – 26.02.2024
Дослідження можливостей комп'ютерного зору для проведення OSINT аналізу	27.02.2024 – 12.03.2024
Аналіз технології вбудовувань, огляд їх можливостей, обмежень та сфер застосувань	13.03.2024 – 20.03.2024
Дослідження існуючих векторних баз даних, їх переваги та недоліки	21.03.2024 – 25.03.2024
Автоматизація збору бази даних медіа-контенту для аналізу	26.03.2024 – 17.04.2024
Побудова системи пошуку зібраного медіа-контенту	18.04.2024 – 25.04.2024
Оформлення пояснювальної записки згідно методичних рекомендацій	26.04.2024 – 12.05.2024
Подача пакету документів на розгляд ЕК	13.05.2024 – 17.05.2024

6. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект	Зниження вартості та витрат часу на розслідування великих баз медіа-контенту.
Соціальний ефект	Пришвидшення роботи розвідки на основі відкритих джерел в сферах журналістських розслідувань, кібербезпеки тощо.

7. ДОДАТКОВІ ВИМОГИ

Завдання видав _____ Іван ПАРХОМЕНКО
(підпис) (Ім'я, ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ Юрій СОКИРАН
(підпис) (Ім'я, ПРІЗВИЩЕ)

Дата видачі завдання: 17.11.2023 р.
Термін подання кваліфікаційної роботи до ЕК 17.05.2024 р.

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел. Основний текст займає 89 сторінок, включає в себе зміст, вступ, чотири розділи дипломної роботи, висновки та список джерел. У пояснювальній записці дипломної роботи міститься 13 рисунків. Список використаних джерел містить 79 найменувань.

Методи дослідження кваліфікаційної роботи:

- аналіз літератури на тему розслідувань;
- аналіз документації та інструментів провідних платформ для обробки даних, таких як Weavite, Milvus, Elasticsearch.
- дослідження блогів, спільнот фахівців з комп'ютерного зору та обробки мультимедіа даних. Відстеження новітніх досягнень і технологій в цій галузі;

Об'єктом дослідження є процес побудови системи для категоризації медіа-контенту.

Предметом дослідження є механізм застосування технології вбудовувань для порівняння медіа-даних, його застосування та етапи.

Метою роботи є реалізація механізму категоризації та пошуку медіа контенту для проведення OSINT аналізу.

Для досягнення зазначеної мети поставлено наступні завдання:

- дослідити основні принципи OSINT розслідувань, основні типи джерел інформації необхідні для проведення розслідувань;
- провести аналіз можливостей комп’ютерного зору задля визначення об’єктів на зображеннях;
- розглянути поняття векторних баз даних та дослідити популярні рішення, альтернативи;
- побудувати серверну частину, яка буде обробляти зображення та повертати числове представлення, результат буде збережено в векторній базі даних;

- побудувати клієнтську частину із інтерфейсом задля полегшення процесу пошуку та порівняння наявних даних.

Практична цінність – покращення роботи дослідника великої бази даних зображень за допомогою системи пошуку.

Наукова новизна – удосконалено процес аналізу, збирання зображень, кадрів відео із використанням вбудовувань для числового представлення контенту та подальшого порівняння.

Ключові слова: OSINT, векторні бази даних, вбудовування, комп’ютерний зір.

ЗМІСТ

ВСТУП

Сьогодні в епоху цифрових технологій та безпрецедентного розвитку інформаційних систем стрімко зростає значення розвідки на основі відкритих джерел (OSINT) [1]. Ця методика збору та аналізу даних з публічно доступних джерел набуває все більшої актуальності в різноманітних сферах, від державної безпеки та розслідувальної журналістики до бізнес-розвідки та кіберзлочинності.

Поширення соціальних мереж, популярність відеохостингів і загальнодоступність величезних обсягів даних в Інтернеті відкривають нові можливості для OSINT-розвідки. Водночас, це створює виклики щодо ефективного пошуку, обробки та аналізу релевантної інформації серед надлишку даних. Тому виникає нагальна потреба в розробці інноваційних підходів та інструментів, здатних полегшити процес OSINT та підвищити його результативність.

Стрімкий розвиток комп'ютерного зору та методів глибинного навчання відкриває нові можливості для аналізу візуального контенту. Сучасні моделі здатні розпізнавати та класифікувати об'єкти на зображеннях, визначати їх семантичні характеристики, а також генерувати компактні векторні представлення (вбудовування) зображень, які можна використовувати для пошуку схожого контенту. Це робить комп'ютерний зір потужним інструментом для OSINT-розвідки. Тому одним із перспективних напрямків є застосування технологій комп'ютерного зору та обробки природної мови для автоматизації збору, структурування і семантичного аналізу даних з відкритих джерел. Зокрема, такі методи, як раніше згадані векторні вбудовування, дозволяють ефективно представляти і порівнювати різнорідний контент - від тексту до зображень та відео.

У цій роботі було проведено розробку системи для пошуку медіа-контенту, зокрема зображень, за допомогою векторних вбудовувань. Така система матиме значні переваги порівняно з традиційними методами, оскільки зможе знаходити семантично близькі зображення навіть за відсутності точних текстових збігів у

метаданих. Це дозволить аналітикам OSINT виявляти приховані зв'язки та отримувати більш повну картину під час розслідувань.

РОЗДІЛ 1
ХАРАКТЕРИСТИКИ ТЕХНОЛОГІЇ OSINT

OSINT (від англ. Open Source Intelligence) - це концепція, методологія і технологія пошуку, збору, обробки та аналізу інформації з відкритих (публічних) джерел з метою отримання розвідувальних даних [1].

1.1 Сфери застосування OSINT

OSINT використовується в таких сферах, як національна безпека, правоохоронні органи, бізнес-розвідка, кібербезпека, журналістські розслідування тощо [2].

Застосування у сфері національної безпеки та оборони:

- Збір розвідувальних даних про потенційні загрози національній безпеці, такі як тероризм, екстремізм, шпигунство тощо.
- Моніторинг військового потенціалу та розвитку зброї противника.
- Аналіз геополітичної ситуації, виявлення зон напруги та потенційних конфліктів.
- Розвідка цілей та планування військових операцій.

Застосування у правоохоронних органах:

- Розслідування кримінальних справ, пошук зниклих людей, встановлення особистості злочинців.
- Збір даних про злочинні групи, їхню структуру та діяльність.
- Моніторинг соціальних мереж та інтернету для виявлення потенційних правопорушень.
- Розслідування фінансових злочинів, шахрайства, відмивання грошей.

Застосування у бізнес-розвідці:

- Збір інформації про конкурентів, їхні продукти, стратегії та плани розвитку.
- Аналіз ринків, трендів та перспектив для власного бізнесу.
- Виявлення потенційних партнерів, постачальників та клієнтів.

- Оцінка ризиків та можливостей у бізнес-середовищі.

Застосування у кібербезпеці:

- Виявлення вразливостей у програмному забезпеченні, мережах та системах.
- Моніторинг шкідливої активності, фішингових атак та мереж, використовуваних злочинцями.
- Аналіз методів та технологій, що використовуються кіберзлочинцями.
- Оцінка ризиків та загроз для захисту інформаційних систем.

Застосування під час журналістських розслідувань:

- Перевірка фактів, що наводяться в різних джерелах, шляхом пошуку підтверджуючої або спростовуючої інформації.
- Виявлення прихованої інформації, що не була оприлюднена офіційно.
- Розслідування діяльності організацій, компаній та окремих осіб, які представляють громадський інтерес.
- Аналіз тенденцій, виявлення непослідовностей та зв'язків у розслідуваних питаннях.

1.2 Приклади використання OSINT в Україні

В Україні OSINT використовується в таких сферах [3][4]:

Ідентифікація користувачів в Інтернеті:

- Встановлення особистості та реальних даних про користувачів інтернет-ресурсів за їхніми нікнеймами, аватарами, метаданими та іншою інформацією, що залишається в публічному доступі.
- Збір даних про приналежність користувачів до певних спільнот, груп чи організацій на основі їхньої онлайн-активності.
- Виявлення зв'язків між різними обліковими записами, що можуть належати одній особі.
- Аналіз взаємодії користувачів у соціальних мережах та на форумах для встановлення їхніх інтересів, переконань та ролі в організаціях.

Аудит використання комп'ютерних мереж:

- Моніторинг активності в локальних мережах організацій та державних установ для виявлення потенційних загроз безпеці.
- Аналіз трафіку та протоколів передачі даних для виявлення підозрілих з'єднань та неправомірного використання ресурсів.
- Перевірка конфігурацій, налаштувань безпеки та політик доступу до мереж для оцінки їхньої стійкості до атак.
- Виявлення вразливостей та слабких місць у захисті інформаційних систем організацій.

Інформування військових про пересування російської техніки:

- Аналіз відкритих джерел, таких як соціальні мережі, новинні сайти та пошукові системи для пошуку інформації про пересування російських військ та техніки.
- Верифікація та підтвердження даних про місцезнаходження військових підрозділів та бойової техніки противника з різних джерел.
- Надання оперативної інформації військовим підрозділам ЗСУ про пересування ворожих військ для оцінки ризиків та планування операцій.

Викриття фейків та пропаганди у ЗМІ:

- Моніторинг інформаційного простору, включаючи традиційні ЗМІ та соціальні мережі, для виявлення підозрілого або незвичного контенту.
- Аналіз джерел, методів поширення та контексту повідомлень для визначення можливих ознак дезінформації або пропаганди.
- Перевірка фактів, даних та посилань, що наводяться в повідомленнях, шляхом пошуку альтернативних джерел та підтверджуючої інформації.
- Викривання фейкових повідомлень, підробок та спотвореної інформації, а також джерел їх походження.

Верифікація втрат російських військ:

- Збір інформації з відкритих джерел, таких як соціальні мережі, новинні сайти та повідомлення, про втрати російських військ у рядових та офіцерському складі.

- Перевірка та підтвердження даних про загиблих, поранених та полонених російських військових з різних джерел.
- Співставлення інформації про втрати з даними про відомих військових, їхнє місцезнаходження та місця дислокації підрозділів.
- Аналіз загальної картини втрат для оцінки боєздатності російських військ та прогнозування їхніх подальших дій.

Отже, OSINT є потужним легальним інструментом отримання розвідувальних даних з відкритих джерел, який активно застосовується в Україні в умовах війни.

1.3 Джерела, які використовуються у ході розслідувань

Основні джерела інформації в OSINT [5]:

Деякі поширені джерела інформації з відкритих джерел (OSINT) включають:

- Загальнодоступні записи та державні бази даних: вони включають судові записи, державні тендери, патенти, дані перепису населення, статистику злочинів тощо. Вони надають велику кількість загальнодоступних даних, які можна проаналізувати.
- Новинні засоби масової інформації та соціальні медіа. Традиційні інформаційні видання, а також соціальні медіа-платформи, як-от Twitter, Facebook і LinkedIn, надають потік даних у реальному часі, які можна збирати з метою розвідки.
- Корпоративні публічні дані: публічні компанії публікують багато даних у своїх фінансових документах, прес-релізах, веб-сайтах, списках вакансій тощо. Ці дані дають змогу зрозуміти їхню діяльність, технології тощо.
- Пошукові системи та веб-сайти: пошукові системи індексують величезні обсяги веб-даних, які можна шукати та аналізувати. Самі веб-сайти розкривають інформацію про технології та послуги, які використовують різні організації.
- Супутникові зображення та геопросторові дані. Супутникові дані забезпечують візуальний інтелект, а геопросторові дані пропонують уявлення на основі розташування. Обидва можуть ідентифікувати об’єкти, дії та зміни з часом.

- Темна мережа (dark web): такі веб-сайти та мережі пропонують важкодоступні дані, які містять інформацію про кіберзагрози, вразливості, інструменти злому тощо. Спеціалізовані пошукові системи, такі як DarkSearch, збирають дані темної мережі [5].

Таким чином, OSINT охоплює широкий спектр загальнодоступних джерел, починаючи від урядових, ЗМІ, корпоративних, технічних та інших важкодоступних джерел. У поєднанні та аналізі ці джерела пропонують потужну розвідувальну інформацію, використання якої є законним і етичним.

1.4 Види розвідки по відкритих джерелах

Різні види OSINT-розвідки різні методи та підходи, які використовуються для збору та аналізу відкритих джерел інформації. Основні види збору розвідданих включають:

1. Розвідка людських джерел (HUMINT, Human Intelligence): це збір, обробка, аналіз та розповсюдження інформації, зібраної з людських джерел. Це може включати проникнення та взаємодію з загрозливими суб'єктами в підпільних злочинних мережах, на форумах і ринках, та інших цільових середовищах, включаючи dark web [6].

Метою HUMINT є збір інформації про супротивників та їхню діяльність, щоб дізнатися більше про людей, які стоять за кібератаками, включно з їхніми мотивами, цілями та технікою. Цю інформацію, особливо в поєднанні з даними та статистичними даними, зібраними за допомогою інструментів безпеки та іншої телеметрії, можна використовувати, щоб допомогти організаціям визначити ризики та запобігти атакам.

Оскільки зловмисники використовують більш складні та просунуті методи атак, такі як безсигнатурні атак, які важко виявити лише за допомогою технологій, HUMINT став необхідним компонентом стратегії кібербезпеки.

HUMINT надає кілька важливих переваг організаціям і державним установам. До них належать [6]:

1. Оповіщення потенційних жертв про напад, що насувається. Збір даних із цих каналів для попередження неминучих атак вимагає набагато більше, ніж просто збір необроблених даних. За допомогою HUMINT кваліфіковані мисливці за загрозами можуть проникнути в ці середовища, щоб зібрати детальнішу інформацію про цих зловмисників та їхні плани, а потім зробити важливий крок — попередити жертв про загрозливу або поточну атаку.

2. Перевірка даних, зібраних за допомогою автоматизованої розвідки. Зловмисники розуміють роль, яку відіграють інструменти безпеки в автоматизації збору даних із цільових середовищ. У результаті багато хто почав навмисно приховувати деталі в публікаціях, наприклад імена жертв або домени. HUMINT необхідний для виявлення нових тенденцій і перевірки надійності та повноти даних, зібраних цими інструментами.

3. Обґрунтування можливостей зловмисника. HUMINT також представляє надзвичайну цінність, коли організацію було скомпрометовано, допомагаючи відділу безпеки зрозуміти заяви, зроблені хакером. Наприклад, під час атаки програми-вимагача зловмисник може стверджувати, що має набагато більші можливості, ніж насправді, у надії отримати значний викуп. У цій ситуації HUMINT може допомогти команді безпеки перевірити заяви зловмисника та відреагувати відповідно.

4. Підтримка правоохоронних органів. Відомо, що цифрові злочини важко розслідувати та притягувати до відповідальності, особливо якщо злочинець працює в іншій країні, а не в країні, де знаходиться жертва. HUMINT, зібраний постачальником послуг кібербезпеки, є надзвичайно цінним для правоохоронних органів, оскільки команда безпеки може поділитися актуальною та необхідною інформацією профілю. Це може включати справжні імена, місце проживання, громадянство та інші важливі деталі, якщо це можливо.

2. Сигнальна розвідка (SIGINT, Signals Intelligence) — це тип збору розвідувальної інформації, який передбачає перехоплення та збір інформації з різних електронних сигналів. Це важливий інструмент, який використовується багатьма організаціями, особливо військовими, щоб краще зрозуміти дії та наміри своїх опонентів [7].

Агентство національної безпеки (NSA) є основною організацією сигнальної розвідки, яка відповідає за збір і аналіз сигналів для виявлення розвідданих. Однак інші організації мають власні можливості SIGINT, наприклад Управління цифрових інновацій ЦРУ та Відділ національної безпеки ФБР [7].

Інформація, зібрана за допомогою SIGINT, допомагає швидше та ефективніше приймати рішення під час виявлення можливих загроз. Хоча SIGINT не є новим, спосіб його збирання змінився з моменту його створення.

SIGINT використовує різні методи, наприклад:

- Прослуховування радіохвиль
- Моніторинг супутникового зв'язку
- Декодування зашифрованих повідомлень
- Прослуховування телефонних розмов
- Аналіз відкритих даних

Для отримання такого роду інформації розвідувальні органи можуть використовувати електронні засоби, включаючи супутники, літаки зі спеціальним обладнанням, підслуховуючі пристрої та перехоплення волоконно-оптичних кабелів. Усі дані, зібрані за допомогою SIGINT, можна використовувати для контррозвідувальних цілей шляхом визначення можливостей і намірів іноземної держави чи інших супротивників.

Використання військовослужбовцями розвідданих відіграє вирішальну роль у забезпеченні кращого розуміння планів і можливостей противника. SIGINT часто поєднується з іншими формами розвідки, такими як HUMINT [8].

Основною метою SIGINT є збір тактичної інформації, яку військові можуть використовувати проти своїх супротивників, перш ніж вони зможуть діяти. Це також допомагає військовим швидко аналізувати вхідні дані для більш ефективного прийняття рішень. Це може бути особливо важливо під час війни, коли кожна секунда на рахунку [8].

SIGINT кардинально змінився протягом багатьох років завдяки технологічному прогресу, який дозволив використовувати більш складні методи збору інформації.

Тепер він включає такі речі, як радіоелектронна боротьба, контррозвідка та кібероперації, які сучасні військові використовують у своїх цілях.

3. Географічна розвідка (GEOINT, Geospatial Intelligence): це збір інформації за допомогою аналізу географічних даних. Це може включати використання супутникових зображень, картографічних даних та інших географічних інформаційних систем.

GEOINT — це термін, створений агентством уряду США в 2005 році для оборонної аналітики, геопросторової розвідки та аналізу урядових даних. GEOINT охоплює всі аспекти зображень і геопросторової інформації. Він не обмежується аналізом буквальних знімків місцевості, а й включає інформацію, яка була технічно отримана в результаті обробки, аналізу спектральних, просторових і часових даних. Ці типи даних можна збирати на нерухомих і рухомих цілях за допомогою електрооптичних радарів, пов'язаних сенсорами і нетехнічними засобами (це, наприклад, геопросторова інформація, отриману персоналом у необхідній локації) [7].

Сфери застосування GEOINT є наступними [9]:

- Комерційні підприємства — GEOINT формує переписну, історичну, метеорологічну та геологічну інформацію для багатьох комерційних цілей. Точні дані GEOINT використовуються від сфери подорожей, нерухомості до нафтогазової промисловості.
- Військові — GEOINT допомагає планувати та виконувати дії, налагоджувати мережі розподілу та операції.
- Автомобільна промисловість - безпека автономних транспортних засобів залежатиме від високоточних даних GEOINT.
- Пожежники — дані GEOINT допомагають боротися з лісовими пожежами та забезпечувати безпеку пожежників за допомогою точного місцезнаходження та даних про погоду.
- Сільське господарство — дані GEOINT можуть максимізувати здоров'я рослин і мінімізувати використання добрив на великих ділянках землі завдяки точному рівню деталізації розташування.

4. Розвідка вимірювання та сигнатур (MASINT, Measurement and Signature Intelligence): це збір інформації шляхом аналізу фізичних характеристик, таких як акустичні, сейсмічні, радіочастотні та інші сигнатури.

5. Кібер розвідка (CYBINT, Cyber Intelligence): це збір інформації шляхом аналізу кібернетичних даних. Це може включати використання кібернетичних засобів для збору інформації з веб-сайтів, соціальних медіа, електронної пошти та інших цифрових джерел [10].

Публічність і доступність Інтернету та його ресурсів для всіх людей у всьому світі можуть створити як можливості, так і проблеми для CYBINT. Це робить можливим збір даних про будь-яку фізичну чи юридичну особу в усьому світі, а також може створювати подібні можливості для хакерів і зловмисних користувачів. Вони можуть використовувати проксі-сервери, віртуальні приватні мережі (VPN), анонімайзери або інструменти підробки, щоб приховати або підробити свою особу. Користувачі з усього світу з обмеженими ресурсами і навіть навичками можуть мати серйозний вплив у всьому світі, враховуючи наявність великого переліку безкоштовних хакерських і інструментів із відкритим кодом [10].

6. Фінансова розвідка (FININT, Financial Intelligence): це збір інформації шляхом аналізу фінансових даних. Це може включати використання фінансових звітів, банківських виписок, торговельних даних та інших фінансових джерел.

Фінансова розвідка є критично важливим інструментом, який використовується урядами, правоохоронними органами та фінансовим сектором для боротьби з фінансовими злочинами та захисту національної та економічної безпеки. FININT передбачає збір та аналіз фінансових даних, щоб зрозуміти фінансову діяльність зацікавлених організацій, передбачити їхні наміри та виявити незаконну діяльність, таку як відмивання грошей, ухилення від сплати податків, фінансування тероризму та злочинних організацій [11].

7. Розвідка соціальних мереж (SOCMINT, Social Media Intelligence): це збір інформації шляхом аналізу даних з соціальних медіа. Інформацію, зібрану за допомогою SOCMINT, можна розділити на три основні категорії [12]:

- Інформація про профіль: це включає в себе статичні деталі, надані користувачами в їхніх профілях у соціальних мережах, такі як посади, роботодавці, навички та контактна інформація.
- Взаємодії: це охоплює різні способи взаємодії користувачів із платформою та один з одним, включаючи дописи, коментарі, оцінки "подобається" та поширення
- Метадані: окрім самого вмісту, метадані надають контекстну інформацію про вміст, таку як місцезнаходження, позначене тегом у публікації, час її публікації або використовуваний пристрій.

Кожна з цих дисциплін має свої власні методи, інструменти та підходи до збору та аналізу даних, і кожна може надати унікальні переваги в залежності від конкретного контексту та цілей збору розвідданих [12].

Висновки за розділом 1

У цьому розділі було розглянуто концепцію розвідки на основі відкритих джерел та її різноманітні сфери застосування. OSINT є потужним інструментом збору розвідувальних даних з загальнодоступних джерел, який широко використовується у сферах національної безпеки, правоохоронних органів, бізнес-розвідки, кібербезпеки, журналістських розслідувань тощо.

Було наведено приклади практичного застосування OSINT в Україні, зокрема для ідентифікації користувачів в Інтернеті, аудиту комп'ютерних мереж, інформування про пересування російської техніки, викриття фейків і пропаганди, верифікації втрат російських військ. Це демонструє, наскільки важливу роль відіграє OSINT в умовах війни для збору розвідданих та протидії дезінформації.

Розглянуто основні джерела даних для OSINT-розслідувань, які охоплюють широкий спектр ресурсів: від державних баз даних, ЗМІ, корпоративних сайтів до геопросторових даних, темної мережі та соціальних медіа. Поєднання та аналіз цих різноманітних джерел дозволяє отримувати цінні розвідувальні знання.

Також було класифіковано види розвідки за відкритими джерелами, включаючи розвідку людських джерел (HUMINT), сигнальну розвідку (SIGINT), географічну розвідку (GEOINT), розвідку вимірювань та сигнатур (MASINT), кібер-розвідку (CYBINT), фінансову розвідку (FININT) та розвідку соціальних мереж (SOCMINT). Кожен з цих видів має свої специфічні методи, інструменти та сфери застосування.

Таким чином, OSINT є багатогранною галуззю, що активно розвивається і знаходить все більше застосування в різних сферах завдяки стрімкому зростанню доступних даних та вдосконаленню аналітичних методів обробки інформації.

РОЗДІЛ 2
ПРИШВИДШЕННЯ ТА АВТОМАТИЗАЦІЯ ЗБОРУ ДАНИХ

2.1 Основні складнощі при OSINT-розвідці

1. Перевантаження інформацією: величезна кількість даних, доступних в Інтернеті, ускладнює сортування та пошук відповідної та надійної інформації [13]. Це включає не тільки традиційні веб-сайти, але й соціальні медіа, форуми, бази даних і багато іншого. Це вимагає досконалого розуміння, як фільтрувати та сортувати дані, щоб відокремити відповідні інформаційні фрагменти.

2. Якість і надійність даних: забезпечення точності та надійності інформації є серйозною проблемою, оскільки дані надходять із різних джерел і не завжди можуть бути надійними[13][14]. З огляду на велику кількість джерел, може бути складно відрізнити вірогідну інформацію від недостовірної або навіть намісно неправдивої. Професійні OSINT-аналітики часто використовують низку методів і складних технік для перевірки всієї зібраної інформації. Інформація в Інтернеті постійно змінюється: нові дані додаються, старі дані видаляються або оновлюються. Це означає, що інформація, яка була правильною одного дня, може бути недостовірною наступного.

3. Верифікація та автентифікація: перевірка автентичності інформації та її джерел є критично важливою, що вимагає структурованих аналітичних методів і розуміння етики належного OSINT[15].

4. Труднощі співпраці: ефективна співпраця та комунікація є важливими, особливо коли задіяно кілька дослідників і зацікавлених сторін, що може бути складно під час віддаленої роботи [13].

5. Ризики для безпеки та конфіденційності: робота з конфіденційною інформацією створює значні ризики для безпеки та конфіденційності, що вимагає запобіжних заходів, таких як безпечні платформи для збору, зберігання та обміну даними, а також найкращі методи захисту даних [13]. Інформація з великої кількості відкритих джерел може бути інфікувана шкідливим ПЗ або бути частиною кібератаки.

Отже, аналітикам потрібно знати, як безпечно збирати дані, не піддаючи себе або їх системи ризику.

6. Інтеграція технологій: інтеграція кількох спеціалізованих інструментів і програмного забезпечення для збору, обробки та аналізу даних може бути складною та ресурсомісткою[13]. Обробка даних, особливо у великих обсягах, вимагає глибокого розуміння та навичок використання технологій і програмного забезпечення, таких як автоматичний збір даних, машинне навчання, аналіз даних тощо.

7. Юридичні та етичні міркування: OSINT має проводитися в рамках закону, і етичні міркування повинні бути прийняті до уваги, особливо коли йдеться про особисті дані чи місцезнаходження[16]. Законодавство з приватності і захисту даних варіюється від країни до країни, а робота з OSINT потребує отримання відповіді на ці проблеми. Також слід дотримуватися почуття етичності, оскільки збирання і використання деяких видів інформації можуть викликати моральні питання.

8. Обмеження ресурсів: необхідність ефективного управління обмеженими ресурсами при роботі з великими наборами даних є постійною проблемою[13].

9. Фільтрація вмісту: високоякісні дані потрібні для цінних розвідувальних звітів, що означає застосування фільтрів вмісту для керування обсягом даних[16].

10. Геополітична чутливість: OSINT, що використовується в оборонних програмах, може привернути геополітичну увагу через транскордонний моніторинг даних, який може бути політично чутливим[16].

2.2 Стратегії вирішення проблем OSINT

Щоб подолати ці проблеми, дослідники OSINT використовують спеціалізовані інструменти та методи для управління даними, встановлюють чіткі протоколи зв’язку, перевіряють джерела, використовують безпечні платформи та автоматизують повторювані завдання [14]. Їм також необхідно постійно розвивати свої навички та адаптуватися до нових технологій і методологій [16][17].

Щоб подолати інформаційне перевантаження в OSINT, дослідники можуть реалізувати такі стратегії:

1. Автоматизувати збір і обробку даних: автоматизація може обробляти величезні обсяги неструктурованих даних, що становить близько 90% даних, з якими мають працювати аналітики. Такі інструменти, як Silobreaker, можуть автоматизувати збір, перевірку та стандартизацію даних із різних джерел, мінімізуючи людські помилки та дозволяючи аналітикам зосередитися на інтерпретації [18].

2. Використовувати централізовані платформи: використання центральної платформи, яка не залежить від постачальника даних, допомагає порівнювати джерела та ефективніше вимірювати надійність і довіру. Це також дозволяє аналізувати та візуалізувати інформацію для виявлення стосунків, ключових гравців і прихованих зв'язків [18].

3. Використовувати розширені аналітичні інструменти: впровадження інструментів, які дозволяють виявляти складні об'єкти та створювати зв'язки між різними сутностями, може допомогти аналітикам ідентифікувати та пов'язувати різні псевдоніми та явища з тією самою сутністю, гарантуючи, що релевантна інформація не буде пропущена [19].

4. Диверсифікація джерел: розширення колекції за всіма темами, загрозами, суб'єктами, географічними регіонами та варіантами використання може пом'якшити упередженість, охоплюючи широкий діапазон джерел і перспектив [19].

5. Ефективна обробка інформації: використання рішень аналізу даних, які сприяють фільтрації неструктурованих даних на всіх етапах циклу обробки інформації, може призвести до більш обсягів даних, які було успішно опрацьовано [18].

6. Розвиток навичок: навчання дослідників даних і аналітиків для покращення їх навичок може допомогти їм краще керувати якістю даних і застосовувати фільтри вмісту для забезпечення високоякісних даних для цінних звітів розвідки [20].

7. Використання штучного інтелекту і машинного навчання: швидкий розвиток штучного інтелекту, машинного навчання та аналітики великих даних може

допомогти точніше аналізувати дані та робити точніші висновки, таким чином керуючи великими обсягами даних OSINT [21].

Завдяки інтеграції цих підходів дослідники OSINT можуть ефективніше керувати перевантаженням інформацією, дозволяючи їм створювати дієві та своєчасні розвідувальні дані.

В контексті OSINT, автоматизація та пришвидшення процесу збору даних можуть бути досягнуті за допомогою різних технологій та підходів. Одним з ключових інструментів для автоматизації збору даних є парсинг.

Парсинг - це процес витягування даних з веб-сайтів або інших джерел інформації. Це може бути використано для автоматичного збору великих обсягів даних з відкритих джерел, що значно пришвидшує процес збору інформації [22].

Парсинг може бути використаний для збору різноманітних типів даних, включаючи текст, зображення, метадані та інше. Це може бути особливо корисним для збору даних з великих веб-сайтів або баз даних, де ручний збір даних був би непрактичним або неможливим [22].

Однак, важливо зазначити, що парсинг повинен бути виконаний відповідно до законодавства та політики конфіденційності відповідних веб-сайтів. Деякі веб-сайти можуть обмежувати або забороняти парсинг, тому важливо завжди перевіряти політику конфіденційності та умови використання веб-сайту перед початком парсингу [23].

Використання автоматизованих інструментів для збору даних може допомогти значно збільшити швидкість та ефективність процесу збору даних. Однак, важливо пам'ятати, що автоматизація може також призвести до збору невірної або неповної інформації, тому важливо завжди перевіряти та аналізувати зібрані дані [23].

2.3 Різниця між парсінгом та скрепінгом

Різниця між парсингом і скрейпінгом є фундаментальною для процесу збору та аналізу даних.

Скрейпінг даних:

- Мета скрейпінгу - це збір даних. Він передбачає автоматизоване вилучення інформації з веб-сайтів або платформ соціальних мереж [24].
- Процес: скрейпінг працює шляхом надсилання HTTP-запитів до веб-серверів і отримання відповідей, які містять дані у вигляді HTML або інших веб-форматів. Ці дані, як правило, в необробленому і неструктурованому вигляді [25].
- Інструменти та методи: для скрепінгу веб ресурсів зазвичай використовують такі інструменти, як Python з такими бібліотеками, як BeautifulSoup і Selenium. Ці інструменти здійснюють навігацію по веб-сторінках, витягують певні дані (текст, зображення, посилання тощо) і збирають їх у необробленому форматі [25].
- Складнощі: скрейпінг повинен враховувати заходи протидії скрейпінгу, такі як CAPTCHA, обмеження швидкості та блокування IP-адрес. Часто це вимагає використання проксі-серверів і складних методів для імітації поведінки людини в Інтернеті, щоб уникнути виявлення [25].

Парсинг даних:

- Мета: парсинг даних - це аналіз і структурування зібраних даних. Він бере необроблені дані, отримані в результаті скрапінгу, і організовує їх у більш читабельний і структурований формат, такий як JSON, CSV або база даних [26].
- Обробка: парсинг передбачає отримання неструктурованих або напівструктурованих даних і перетворення їх у структуровану форму. Цей процес включає очищення даних, виявлення закономірностей та організацію даних таким чином, щоб зробити їх придатними для аналізу [26].
- Інструменти та методи: парсинг можна виконувати за допомогою різних мов програмування та бібліотек, призначених для обробки тексту, XML або JSON. Вибір інструментів залежить від формату вихідних даних і бажаного вихідного формату [27].
- Проблеми: основною проблемою парсингу є робота з різноманітністю форматів і структур даних. Парсери повинні бути достатньо гнучкими та надійними, щоб впоратися з невідповідностями та помилками у вихідних даних [27].

Отже, скрейпінг- це процес збору даних з Інтернету, тоді як парсинг - це процес перетворення цих даних у структурований формат для аналізу. Обидва вони є

важливими етапами процесу OSINT, причому скрейпінг слугує засобом для збору необроблених даних, а парсинг - методом, що дозволяє зробити ці дані придатними для аналізу та дій [27].

2.4 Скрейпінг динамічних сайтів

Використання автоматизованого браузера або браузеру без інтерфейсу (англ. headless browser) слід використовувати замість звичайних HTML-запитів для скрапінгу в наступних сценаріях:

1. Динамічний контент: Коли вміст веб-сайту динамічно завантажується за допомогою JavaScript, браузер без інтерфейсу може виконувати JavaScript так само, як звичайний браузер, гарантуючи, що весь вміст буде відображено перед вилученням [28].

2. Інтерактивні елементи: Якщо веб-сайт вимагає взаємодії, наприклад, натискання кнопок, заповнення форм або навігації через робочі процеси користувача, автоматизація браузера може імітувати ці дії для доступу до контенту [29].

3. Складна навігація: Деякі веб-сайти мають складні схеми навігації, які вимагають підтримки станів сеансу, файлів cookie або обробки перенаправлень, якими можна ефективно керувати за допомогою безголових браузерів [29].

4. CAPTCHA та виявлення ботів: Якщо на веб-сайті є засоби захисту від збору даних, такі як CAPTCHA або алгоритми виявлення ботів, браузери без інтерфейсу можуть допомогти з ними впоратися, імітуючи взаємодію з людиною [30].

Використання браузерів без інтерфейсу та інструментів автоматизації браузерів, таких як Selenium, Puppeteer або Playwright, може забезпечити більш надійне та гнучке рішення для скрапінгу в цих сценаріях, дозволяючи більш складну взаємодію з веб-сторінками та роботу з сучасними веб-технологіями[31].

Парсінг та скрапінг можуть значно допомогти в управлінні перевантаженням інформацією в OSINT (Open Source Intelligence) шляхом автоматизації збору та початкової обробки даних із різних відкритих джерел. Ось як ці методи сприяють подоланню інформаційного перевантаження [32]:

1. Ефективний збір даних:

Веб-скрейпінг автоматизує процес збору даних із веб-сайтів, платформ соціальних мереж, форумів та інших онлайн-джерел. Це економить час і зусилля порівняно зі збором даних вручну та забезпечує швидкий збір великих обсягів даних.

2. Вилучення структурованих даних:

Парсінг корисний для систематичного аналізу даних (часто необроблених даних, зібраних за допомогою аналізу) для вилучення значущої інформації. Він перетворює неструктуровані або напівструктуровані дані в структурований формат, який легше аналізувати та розуміти. Потім ці структуровані дані можна легше шукати, порівнювати та аналізувати.

3. Автоматизований моніторинг:

Інструменти збирання можна налаштувати для постійного моніторингу певних веб-сайтів або джерел даних, сповіщаючи дослідників про нову інформацію, щойно вона стає доступною. Це гарантує, що найновіші дані завжди аналізуються, що є вирішальним для своєчасної та релевантної інформації OSINT.

4. Фільтрування та релевантність:

Як синтаксичний аналіз, так і сканування можна налаштувати для пошуку конкретних ключових слів, фраз або шаблонів у даних, відфільтровуючи нерелевантну інформацію. Це допомагає зосередитися на найбільш відповідних даних, зменшуючи обсяг інформації, яку потрібно переглядати вручну.

5. Збагачення даних:

Скрепінг та парсинг також можна використовувати для збагачення зібраних даних додатковим контекстом або метаданими. Наприклад, сканування може збирати дати, імена авторів або географічні розташування, пов’язані з даними, тоді як аналіз може класифікувати інформацію на основі попередньо визначених критеріїв. Ці збагачені дані забезпечують глибший рівень розуміння та полегшують більш складний аналіз.

6. Масштабованість:

Оскільки обсяг доступних онлайн-даних продовжує зростати, сканування та аналіз пропонують масштабовані рішення для досліджень OSINT. Ці методи можуть обробляти зростаючі обсяги даних без пропорційного збільшення часу або

необхідних ресурсів, що робить їх незамінними інструментами для управління інформаційним перевантаженням.

Для OSINT-парсингу використовуються різноманітні інструменти. Ось деякі з них:

1. OSINT Framework - це інтерактивний інструмент, який допомагає користувачам визначити, які OSINT-інструменти використовувати на основі введених даних [2].

2. Babel X - це платформа для моніторингу загроз, яка збирає та аналізує дані з відкритих джерел [2].

3. Shodan - це пошукова система, яка дозволяє користувачам знаходити конкретні типи комп'ютерів, веб-камер, маршрутизаторів та інших пристроїв, підключених до Інтернету[2].

4. Maltego - це інструмент для візуалізації та аналізу OSINT-даних, який дозволяє користувачам створювати графічні моделі для аналізу зв'язків між об'єктами[2].

Ці інструменти можуть бути використані для автоматизації та пришвидшення процесу збору даних в рамках OSINT-досліджень.

Висновки за розділом 2

У цьому розділі було розглянуто основні виклики та стратегії для подолання складнощів при OSINT-розвідці. Ключовими проблемами є перевантаження інформацією, забезпечення якості та надійності даних, верифікація та автентифікація джерел, складнощі співпраці, ризики для безпеки та конфіденційності, інтеграція технологій, юридичні та етичні міркування, обмеження ресурсів, фільтрація вмісту та геополітична чутливість.

Для вирішення цих проблем дослідники OSINT можуть реалізувати різні стратегії, такі як автоматизація збору та обробки даних, використання централізованих платформ, розширених аналітичних інструментів, диверсифікація

джерел, ефективна обробка інформації, розвиток навичок аналітиків та застосування штучного інтелекту і машинного навчання.

У розділі також детально розглянуто парсинг і скрепінг як ключові методи для автоматизації збору та обробки даних з відкритих джерел. Парсинг - це процес витягування та структурування даних, тоді як скрепінг - це автоматизований збір необроблених даних. Вони допомагають подолати інформаційне перевантаження, забезпечуючи ефективний збір даних, вилучення структурованої інформації, автоматизований моніторинг, фільтрацію та збагачення даних.

Крім того, розглянуто особливості скрепінгу динамічних веб-сайтів за допомогою автоматизованих браузерів і безголових браузерів.

Таким чином, цей розділ підкреслює важливість ефективних стратегій і інструментів для подолання труднощів OSINT-розвідки та управління величезними обсягами даних з відкритих джерел.

РОЗДІЛ 3 ЗАСТОСУВАННЯ КОМП’ЮТЕРНОГО ЗОРУ ДЛЯ OSINT-У

3.1 Можливості комп'ютерного зору для аналізу зображень

Комп'ютерний зір є важливою галуззю штучного інтелекту, яка займається обробкою та аналізом візуальної інформації. В контексті аналізу зображень, комп'ютерний зір використовується для розпізнавання, класифікації, відстеження та сегментації об'єктів на зображеннях [37].

По-перше, візуальна геолокація. Вона використовує зображення для визначення географічного розташування об'єктів. Це може бути досягнуто за допомогою порівняння зображень з базою даних карт або використанням візуальних орієнтирів для визначення місця розташування без GPS-сигналів, як це було продемонстровано в тестуванні системи візуальної навігації армією США [38].

У цьому контексті комп'ютерний зір відіграє важливу роль в аналізі зображень для GEOINT. Аналіз зображень з супутників і безпілотних літальних апаратів дозволяє автоматично виявляти та відстежувати об'єкти, визначати зміни в ландшафті, інфраструктурі тощо. Це забезпечує цінні розвідувальні дані для військових, урядових та комерційних організацій.

Фотограметрія, яка використовує комп'ютерне бачення для вимірювання та відтворення 3D-форм з фотографій, допомагає створювати точні 3D-моделі місцевості, будівель та інфраструктури. А також автоматичне розпізнавання патернів в супутникових знімках та аерофотознімках може допомогти у виявленні посівів, забруднень, змін в землекористуванні тощо для цілей моніторингу довкілля [39].

По-друге, аналіз зображень у великій кількості. Для аналізу великої кількості зображень використовуються алгоритми машинного навчання та нейронні мережі, які можуть автоматично класифікувати, виявляти та сегментувати об'єкти на зображеннях [40]. Ці методи дозволяють швидко обробляти великі набори даних і виявляти важливі візуальні особливості.

По-третє, форензичний аналіз зображень. Цей аналіз зображень використовується для перевірки автентичності фотографій, виявлення редагувань та маніпуляцій. Інструменти, такі як Forensically, дозволяють аналізувати артефакти стиснення, градієнти країв та інші деталі, що можуть вказувати на зміни в зображенні [41].

Комп'ютерний зір продовжує розвиватися, і нові технології, такі як глибоке навчання, розширюють можливості аналізу зображень. Це включає покращення точності розпізнавання об'єктів, здатність працювати з низькоякісними зображеннями та в реальному часі [42].

Комп'ютерний зір застосовується в різних сферах, включаючи безпеку, сільське господарство, медицину, військову справу та багато інших. Він може використовуватися для автоматизації процесів, підвищення точності та ефективності роботи [43].

У контексті кібербезпеки, комп'ютерний зір може використовуватися для OSINT аналізу, дозволяючи ідентифікувати та аналізувати візуальний контент з відкритих джерел для збору розвідувальної інформації [43].

3.2 Застосування технологій вбудовувань для пошуку зображень

3.2.1 Загальна характеристика вбудовувань

Вбудовування (англ. embeddings) відіграють вирішальну роль у проведенні запитів до баз даних за допомогою комп'ютерного зору, перетворюючи зображення на високовимірні вектори, які можуть бути ефективно оброблені та порівняні [45]. Ось як вбудовування допомагають у конкретних завданнях, пов'язаних із запитами комп'ютерного зору:

1. Пошук схожих зображень: вбудовування дозволяють порівнювати високовимірні вектори, що представляють зображення, дозволяючи ідентифікувати візуально схожі зображення, навіть якщо вони не є точно такими ж. Це особливо

корисно в системах відновлення зображень, де метою є знаходження зображень, візуально схожих на запитуване зображення [46].

2. Кластеризація зображень: завдяки перетворенню зображення на вбудовування, стає можливим застосування алгоритмів кластеризації для групування зображень на основі візуальної схожості. Це може бути використано для організації великих колекцій зображень у значущі категорії без ручного маркування, що є корисним для завдань, таких як організація фотогалерей або аналіз тенденцій візуального контенту [47].

3. Аналіз великих колекцій: При роботі з великими наборами даних, вбудовування можуть значно знизити розмірність даних, зберігаючи при цьому основну візуальну інформацію. Це робить можливим виконання завдань, таких як аналіз тенденцій, виявлення аномалій та категоризація контенту в великому масштабі. Моделі машинного навчання можуть обробляти ці вбудовування для ідентифікації шаблонів та робити прогнози щодо зображень [48].

Загалом, вбудовування є потужним інструментом для перетворення візуальних даних у формат, який може бути легко маніпульований та аналізований, сприяючи виконанню різноманітних завдань комп'ютерного зору, пов'язаних із запитами до баз даних.

3.2.2 Обмеження вбудовувань

Вбудовування мають власний набір практичних обмежень і проблем, особливо коли вони застосовуються в реальних випадках використання. Ці обмеження можуть значно вплинути на фазу попередньої обробки даних і навчання, а також на продуктивність і масштабованість моделі.

По-перше, це попередня обробка даних і навчання.

1. Робота з невідомими або позалексичними словами. Однією із значних проблем із вбудовуванням, особливо в нейролінгвістичному програмуванні (галузь штучного інтелекту, яка займається аналізом, розумінням та генерацією людської мови за допомогою комп'ютерів), є робота з невідомими або позалексичними

словами. Коли слова немає в навчальних даних, модель намагається призначити йому значущий вектор, часто вдаючись до випадкового або загального вектора, що може погіршити продуктивність наступних завдань [49].

2. Розрідженість даних і висока розмірність: вбудовування може страждати від розрідженості даних, коли конкретне слово чи функція не є частиною навчального корпусу, його вбудовування не може бути згенероване, що призводить до низької продуктивності в таких завданнях, як класифікація та генерація тексту [50] . Крім того, вбудовані компоненти зазвичай мають високу розмірність, що означає, що вони можуть мати великий обсяг пам’яті та менший час обчислень, що ускладнює їх використання в середовищах з обмеженими ресурсами [49].

3. Обчислювальна складність. Алгоритми, які використовуються для генерації вбудовувань, потребують багато обчислень і можуть зайняти багато часу на їх виконання. Ця висока обчислювальна складність може бути перешкодою, особливо коли маєте справу з великими наборами даних або коли потрібна швидка обробка [51].

По-друге, це продуктивність і масштабованість моделі.

1. Семантичний дрейф: з часом значення слів може змінюватися, що призводить до так званого семантичного дрейфу. Це явище може призвести до того, що вбудовані елементи застаріють, оскільки вони більше не точно відображають поточне використання або значення слів, що потенційно може призвести до неточних результатів у завданнях прогнозування [49].

2. Омографи та флексія: вкладення можуть мати проблеми з такими мовними явищами, як омографи (слова, які пишуться однаково, але мають різні значення) і флексія (зміни слова для вираження різних граматичних категорій). Ці проблеми можуть суттєво вплинути на продуктивність систем машинного навчання, оскільки вбудовування може не точно вловлювати нюанси значень або граматичні варіації слів [52].

3. Згортання вбудовування в масштабуванні. Під час масштабування моделей, особливо моделей рекомендацій, виникає явище, відоме як «згортання вбудовування», коли матриця вбудовування має тенденцію розташовуватися в

низьковимірному підпросторі, що зрештою перешкоджає масштабованості. Ця проблема підкреслює проблему підтримки продуктивності та масштабованості моделі в міру збільшення розміру моделі [53].

4. Алгоритмічне зміщення: впровадження даних, на яких навчаються, можуть внести або зберегти зміщення в моделі. Особливо це стосується додатків, які безпосередньо стосуються окремих осіб, як-от інструменти найму чи системи схвалення кредитів. Забезпечення алгоритмічної чесності під час використання вбудовування є значною проблемою [54].

5. Вимоги до ресурсів: більші та складніші моделі вбудовування вимагають значних обчислювальних ресурсів, включаючи пам’ять і обчислювальну потужність. Це може обмежити їх використання в середовищах з обмеженими ресурсами або збільшити вартість розгортання [55].

Підсумовуючи, хоча вбудовування пропонують потужні інструменти для представлення даних у низьковимірному просторі, вони мають низку обмежень і проблем, які необхідно вирішити. До них належать робота з невідомими словами, розрідженість даних, висока розмірність, обчислювальна складність, семантичний дрейф, лінгвістичні явища, такі як омографи та флексія, а також специфічні проблеми в таких областях, як клінічний текст. Вирішення цих проблем вимагає постійних досліджень і розробки більш складних алгоритмів, попередньо навчених вбудовувань і методів зменшення розмірності та лінгвістичної попередньої обробки.

3.2.3 Сфери застосування вбудовувань

Вбудовування є потужною технікою в машинному навчанні, що має численні практичні застосування у різноманітних галузях. Приклади застосування вбудовувань у реальних сценаріях наступні:

1. Рекомендаційні системи, такі як Netflix, використовують вбудовування для розуміння уподобань користувачів щодо фільмів, базуючись на словах з описів фільмів та іменах акторів [56]. Порівнюючи ці вбудовування з новими фільмами,

система може рекомендувати контент, подібний до того, що користувачі раніше позитивно оцінювали.

2. Пошукові системи, такі як Google, застосовують вбудовування до речень для розуміння контексту та семантичного значення пошукових запитів користувачів [57]. Це дозволяє отримувати релевантні результати, навіть якщо запит сформульовано нестандартним чином.

3. Безпілотні автомобілі використовують вбудовування зображень для ідентифікації об'єктів на дорозі, таких як дорожні знаки, пішоходи та інші перешкоди. Ця інформація дозволяє автомобілю безпечно орієнтуватися в дорожній обстановці та приймати швидкі рішення [57].

4. Фінансові установи застосовують вбудовування для виявлення шахрайських транзакцій, аналізуючи закономірності у фінансових даних та виявляючи підозрілу діяльність.

5. Музичні рекомендаційні сервіси, такі як Spotify, використовують вбудовування до аудіо для розуміння музичних особливостей пісень, які слухають користувачі. Це дозволяє рекомендувати схожий контент та створювати персоналізовані плейлисти відповідно до смаків користувачів.

6. Чат-боти та віртуальні асистенти застосовують вбудовування для розуміння змісту запитань і прохань користувачів, навіть якщо вони сформульовані неформально або неповно [56]. Це забезпечує більш природну та ефективну взаємодію.

7. Персоналізовані стрічки новин на платформах, таких як Facebook та Bluesky, використовують вбудовування до публікацій для адаптації контенту відповідно до інтересів користувачів. Розуміючи зміст і настрій статей, вони можуть показувати релевантні та цікаві новини [57].

8. Інтернет-магазини застосовують вбудовування до товарів для розуміння їхніх особливостей та атрибутів. Це дозволяє показувати релевантні результати під час пошуку та рекомендувати схожі продукти на основі попередніх покупок користувачів.

9. Системи кібербезпеки використовують вбудовування до мережевого трафіку для виявлення аномалій та потенційних загроз. Аналізуючи закономірності мережевої активності, вони можуть ідентифікувати зловмисну діяльність і запобігати кібератакам [57].

Ці приклади демонструють різноманітність та широке застосування вбудовувань у галузі машинного навчання для вирішення практичних завдань та покращення досвіду користувачів у різноманітних контекстах.

3.2.4 Типи вбудовувань

У комп'ютерному зорі для проведення запитів до баз даних використовуються різні типи вбудовувань, які дозволяють представляти зображення у вигляді високовимірних векторів. Ці вектори можуть бути ефективно оброблені та порівняні, що сприяє виконанню різноманітних завдань, таких як пошук схожих зображень, кластеризація зображень за їхньою візуальною схожістю, аналіз великих колекцій зображень тощо. Основні типи вбудовувань, які використовуються у комп'ютерному зорі, включають:

1. Вбудовування з Convolutional Neural Networks (CNNs): CNNs часто використовуються для генерації вбудовувань для зображень. Глибокі шари CNN можуть навчитися кодувати візуальний контент зображення у компактний вектор, який вловлює суть зображення. Ці вбудовування можуть використовуватися для завдань, таких як класифікація зображень або порівняння за схожістю [58].

2. Вбудовування з Siamese Networks: Ці мережі призначені для генерації вбудовувань, які можуть використовуватися для порівняння схожості між двома зображеннями. Вони навчаються, використовуючи пари зображень, і особливо корисні для завдань, таких як верифікація осіб або відстеження об'єктів [59]. Також є варіація Siamese networks, яка використовує triplet loss. Ці моделі навчаються на трійках зображень (основа, позитивний приклад і негативний приклад), щоб навчитися генерувати вбудовування, які наближають схожі зображення одне до

одного та віддаляють несхожі зображення у просторі вбудовувань. Вони корисні для завдань, таких як пошук зображень [60].

3. Вбудовування з CLIP (Contrastive Language-Image Pretraining): Розроблена OpenAI, CLIP є багатомодальною моделлю, яка може генерувати вбудовування як для тексту, так і для зображень. Вона була навчена на великому наборі пар зображень і тексту і може використовуватися для завдань класифікації без попередньо заданих даних, коли модель може класифікувати зображення у категорії, які вона не бачила під час навчання [61].

Ці типи вбудовувань дозволяють ефективно виконувати запити до баз даних, використовуючи комп'ютерний зір, завдяки можливості представлення зображень у вигляді високовимірних векторів, які можуть бути порівняні та аналізовані.

3.3 Застосування векторних баз даних задля оптимізації пошуку

Векторні бази даних стали потужним інструментом для ефективного пошуку та пошуку зображень. Використовуючи концепцію вбудовування, ці бази даних забезпечують швидкий і точний пошук подібності у великомасштабних колекціях зображень.

Традиційні бази даних, розроблені для структурованих даних, погано підходять для роботи з вбудованими зображеннями великої розмірності. Тому для таких задач доцільно використовувати саме векторні бази даних. Векторні бази даних спеціально розроблені для зберігання та керування векторними даними, такими як вбудовування зображень [62].

Ці бази даних забезпечують спеціалізовані механізми індексування та запитів, оптимізовані для векторів великої розмірності. Вони використовують такі методи, як алгоритми пошуку приблизного найближчого сусіда (англ. approximate nearest neighbour - ANN), щоб ефективно знаходити найбільш подібні вбудовування до заданого вбудовування запиту [63].

Щоб забезпечити швидкий пошук подібності, векторні бази даних створюють індекси на збережених вбудованих зображеннях. Ці індекси є структурами даних, які

розділяють векторний простір і дозволяють швидко отримувати найближчі вбудовування [63].

Загальні методи індексування, які використовуються у векторних базах даних, включають:

- Деревоподібні методи: ці методи поділяють векторний простір на ієрархічні області, що забезпечує ефективний просування по дереву під час пошуку[2].

- Методи на основі хешування: методи хешування відображають багатовимірні вбудовування в компактні двійкові коди, що забезпечує швидкий наближений пошук подібності [63].

- Методи на основі квантування: квантування стискає вкладення в менше представлення, зберігаючи відносні відстані між ними [63].

Коли надається зображення запиту, його вбудовування обчислюється та використовується для пошуку в індексованій базі даних. Векторна база даних використовує вибраний метод індексування для ефективного отримання найбільш схожих вбудованих зображень на основі визначеної метрики подібності, наприклад косинусної подібності або евклідової відстані [62].

3.3.1 Застосування векторних баз даних у пошуку зображень

Векторні бази даних мають численні застосування в задачах пошуку та пошуку зображень. Деякі відомі випадки використання включають:

1. Пошук зображень на основі вмісту: Векторні бази даних дозволяють шукати візуально схожі зображення на основі їх вмісту, а не покладатися на текстові метадані [62].

2. Дедуплікація зображень: шляхом порівняння вбудованих зображень векторні бази даних можуть ідентифікувати та видаляти дублікати або майже дублікати зображень із великих колекцій [62].

3. Візуальний пошук продуктів: платформи електронної комерції можуть використовувати векторні бази даних, щоб дозволити користувачам шукати візуально схожі продукти на основі еталонного зображення [62].

4. Кластеризація та категоризація зображень: векторні бази даних сприяють групуванню візуально подібних зображень у кластери або категорії на основі їх близькості вбудовування [62].

Векторні бази даних у поєднанні з вбудованими зображеннями забезпечують потужне рішення для ефективного та точного пошуку зображень. Використовуючи семантичну та візуальну інформацію, отриману під час вбудовування, ці бази даних забезпечують швидкий пошук подібності у великомасштабних колекціях зображень. Завдяки спеціалізованим можливостям індексування та запитів векторні бази даних стали важливим інструментом у різних областях, від пошуку зображень на основі вмісту до візуального пошуку продуктів. Оскільки обсяг візуальних даних продовжує зростати, важливість векторних баз даних у програмах пошуку зображень лише зростатиме.

Основна відмінність між векторними базами даних і традиційними базами даних полягає в тому, як вони зберігають і обробляють дані, а також у типах запитів, для яких вони оптимізовані.

3.3.2 Порівняння реляційних та векторних баз даних

Традиційні бази даних, такі як реляційні бази даних (наприклад, MySQL, PostgreSQL) і бази даних NoSQL (наприклад, MongoDB, Cassandra), призначені для зберігання та запиту структурованих даних, організованих у таблицях із рядками та стовпцями. Вони відмінно справляються із запитами точної відповідності, транзакціями та складними операціями SQL зі структурованими даними [64].

Навпаки, векторні бази даних оптимізовані для зберігання та запитів до векторних даних великої розмірності, які представляють неструктуровані дані, такі як текст, зображення та аудіо [65].

Основні характеристики, які відрізняють векторні бази даних від традиційних баз даних:

1. Представлення даних: векторні бази даних зберігають дані як щільні векторні вбудовування, які є числовими представленнями, які фіксують семантичне значення

та контекст неструктурованих даних [66]. Традиційні бази даних зберігають дані в рядках і стовпцях.

2. Типи запитів: векторні бази даних розроблені для пошукових запитів подібності, що дозволяє користувачам знаходити точки даних, які семантично подібні до заданого вектора запиту [65]. Традиційні бази даних в основному обробляють запити точної відповідності та складні операції SQL.

3. Індексування: векторні бази даних використовують спеціалізовані методи індексування, такі як деревовидні, хешування або методи квантування, для ефективного індексування та пошуку у векторних просторах великої розмірності [67]. Традиційні бази даних використовують такі індекси, як структуру Б-дерева або хеш-індекси, оптимізовані для структурованих даних.

4. Варіанти використання: Векторні бази даних добре підходять для додатків, пов’язаних із машинним навчанням, обробкою природної мови, системами рекомендацій і пошуком на основі вмісту [66]. Традиційні бази даних краще підходять для транзакційних робочих навантажень, зберігання структурованих даних і складних запитів SQL.

У той час як традиційні бази даних чудово справляються з керуванням структурованими даними та транзакційними навантаженнями, векторні бази даних створені для роботи зі складністю неструктурованих даних і забезпечують розширені можливості пошуку за подібністю [64]. Проте векторні бази даних є відносно новою технологією, і їм може бракувати зрілості, інструментарію та екосистемної підтримки традиційних баз даних [64].

Важливо відзначити, що деякі сучасні бази даних, такі як SingleStore та Postgres, почали включати можливості пошуку векторної схожості у свої механізми, стираючи межі між традиційними та векторними базами даних [68]. Вибір між векторною базою даних і традиційною базою даних залежить від конкретних вимог програми, характеру даних і типів запитів, які необхідно підтримувати.

Є кілька ключових обмежень традиційних реляційних баз даних, коли мова йде про обробку векторних даних:

1. Складність векторних даних. Традиційні бази даних призначені для структурованих даних, організованих у таблиці з рядками та стовпцями. Векторні бази борються зі складністю та великою розмірністю векторних даних, які представляють неструктуровані дані, такі як текст, зображення та аудіо [69].

2. Можливості запитів: традиційні бази даних чудово справляються із запитами точної відповідності та складними операціями SQL над структурованими даними. Однак вони погано підходять для пошукових запитів подібності, які є вирішальними для пошуку семантично подібних точок даних у векторних просторах [70].

3. Масштабованість: у той час як традиційні бази даних можуть масштабуватися вертикально та горизонтально, вони часто стикаються з проблемами при роботі з великими неструктурованими наборами даних. Векторні бази даних розроблені для горизонтальної масштабованості, особливо в розподілених архітектурах, для обробки великого обсягу та складності векторних даних [71].

4. Обробка в режимі реального часу. Традиційні бази даних можуть мати проблеми з обробкою та аналізом великомасштабних векторних даних у режимі реального часу, що є ключовою вимогою для багатьох програм машинного навчання та науки про дані [69].

5. Інтеграція з робочими процесами штучного інтелекту та машинного навчання. Традиційні бази даних не мають вбудованої підтримки звичайних векторних операцій і можуть вимагати громіздких процесів інтеграції зі інструментами машинного навчання. Векторні бази даних забезпечують бездоганну інтеграцію та підтримку цих робочих процесів [72].

Традиційні бази даних не оптимізовані для унікальних характеристик і вимог векторних даних. Вони стикаються з обмеженнями щодо складності, запитів, індексування, масштабованості, обробки в реальному часі та інтеграції з робочими процесами AI/ML. Векторні бази даних усувають ці обмеження, надаючи спеціалізовані можливості для ефективного зберігання, керування та пошуку векторних даних великої розмірності.

Приклади спеціалізованих векторних баз даних включать:

1. Faiss – це бібліотека, розроблена Facebook для ефективного пошуку подібності та кластеризації щільних векторів [73]. Вона оптимізована для зберігання та запиту векторних даних і використовується такими компаніями, як Google, Microsoft і Amazon [74].

Faiss надає набір алгоритмів і структур даних, які забезпечують швидкий пошук найближчих сусідів і кластеризацію в просторах великої розмірності. Бібліотека призначена для роботи з великомасштабними наборами даних, що містять мільярди векторів, і може використовуватися для різних програм, таких як пошук зображень і відео, системи рекомендацій і обробка природної мови.

Faiss реалізовано на мові програмування C++ з міркувань продуктивності, але також забезпечує прив’язку до мови Python для зручності використання. Бібліотека має відкритий вихідний код і оптимізована як для архітектур центрального процесору, так і для графічного процесора, що робить її придатною для широкого діапазону апаратних конфігурацій.

2. Milvus — це векторна база даних із відкритим кодом, яка пропонує додатковий акцент на масштабованість [74].

Milvus забезпечує високопродуктивну розподілену архітектуру, яка забезпечує ефективне зберігання, індексацію та пошук величезних обсягів векторних даних. Однією з ключових переваг Milvus є його здатність горизонтального масштабування між кількома вузлами, що дозволяє обробляти мільярди векторів і підтримувати високий рівень паралелізму. Система використовує механізм шардингу, який автоматично розподіляє векторні дані між різними вузлами, забезпечуючи оптимальне балансування навантаження та продуктивність запитів.

3. Pinecone — це векторна база даних, створена на основі Faiss [75]. Вона призначений для зберігання та пошуку високовимірних векторів і використовується такими компаніями, як Google, Microsoft і Uber [74]. Pinecone надає повністю кероване хмарне рішення, яке спрощує розгортання та масштабування програм векторного пошуку.

Однією з головних переваг Pinecone є бездоганна інтеграція з фреймворками й інструментами машинного навчання. Він пропонує API та клієнтські бібліотеки для

таких популярних мов програмування, як Python, Java та JavaScript, що дозволяє легко включати можливості векторного пошуку в існуючі робочі процеси ML. Pinecone також інтегрується з такими популярними платформами машинного навчання, як TensorFlow, PyTorch і Keras, що дозволяє користувачам легко зберігати та отримувати вектори, згенеровані їхніми моделями.

Ще однією ключовою особливістю Pinecone є підтримка оновлень у реальному часі та поступового індексування. Користувачі можуть легко додавати, оновлювати або видаляти вектори в режимі реального часу без необхідності перебудувати весь індекс. Це робить Pinecone придатним для програм, які потребують динамічних і часто оновлюваних векторних даних, таких як системи рекомендацій і виявлення аномалій у реальному часі.

4. Weaviate [74]: Weaviate — це векторна база даних із відкритим вихідним кодом, призначена для зберігання та пошуку пов’язаних даних. Вона базується на пошуковій системі Elasticsearch і використовується такими компаніями, як Zalando та eBay [74].

Графоподібна модель даних Weaviate дозволяє користувачам визначати власні схеми даних і встановлювати зв’язки між об’єктами даних. Це дозволяє представляти складні моделі домену та підтримує розширені можливості запитів. Користувачі можуть переглядати графік, щоб досліджувати зв’язки між точками даних і виконувати складні запити, які поєднують векторний пошук із навігацією по графіку.

Іншим важливим аспектом Weaviate є його зосередженість на контекстуалізації даних. Він надає багатий набір функцій для додавання контексту до даних, таких як перехресні посилання, класифікації та таксономії. Ця контекстна інформація допомагає зрозуміти взаємозв’язки між точками даних і покращує пошук, надаючи більш релевантні та значущі результати.

5. Chroma [74]: Chroma — це власна база даних вбудовування з відкритим вихідним кодом, оптимізована для зберігання та пошуку великих наборів даних вбудовування. Його використовують такі компанії, як Google, Amazon і Facebook[2]. Chroma розроблено, щоб забезпечити швидкий і ефективний пошук подібності у

векторних даних великої розмірності, що робить його добре придатним для різноманітних програм ШІ та машинного навчання.

Однією з ключових переваг Chroma є його орієнтація на простоту та легкість використання. Він надає простий API, який дозволяє користувачам легко зберігати, отримувати та шукати вбудовані файли без необхідності складної конфігурації чи налаштування. Chroma абстрагує базові механізми зберігання та індексування, дозволяючи розробникам зосередитися на логіці додатків, а не на тонкощах керування векторними базами даних.

Ці спеціалізовані векторні бази даних розроблені та оптимізовані спеціально для зберігання та запиту векторних даних, дотримуючись принципу «одного розміру для всіх» [76].

3.4 Алгоритм пошуку схожих зображень за вбудовуваннями

3.4.1 Збір необхідних даних

В першу чергу проводимо скрепінг зображень: це включає в себе застосування інструментів веб-скрепінгу, таких як Scrapy, Selenium, або спеціалізованих бібліотек скрепінгу зображень для автоматизованого збору зображень з різноманітних джерел в інтернеті. Типовими джерелами можуть бути веб-сайти зображень, соціальні мережі (Instagram, Facebook тощо), месенджери (Telegram), новинні сайти, форуми та інші веб-ресурси, що містять зображення [77]. Під час скрепінгу важливо дотримуватись правил виключення (robots.txt) та політик використання даних відповідних веб-сайтів. Зібрані зображення зберігаються у відповідних форматах (JPEG, PNG та ін.) у належній структурі на диску.

Далі разом із зібраними зображеннями ми також збираємо супутні метадані та текстові дані, такі як текстові описи, заголовки, теги, коментарі користувачів тощо. Ці дані можуть суттєво підвищити якість векторних вбудовувань та точність пошуку, забезпечуючи додатковий контекст. На цьому етапі доцільно застосовувати різноманітні парсери та скрепери даних, щоб витягувати корисну інформацію з веб-

сторінок, PDF-файлів, документів та інших цифрових джерел даних. Залежно від формату даних, можуть використовуватись бібліотеки парсингу HTML/XML, Regular Expressions, спеціальні PDF-парсери тощо. Отримані текстові дані зберігаються разом із зображеннями та можуть бути використані як додаткові супутні дані для покращення та уточнення пошукових результатів.

3.4.2 Попередня обробка даних

Необхідно виконувати ретельне очищення даних: видалення шумів, помилок та некоректних даних з набору зображень та супутніх даних. Цей процес може включати:

- видалення повних дублікатів зображень та близьких дублікатів з незначними відмінностями;
- видалення зображень дуже низької якості, розмитих або сильно зашумлених;
- усунення очевидно помилкових або нерелевантних тегів та текстових описів;
- видалення зображень із вузькоспеціалізованими об'єктами, що не становлять інтересу для системи;
- заповнення відсутніх метаданих синтетичними даними, згенерованими за допомогою моделей глибокого навчання [78].

Проводимо нормалізацію даних - приведення зображень та текстових даних до єдиного уніфікованого формату для полегшення подальшої обробки.

Після очищення та нормалізації ми використовуємо отримані дані для обчислення векторних вбудовувань: застосовуємо передові моделі глибокого навчання комп'ютерного зору (наприклад, CLIP, BLIP, VGG) для обчислення векторних вбудовувань для кожного зображення в наборі даних [78]. Саме ці вбудовування буде використано для ефективного пошуку схожих зображень на наступних етапах алгоритму.

3.4.3 Побудова векторної бази даних

Вибираємо базу даних: вибір відповідної векторної бази даних, яка забезпечить ефективний пошук за векторами вбудовувань [78]. Популярними варіантами є Faiss, Weavite тощо.

Вибір зазвичай залежить від таких факторів [78]:

- розмір векторів вбудовувань (деякі бази краще працюють з високою або низькою розмірністю);
- очікуваний обсяг даних (деякі бази ефективніші для дуже великих наборів);
- підтримувані метрики відстані та типи індексів;
- вимоги до швидкодії під час пошуку та обновлення даних;
- можливості масштабування, розподіленого індексування, відмовостійкості;
- сумісність з обраними бібліотеками та фреймворками (наприклад, PyTorch, TensorFlow).

Налаштовуємо параметри: налаштування векторної бази даних, таких параметрів як розмірність векторів, тип індексування, метрика відстані (наприклад, косинусна відстань, евклідова відстань) тощо. Правильний вибір цих параметрів може значно вплинути на швидкість та точність пошуку, особливо коли мова йде про великі обсяги даних [78].

Після завантажуємо дані: обчислені вбудовування зображень та супутні дані завантажуються до векторної бази даних. Це дозволить ефективно зберігати та індексувати дані для подальшого пошуку.

Цей та попередні етапи представлено на рис. 3.1.

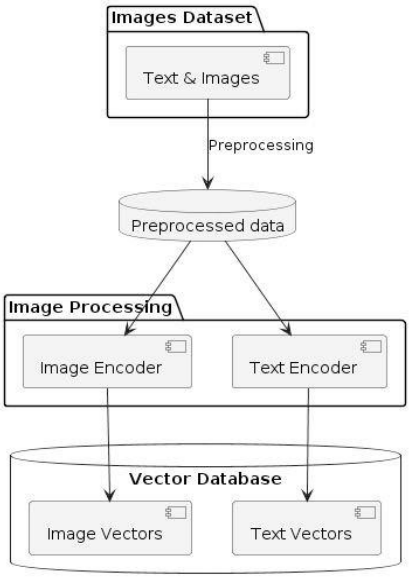


Рисунок 3.1 – Процес збирання, обробки та заповнення бази даних

3.4.4 Обробка запиту користувача

Отримуємо запит: запит користувача може бути у формі зображення, тексту або комбінації обох.

Якщо користувач надав текстовий опис або інші супутні дані, виконується необхідна обробка та очищення цих даних. На цьому етапі здійснюються наступні кроки:

- переклад тексту на одну з підтримуваних моделлю мов, якщо це необхідно;
- видалення стоп-слів, знаків пунктуації, HTML-тегів та іншого "шуму" з тексту;
- доповнення запиту синонімами ключових слів та релевантними словами з семантично близьких тем;
- перефразування запиту з метою покращення його сумісності з моделлю вбудовувань.

Якщо користувач надав запит у формі зображення, то можемо застосувати моделі глибокого навчання, щоб отримати додатковий опис отриманого зображення.

Після слідує обчислення вбудовування запиту: використовуючи ту саму модель глибокого навчання, що і для вбудовувань зображень у базі даних, обчислюємо векторне вбудовування запиту користувача.

Виконуємо пошук найближчих сусідів: застосування ефективних алгоритмів пошуку найближчих сусідів (наприклад, алгоритм k-найближчих сусідів) для знаходження векторів вбудовувань у базі даних, найближчих до вбудовування запиту.

Загальний процес обробки запиту користувача представлено на рис. 3.2.

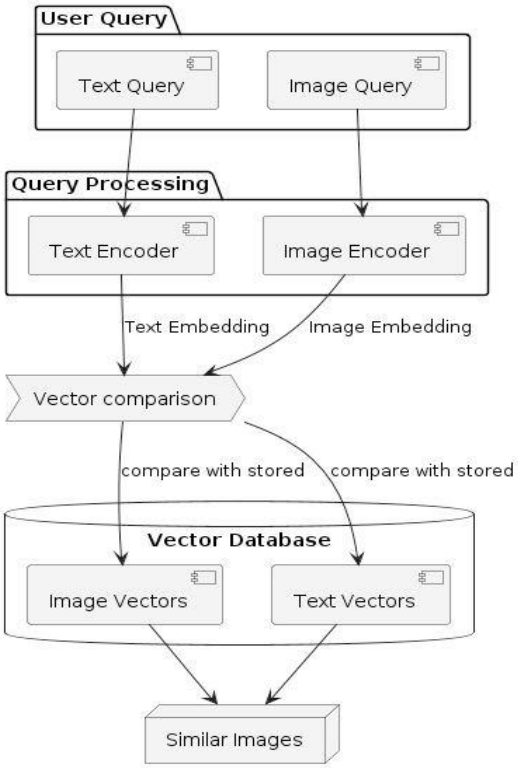


Рисунок 3.2 – Процес використання запиту користувача для пошуку схожих зображень

3.4.5 Виведення результатів пошуку

Відображення результатів: зображення запиту та перші k найближчих зображень з бази даних показуємо користувачеві, впорядковуючи їх за зменшенням подібності до запиту. Поряд з кожним знайденим зображенням можна відображати

числове значення відстані або метрику подібності до запиту для більш наочного порівняння релевантності. Зображення можуть бути представлені у вигляді мініатюр або з можливістю перегляду в повному розмірі.

Відображення метаданих: опціонально, якщо такі метадані є, система може відображати супутні метадані (текстові описи, теги, заголовки тощо) для кожного знайденого зображення, надаючи користувачеві більше контексту та покращуючи зрозумілість результатів пошуку.

Навігація та фільтрація: користувачеві може надаватися можливість переглядати більше результатів за допомогою розбиття на сторінки або безкінечного пролистування. Також може бути реалізовано фільтрування результатів за різними критеріями, такими як розмір зображення, тип файлу, джерело отримання зображення, дата створення тощо. Це дозволить користувачеві уточнювати пошукові запити та концентруватися на найбільш релевантних результатах.

Крім того, можна реалізувати функціональність для внесення уточнень до початкового запиту шляхом надання нового зображення, тексту або комбінації обох. Це дозволить користувачеві поступово звужувати область пошуку та досягати більш точних результатів запиту.

Також доцільним буде розробка спеціалізованого інтерфейсу для користувача, який би забезпечував зручну взаємодію з системою пошуку схожих зображень та чітке подання результатів. Такий інтерфейс може включати в себе: зручну панель для завантаження зображень або введення текстових запитів користувачем, структуровану галерею із сіткою мініатюр результуючих зображень, впорядкованих за релевантністю, можливість перегляду окремих зображень в повному розмірі з відображенням супутніх метаданих, наявність фільтрів та налаштувань для уточнення пошукових запитів, елементи навігації для перегляду додаткових результатів пошуку, опцію надсилання зворотного зв'язку щодо релевантності результатів.

Інтуїтивний та зрозумілий інтерфейс користувача суттєво полегшить взаємодію з системою пошуку схожих зображень, дозволить користувачеві ефективно аналізувати результати та вносити необхідні уточнення для досягнення максимальної точності пошуку відповідно до конкретних потреб.

3.4.6 Зворотний зв'язок та вдосконалення пошуку

На цьому етапі система може отримувати зворотний зв'язок від користувачів щодо релевантності результатів пошуку та використовувати цю інформацію для вдосконалення алгоритму.

Можливі кроки для поліпшення системи:

- надати користувачам можливість оцінювати корисність знайдених зображень (наприклад, за допомогою системи голосування або рейтингів);
- аналізувати отримані оцінки та виявляти випадки, коли результати пошуку не відповідали очікуванням користувача;
- використовувати ці дані для налаштування параметрів моделі вбудовувань, метрики подібності або алгоритму пошуку;
- періодично переналаштовувати систему з урахуванням отриманих даних зворотного зв'язку для покращення точності пошуку.

Це загальна структура алгоритму пошуку схожих зображень за вбудовуваннями. Кожен етап може мати додаткові деталі та варіації в залежності від конкретної реалізації та вимог до системи.

Висновки за розділом 3

У розділі було описано застосування технологій комп'ютерного зору та векторних вбудовувань для аналізу та пошуку зображень. Було розглянуто основні можливості комп'ютерного зору, такі як візуальна геолокація, аналіз великих обсягів зображень та форензичний аналіз. Також було описано концепцію векторних вбудовувань та їх застосування для представлення й порівняння зображень у векторному просторі.

Було проаналізовано переваги застосування векторних баз даних для ефективного зберігання та пошуку векторних вбудовувань зображень. Наведено порівняння векторних і традиційних реляційних баз даних, висвітлено їхні

відмінності та обмеження. Окрім того, представлено огляд популярних векторних баз даних, таких як Faiss, Milvus, Pinecone та Weaviate.

У розділі також було детально розглянуто алгоритм пошуку схожих зображень за векторними вбудовуваннями. Описано ключові етапи цього процесу: збір вихідних даних, попередня обробка, побудова векторної бази даних, обробка запиту користувача, виведення результатів пошуку та отримання зворотного зв'язку для покращення системи. Наголошено на важливості забезпечення зручного інтерфейсу для взаємодії користувача із системою пошуку зображень.

Таким чином, у розділі було продемонстровано підхід, заснований на комп'ютерному зорі та векторних вбудовуваннях, для вирішення завдань аналізу й пошуку зображень. Застосування цих технологій дозволяє ефективно індексувати, порівнювати та знаходити візуально схожі зображення в масштабних колекціях даних.

РОЗДІЛ 4
ПОБУДОВА СИСТЕМИ ДЛЯ ПОШУКУ МЕДІА КОНТЕНТУ

У практичній частини розробимо додаток, який реалізуватиме зазначений у минулому розділі алгоритм пошуку схожих зображень за вбудовуваннями.

4.1 Збір контенту для аналізу

Для побудови системи необхідно накопичити базу даних медіа контенту, серед якого будемо проводити пошук. В першу чергу треба визначитись із джерелами пошуку.

Починаючи від 24 лютого 2022 року, Росія веде повномасштабну війну проти України. Саме тому важливим є отримання різного роду розвідувальних даних у ході розслідувань. Серед ймовірних джерел, які допоможуть під час OSINT розслідувань є канали в Telegram російських медіа та воєнних кореспондентів, які часто можуть через недбалість та неосвіченість висвітлити інформацію про розміщення російських військ, їхнє спорядження, тощо.

Зібрати великий список можна почавши з одного або двох каналів, а потім, слідкуючи за тим на які джерела ці канали посилаються в своїх публікаціях, можна доповнювати цей список подібними каналами. Таким чином можна знайти невеликі канали, але такі що публікують маловідому інформацію та почати збирати її.

Для збирання інформації з Telegram скористаємось бібліотекою Telethon та мовою програмування Python, на якій ця бібліотека написана. Сам Telegram має власний API, який можуть використовувати розробники сторонніх клієнтів – і саме це API використовує бібліотека.

Для початку роботи з нею треба зареєструвати свій додаток у Telegram, аби отримати ключ для авторизації під час відправки запитів. Переходимо за посиланням https://my.telegram.org, авторизуємось та обираємо пункт «API developments tools» (рис. 4.1). Далі заповнюємо необхідні дані та отримуємо значення властивостей app_id та app_hash нашого застосунку (рис. 4.2).

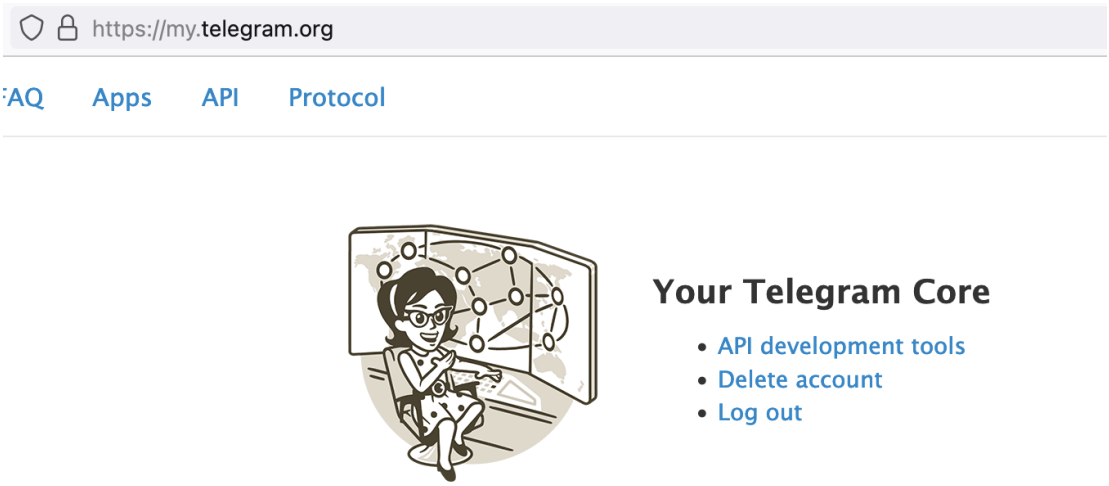


Рисунок 4.1 – Основна сторінка створення додатку Telegram

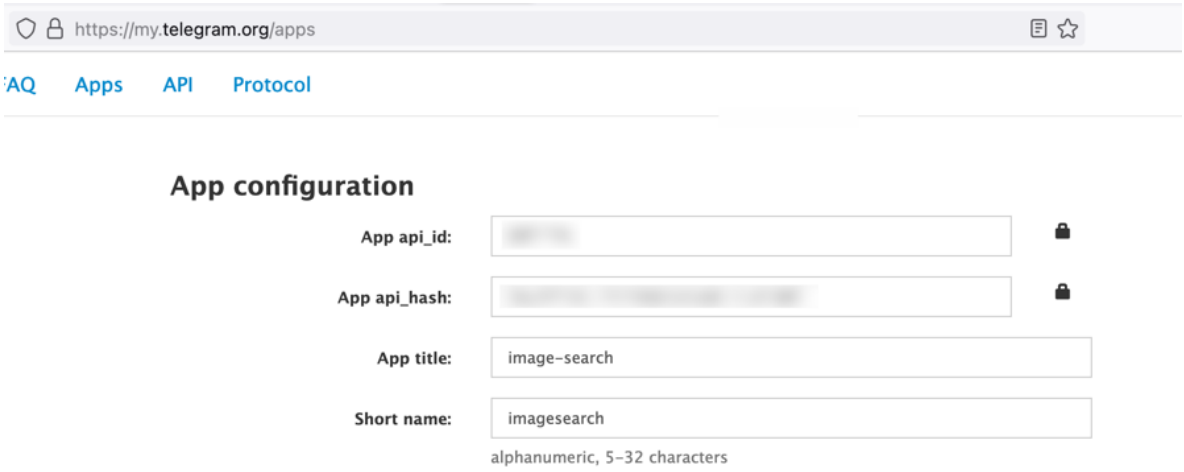


Рисунок 4.2 – Необхідні дані для роботи з API Telegram

Тепер маємо змогу використовувати ці дані для роботи з Telethon.

Маємо наступний код для початку роботи із бібліотекою:

```
from telethon import TelegramClient
api_id = 123
api_hash = "somehash"
# The first parameter is the .session file name (absolute paths allowed)
```

```
client = TelegramClient("anon", api_id, api_hash)
```

Розглянемо складові коду:

“from telethon import TelegramClient” – імпорт необхідного класу з бібліотеки.

“api_id, api_hash” – ідентифікатор та хеш, який ми отримали під час створення застосунку Telegram.

“client = TelegramClient("anon", api_id, api_hash)” – ініціалізація клієнту, передаємо усі необхідні дані в клас клієнту.

Під час першого запуску програма запитає необхідні дані для входу – авторизаційний код, тощо. Загалом у ході збору інформації буде використано персональний акаунт, з якого було створено застосунок раніше. Тому варто зареєструвати окремий акаунт задля цілей розслідування.

Також після першого запуску скрипту буде створено файлу розширення .session, в даному випадку “anon.session”. Файл сесії містить достатньо інформації, щоб скрипт міг ввійти без повторного надсилання коду.

В першу чергу необхідно дізнатися необхідні ідентифікатори каналів з яких плануємо діставати інформацію. Швидким способом буде підписатися на цей канал та викликати функцію iter_dialogs.

Код для отримання даних про діалоги:

```
client = TelegramClient("anon", api_id, api_hash)
async def get_dialogs():
    async for dialog in client.iter_dialogs():
        print(dialog.name, dialog.id)
```

“async for dialog in client.iter_dialogs()” – виклик функції-генератору, яка повертає список усіх діалогів.

“dialog” - спеціальний клас, який представляє діалог (відкриту «розмову» з кимось, групою чи каналом), надаючи абстракцію для легкого доступу до повідомлень, ідентифікаторів та різних властивостей, так як назва, стан діалог (заархівований, закріплений), дата останнього повідомлення, тощо.

Результат виклику функції виведе усі діалоги та їх інформацію. Сутність каналу тут теж вважається діалогом, тому ми так само отримуємо дані про канал. За потреби маємо змогу відфільтрувати за властивостями діалогу is_user, is_group, is_channel.

Далі є два можливих способи зібрати медіа дані з каналу. Перший – використати можливості бібліотеки Telethon задля скачування медіа-контенту. Функція get_message_photos, що очікує на вхід ідентифікатор каналу, може в циклі отримати повідомлення каналу. Функція iter_messages в якості аргументу приймає фільтр, в цьому випадку це InputMessagesFilterPhotos, який фільтрує усі повідомлення із фотографіями у змісті. Далі маємо змогу використовуючи метод download_media об’єкта message завантажити зображення у визначену директорію.

```
from telethon.tl.types import InputMessagesFilterPhotos
async def get_message_photos(chat_id):
    count = 0
    folder_name = "channel_photos"
    photo_data = {}
    async for message in client.iter_messages(
        chat_id, filter=InputMessagesFilterPhotos, limit=1500, wait_time=5
    ):
        count += 1
        filename = f"{folder_name}/{count}.jpg"
        if message.raw_text:
            shortened_text = (
                ".".join([x.capitalize() for x in message.raw_text.split(" ")[:3]])
                .replace(":", "")
                .replace("!", "")
                .replace(";", "")
                .replace("/", "_")
            )
            filename = f"{folder_name}/{count}{shortened_text}.jpg"
```

```
            await message.download_media(file=filename)
            print(f"{count}: {message.raw_text}")
            photo_data[message.id] = {
                "text": message.text,
                "date": str(message.date),
                "filename": filename,
            }
        with open(f"{folder_name}-photo_data.json", "w+", encoding="utf8") as f:
            json.dump(photo_data, f, ensure_ascii=False)
```

Розглянемо частини коду:

“from telethon.tl.types import InputMessagesFilterPhotos” – імпортуємо спеціальний клас, який надає нам бібліотека, задля фільтрування повідомлень за певними характеристиками. В цьому випадку це повідомлення, які містять в собі зображення.

“async for message in client.iter_messages()” – асинхроно отримуємо повідомлення. Функція iter_messages приймає в якості аргументів chat_id (ідентифікатор чату, діалогу), filter (фільтр, в даному випадку це спеціальний клас InputMessagesFilterPhotos), limit (максимальна кількість результатів, яку необіхдно отримати. Якщо повідомлень в чаті буде менше, то функція просто припинить свою роботу), wait_time (час очікування у секундах між різними запитами отримування історії чату. Цей параметр використовується, щоб уникнути потрапляння помилки FloodWaitError за потреби. Якщо значення відсутнє, за замовчуванням буде 1 секунда, лише якщо ліміт очікуваних повідомлень перевищує 3000).

“message.raw_text.split(" ")[:3]]).replace(":", "")” – цей та інші виклики replace використовується для того щоб утворити ім’я файлу в який ми зберігаємо зображення. Очищуємо назву від спеціальних символ щоб у системи не було помилок при спробі зберегти файл.

“await message.download_media(file=filename)” – збереження зображень конкретного повідомлення, де аргумент file – шлях до файлу, який буде створено. В даному випадку використовуємо раніше створену очищену назву файлу.

“photo_data[message.id] = {” – створюємо словник із даними повідомлення аби потім мати змогу дістати конкретне повідомлення та його зображення або відфільтрувати необхідні.

“with open(f"{folder_name}-photo_data.json", "w+", encoding="utf8") as f: json.dump(photo_data, f, ensure_ascii=False)” – зберігаємо утворений словник photo_data у файл із розширенням .json. В даному випадку важливо передавати аргумент ensure_ascii=False так як пости будуть містити кирилицю, тому вміст може зберігатися неправильно якщо функцію буде очікувати тільки ASCII символи. Стандартний набір ASCII не містить кирилицю.

Таким чином, папка виглядатиме наступним чином, як на рис. 4.3.

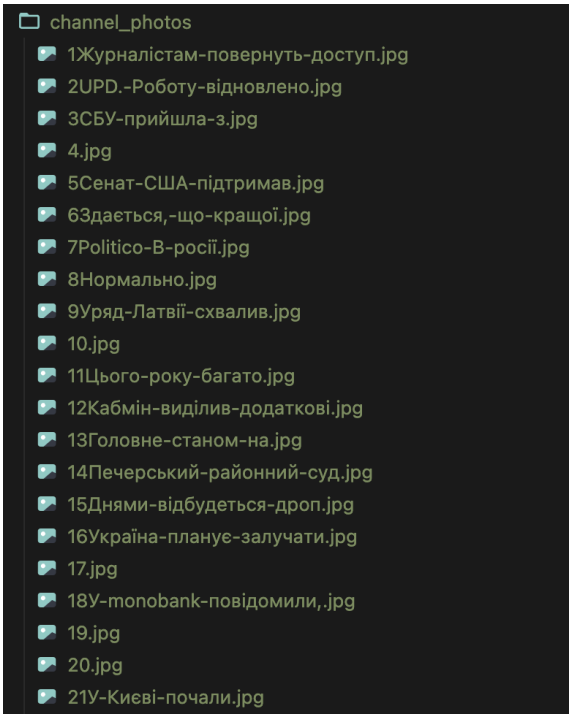
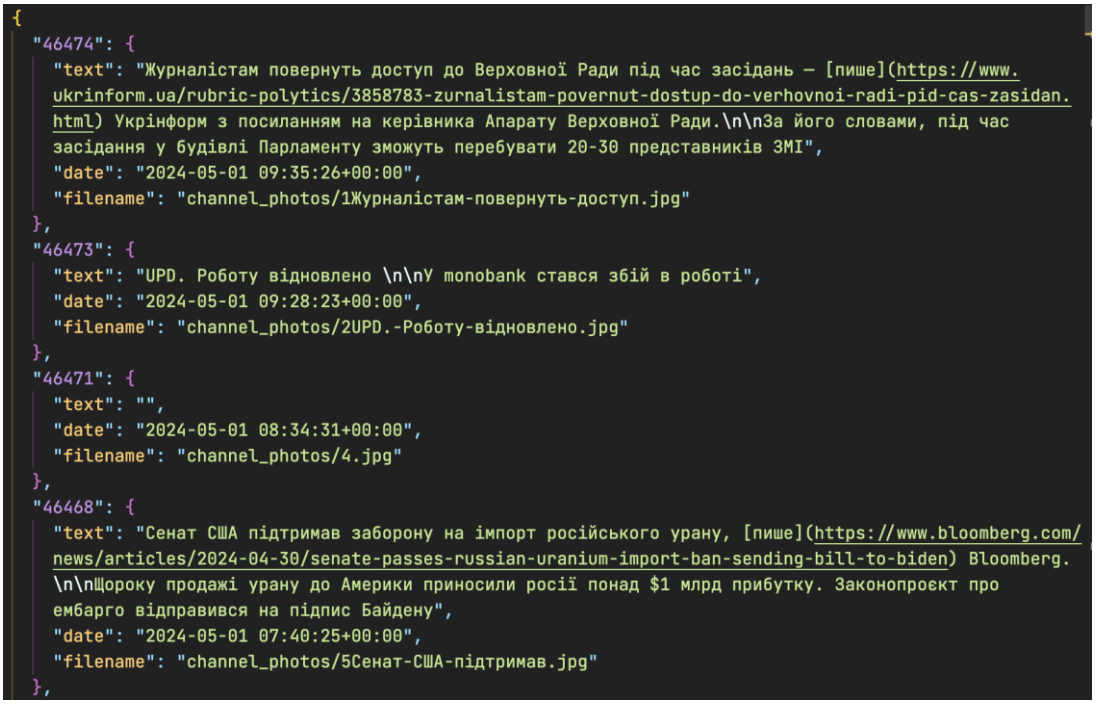


Рисунок 4.3 – Директорія із скачаними фотографіями. Фото мають відформатовану назву

Назви фотографій було складно із текстових даних щоб їх можна було відрізнити між собою, а на початку завжди проставляється індекс аби зберігати фотографії без текстових даних.

Утворений channel_photos-photo_data.json виглядає наступним чином як на рис. 4.4 та містить текст, дату та фотографію посту з каналу.



```
{
  "46474": {
    "text": "Журналістам повернуть доступ до Верховної Ради під час засідань – [пише](https://www.
    ukrinform.ua/rubric-polytics/3858783-zurnalistam-povernut-dostup-do-verhovnoi-radi-pid-cas-zasidan.
    html) Укрінформ з посиланням на керівника Апарату Верховної Ради.\n\nЗа його словами, під час
    засідання у будівлі Парламенту зможуть перебувати 20-30 представників ЗМІ",
    "date": "2024-05-01 09:35:26+00:00",
    "filename": "channel_photos/1Журналістам-повернуть-доступ.jpg"
  },
  "46473": {
    "text": "UPD. Роботу відновлено \n\nУ monobank стався збій в роботі",
    "date": "2024-05-01 09:28:23+00:00",
    "filename": "channel_photos/2UPD.-Роботу-відновлено.jpg"
  },
  "46471": {
    "text": "",
    "date": "2024-05-01 08:34:31+00:00",
    "filename": "channel_photos/4.jpg"
  },
  "46468": {
    "text": "Сенат США підтримав заборону на імпорт російського урану, [пише](https://www.bloomberg.com/
    news/articles/2024-04-30/senate-passes-russian-uranium-import-ban-sending-bill-to-biden) Bloomberg.
    \n\nЩороку продажі урану до Америки приносили росії понад $1 млрд прибутку. Законопроєкт про
    ембарго відправився на підпис Байдену",
    "date": "2024-05-01 07:40:25+00:00",
    "filename": "channel_photos/5Сенат-США-підтримав.jpg"
  },
```


Рисунок 4.4 – Приклад .json файлу із зібраними даними про пости

Перевагою цього підходу є застосування тільки можливостей бібліотеки та наданого нам API Telegram. Не треба використовувати сторонні рішення та дописувати багато власної логіки для досягнення цілі. Але є недолік – обмеження на кількість скачувань в хвилину. API Telegram задля збереження ресурсів та запобігання недобросовісного використання має ліміти, за які виходити не дозволяється. Таким чином, якщо скачувати дуже велику директорію картинок, то можна перетнути ці ліміти і отримати обмеження на використання на певний час або й взагалі отримати заборону на користування API.

Другим способом зібрати медіа дані з каналу є комбінація першого та використання скреперів. Telegram також має веб-версію клієнту, яка так само має можливість відображати повідомлення та пости. Скориставшись бібліотекою на мові

програмування Python Telegram-Post-Scraper ми можемо ці пости збирати саме із веб-версії.

Таким чином обмеження API все ще присутнє – бо нам необхідні зібрати повідомлення та їх ідентифікатори. Але таким чином ми позбуваємось другого запиту на скачування зображення, що дає нам можливість робити вдвічі більше запитів.

4.2 Створення та конфігурація бази даних

Для створення прототипу було обрано векторну базу даних Weavite.

Особливості Weavite:

1. Багатофункціональність: Weaviate - це повноцінна векторна база даних з підтримкою не лише векторних вбудовувань, а й різноманітних типів даних, таких як текст, зображення, JSON-об'єкти тощо. Це підходить для проекту, адже планується збирати багато додаткової інформації.

2. Потужні можливості пошуку: Weaviate забезпечує гнучкі можливості пошуку, включаючи пошук за векторними вбудовуваннями, текстом, фільтрацію, сортування результатів тощо. Це важливо для проекту, адже його основа пошук, але для кращого досвіду користувача варто додати можливість відфільтровувати пошукові результаті за певними характеристиками.

3. Масштабованість: Weaviate спроектована для горизонтального масштабування та розподілених обчислень, що робить її придатною для великих наборів даних та проектів, які можуть зростати в майбутньому. Для прототипу не буде використано багато даних, але варто закладати що в майбутньому при постійному зборі даних, які оновлюються щодня, масштабованість буде грати важливу роль. Це, наприклад, дозволить тримати швидкість та якість пошуку стабільною, а при потребі можна розподіляти задачі обчислення горизонтально на інші сервери.

4. RESTful API: Weaviate надає зручний RESTful API для взаємодії з базою даних, що спрощує її інтеграцію з різними клієнтськими додатками. Для простого прототипу може бути достатньо API яке надає база даних, але для можливості

самостійного доповнення процесу пошуку треба розроблювати серверну частину із власним API, що не є проблематичним та чимось незвичним.

Для створення бази даних Weavite використаємо контейнеризацію та технологію Docker Compose. Вона забезпечує легке налаштування та розгортання середовища для піднімання проекту, створюючи ізольоване та відтворюване середовище для запуску Weaviate та інших залежностей проекту за допомогою єдиного файлу конфігурації docker-compose.yml. Це спрощує керування залежностями, полегшує масштабування та забезпечує портативність, що є особливо важливим для невеличких проектів та середовищ розробки, де ефективне налаштування та відтворюваність мають вирішальне значення для продуктивної роботи команди розробників та подальшого розгортання додатку.

Маємо наступний конфігураційний файл docker-compose.yml:

```
version: "3.4"
services:
  weaviate:
    command:
      - --host
      - 0.0.0.0
      - --port
      - "8080"
      - --scheme
      - http
    image: cr.weaviate.io/semitechnologies/weaviate:1.24.7
    ports:
      - 8080:8080
      - 50051:50051
    restart: on-failure:0
    environment:
      QUERY_DEFAULTS_LIMIT: 25
      AUTHENTICATION_ANONYMOUS_ACCESS_ENABLED: "true"
```

```
PERSISTENCE_DATA_PATH: "/var/lib/weaviate"
DEFAULT_VECTORIZER_MODULE: "none"
CLUSTER_HOSTNAME: "node1"
```

Розглянемо основні моменти:

version: "3.4" - вказує версію файлу складання Docker Compose.

services: розділ, де визначаються сервіси, які потрібно запустити.

command: визначає команди, які потрібно виконати при запуску контейнера Weaviate. У цьому випадку, вказано налаштування хоста, порту та схеми для Weaviate.

image: cr.weaviate.io/semitechnologies/weaviate:1.24.7 - вказує образ Docker, який буде використано для створення контейнера Weaviate. Ця конкретна версія 1.24.7 є попередньо збудованим образом Weaviate від компанії SemiTechnologies.

ports: відзеркалює порти контейнера на порти хоста. У цьому випадку, порт 8080 (HTTP) та 50051 (gRPC) контейнера відображаються на відповідні порти хоста.

restart: on-failure:0 - вказує політику перезапуску контейнера. Тут контейнер буде перезапущено лише при збої (not on-failure).

environment: визначає змінні середовища для контейнера Weaviate.

QUERY_DEFAULTS_LIMIT: 25 - встановлює обмеження на кількість результатів за замовчуванням для запитів.

AUTHENTICATION_ANONYMOUS_ACCESS_ENABLED: "true" - дозволяє анонімний доступ до Weaviate (без автентифікації).

PERSISTENCE_DATA_PATH: "/var/lib/weaviate" - вказує шлях для зберігання даних всередині контейнера.

DEFAULT_VECTORIZER_MODULE: "none" - вказує, що модуль векторизатора за замовчуванням не використовується.

CLUSTER_HOSTNAME: "node1" - встановлює ім'я хоста кластера Weaviate як "node1". Поняття кластерів використовується у горизонтальному масштабуванні.

Після того як файл конфігурації було створено, запускаємо контейнер використовуючи команду docker-compose up -d. Вона запускає процес контейнеру в

фоновому режимі, файл налаштувань не було вказано, адже він використовує стандартне ім’я і одразу буде застосований.

Процес підключення відбувається наступним чином:

```
import weaviate
client = weaviate.connect_to_local(
    host="localhost",
    port=8080,
    grpc_port=50051,
) # Connect with default parameters
```

Імпортуємо бібліотеку weavite та викликаємо допоміжний метод connect_to_local, який інкапсулює всередині основні деталі підключення до локального контейнеру. Передаємо в якості аргументів хост, порт протоколу HTTP та порт протоколу gRPC. Також є можливість передавати аргумент даних для авторизації auth_credentials. В нашому випадку задля зручності локального прототипу авторизацію було вимкнено, але як тільки проект почне розвиватись далі необхідно це налаштувати.

Базу даних першопочатково створено, далі маємо налаштувати процес утворення вбудовувань із текстів та зображень.

4.3 Застосування моделі для утворення вбудовувань

Модель, яку будемо використовувати – CLIP від компанії OpenAI. OpenAI випустила CLIP як відкрите програмне забезпечення у 2021 році, щоб сприяти дослідженням та розвитку в сфері з'єднання зображень і тексту. Відкритий вихідний код дозволяє дослідникам і розробникам вивчати, модифікувати та розширювати модель для різних завдань. Проте, новіші великі мовні моделі, такі як GPT-4, є основними комерційними продуктами OpenAI, і мають закритий вихідний код.

Наразі найбільш популярним джерелом моделей із відкритим джерелом є платформа HuggingFace. Вона є однією з найпопулярніших спільнот для розробників моделей штучного інтелекту, особливо в галузі обробки природної мови (NLP),

завдяки величезному репозиторію готових до використання моделей, потужним бібліотекам, таким як "transformers", простоті використання, активній спільноті розробників, можливостям навчання та розгортання моделей, а також відкритому вихідному коду більшості інструментів та бібліотек. Це рішення забезпечує гнучкість, зручність та прозорість для швидкої роботи з найсучаснішими моделями штучного інтелекту.

Тому використаємо модель, завантажену із HuggingFace, використовуючи бібліотеку, яку надає платформа – transformers.

```
import torch
from transformers import CLIPProcessor, CLIPModel, CLIPTokenizer
def get_clip_model_info(model_ID, device):
    # Save the model to device
    model = CLIPModel.from_pretrained(model_ID).to(device)
    # Get the processor
    processor = CLIPProcessor.from_pretrained(model_ID)
    # Get the tokenizer
    tokenizer = CLIPTokenizer.from_pretrained(model_ID)
    # Return model, processor & tokenizer
    return model, processor, tokenizer
# Set the device
device = "cuda" if torch.cuda.is_available() else "cpu"
model_ID = "openai/clip-vit-base-patch32"
clip_model, clip_processor, clip_tokenizer = get_clip_model_info(model_ID, device)
```

Розглянемо складові коду:

“from transformers import CLIPProcessor, CLIPModel, CLIPTokenizer” – імпортуємо необхідні класи із бібліотеки. В цьому випадку CLIPProcessor – це допоміжний клас для обробки та підготовки вхідних даних (зображень та тексту) до формату, сумісного з CLIP, CLIPModel – клас, який завантажує і працює з натренованою моделлю CLIP. Він виконує основні операції кодування зображень та

тексту. CLIPTokenizer - клас, який токенізує (розбиває на токени) текстові вхідні дані перед подачею їх на вхід CLIP моделі.

“device = "cuda" if torch.cuda.is_available() else "cpu"” - визначення пристрою (графічний процесор (GPU) чи центральний процесор (CPU)), на якому буде виконуватися модель глибокого навчання. В цьому випадку CUDA - це платформа паралельних обчислень розроблена NVIDIA для роботи з GPU. Вибір "cuda" є бажаним, оскільки GPU значно ефективніші за CPU у виконанні масивних паралельних обчислень, необхідних для навчання та використання моделей глибокого навчання. Проте, якщо на комп'ютері немає сумісної NVIDIA GPU з підтримкою CUDA, код автоматично вибере "cpu" для виконання обчислень на центральному процесорі. Це дозволяє забезпечити сумісність коду на різних системах, але з меншою продуктивністю.

Далі необхідно написати функції, які будуть займатись обробкою зображень та текстових значень.

```
def get_single_image_embedding(my_image):
    image = clip_processor(
        text = None,
        images = my_image,
        return_tensors="pt"
    )["pixel_values"].to(device)
    embedding = clip_model.get_image_features(image)
    embedding_as_np = embedding.cpu().detach().tolist()
    return embedding_as_np[0]
```

Ця функція має наступні складові:

“clip_processor” - це екземпляр класу CLIPProcessor, який використовується для підготовки вхідних даних до формату, сумісного з CLIP моделлю.

“text = None” – цей аргумент вказує на те, що функція не обробляє текстові дані на вході, адже в цьому випадку обробляємо лише зображення.

“images = my_image” – передає вхідне зображення до CLIPProcessor.

“return_tensors="pt"”” – інструкція для CLIPProcessor повернути оброблені дані у форматі PyTorch тензорів, де тензор – це багатовимірний масив чисел, а PyTorch – популярна бібліотека для навчання та оптимізації моделей глибокого навчання. Також в цьому випадку доступні ще два варіанти: “tf” та “None”. “tf” повертає тензори у форматі бібліотеки Tensorflow, а “None” – у вигляді звичних списків мови Python. Обираємо саме “pt” через те що модель CLIP працює саме з тензорами цього виду.

“)["pixel_values"].to(device)” – отримує тензор пікселів зображення та переміщує його на обчислювальний пристрій (GPU або CPU).

“clip_model.get_image_features(image)” – пропускає оброблене зображення через модель CLIP та отримує його вбудовування.

“embedding.cpu().detach().tolist()” – .cpu() переміщує тензор вбудовування на CPU, .detach() створює нову копію тензора, відокремлену від обчислень PyTorch та .tolist() конвертує в список Python.

Таким чином, ця функція обробляє зображення за допомогою CLIP моделі та повертає його вектор ознак у форматі Python, який може бути використаний для подальшої обробки чи порівняння з іншими вбудовуваннями.

Також маємо функцію для обробки текстових значень:

```
def get_single_text_embedding(text):
    inputs = clip_tokenizer(text, return_tensors = "pt")
    text_embeddings = clip_model.get_text_features(**inputs)
    embedding_as_np = text_embeddings.cpu().detach().tolist()
    return embedding_as_np[0]
```

Ця функція приймає текстовий рядок text і повертає його вбудовування за допомогою моделі CLIP. Розглянемо основні складові:

“clip_tokenizer(text, return_tensors="pt")” – викликає CLIPTokenizer для токенізації вхідного тексту та перетворення його на тензор PyTorch. Параметр return_tensors="pt" вказує, що результат має бути поверненим у форматі тензорів PyTorch.

“clip_model.get_text_features(**inputs)” – пропускає токенізований текст через модель CLIP та отримує його текстове вбудовування. Оператор ** розпаковує словник inputs в окремі аргументи при виклику функції.

“text_embeddings.cpu().detach().tolist()” - переміщує тензор вбудовування на CPU, відокремлює його від обчислювального графа PyTorch (для оптимізації пам'яті) та конвертує в список Python.

Викликавши функцію get_single_text_embedding із аргументом “a photo of a person near tree” отримаємо наступний масив значень, як на рис. 4.5.

```
[0.11305087804794312,
 0.23411786556243896,
 -0.03421226143836975,
 -0.07256706058979034,
 -0.053611576557159424,
 -0.26319730281829834,
 -0.050834089517593384,
 -0.7242017388343811,
 -0.0391085147857666,
 -0.14169460535049438,
 -0.20379725098609924,
 -0.040321677923202515,
 0.16812177002429962,
 -0.07767930626869202,
 0.041063372045755386,
 0.007127746939659119,
 0.1045302152633667,
 0.23014433681964874,
 -0.18603099882602692,
 0.08685174584388733,
 0.8391805291175842,
 0.014561668038368225,
 0.03853832930326462,
 -0.21771860122680664,
 0.06631378829479218,
...
```

Рисунок 4.5 – Приклад представлення текстових даних, а саме у вигляді масиву чисел.

4.4 Заповнення бази даних за допомогою вбудовувань

Тепер маючи зображення, текстові дані, .json файл який об’єднує всю цю інформацію, підготовлену векторну базу даних та готові функцію, що використовують модель для створення вбудовувань можемо використати всі ці складові для заповнення бази та подальшого пошуку інформації по ній.

Для цього спочатку створимо функцію, яка зчитує інформацію з .json файлу:

```
def read_json_from_file(file_name):
    with open(file_name, "r") as file:
        data = json.load(file)
    return data
```

Наразі у нас зберігаються шляхи до зображень та самі зображення, але їх ще необхідно завантажити у пам’ять у відповідному форматі щоб модель змогла їх зчитати. Для цього імпортуємо із бібліотеки PIL (інша назва pillow) спеціальний клас Image та конвертуємо зображення у простір RGB:

```
from PIL import Image
def get_local_image(local_filename):
    return Image.open(local_filename).convert("RGB")
```

На початку файлу також додаємо новий імпорт, він буде необхідний для приведення даних до формату, який очікується векторною базою даних Weavite:

```
import weaviate.classes as wvc
```

Далі створимо масив із даними, які потім завантажимо до бази даних:

```
posts_objs = list()
for i, k in read_json_from_file("photo_data.json").items():
    properties = {
        "date": k["date"],
        "text": k["text"],
        "filename": k["filename"],
    }
    print(i)
```

```
        vector = get_single_image_embedding(get_local_image(k["filename"]))
        data_object = wvc.data.DataObject(
            properties=properties,
            vector=vector
        )
        posts_objs.append(data_object)
```

Основні складові коду:

“posts_objs = list()” – створює порожній список для зберігання об'єктів даних.

“for i, k in read_json_from_file("photo_data.json").items()” – цикл, що проходиться по парах ключ-значення з JSON-файлу photo_data.json, де i - індекс (ключ), а k - відповідне значення.

“properties = { ... }” – створює словник properties з трьома ключами: "date", "text" та "filename", значення яких беруться з відповідних ключів у JSON-об'єкті k.

“vector = get_single_image_embedding(get_local_image(k["filename"]))” – обчислює вектор вбудовування зображення, використовуючи функцію get_single_image_embedding з назвою файлу зображення з JSON-об'єкта k.

“data_object = wvc.data.DataObject(properties=properties, vector=vector)” – створює об'єкт DataObject з модуля wvc.data, передаючи йому словник properties та вектор вбудовування vector.

“posts_objs.append(data_object)” – додає створений data_object до списку posts_objs.

Тепер коли ми маємо масив із об’єктами posts_objs можемо внести їх в базу даних. Для цього знаходимо потрібну нам колекцію TelegramPosts та використаємо функцію insert_many:

```
telegram_posts = client.collections.get("TelegramPosts")
telegram_posts.data.insert_many(posts_objs)
```

Перевірити, що дані дійсно були завантажені в систему можна за допомогою хмарної консолі Weavite, яка також дає можливість підключатись до локальних кластерів та роботи запити до них. Таким чином переходимо до https://console.weaviate.cloud, авторизуємось, переходимо до розділу Clusters та

створюємо External cluster, вводячи адресу нашої локальної бази даних (рис. 4.6). Адресу ми вказували у файлі конфігурації.

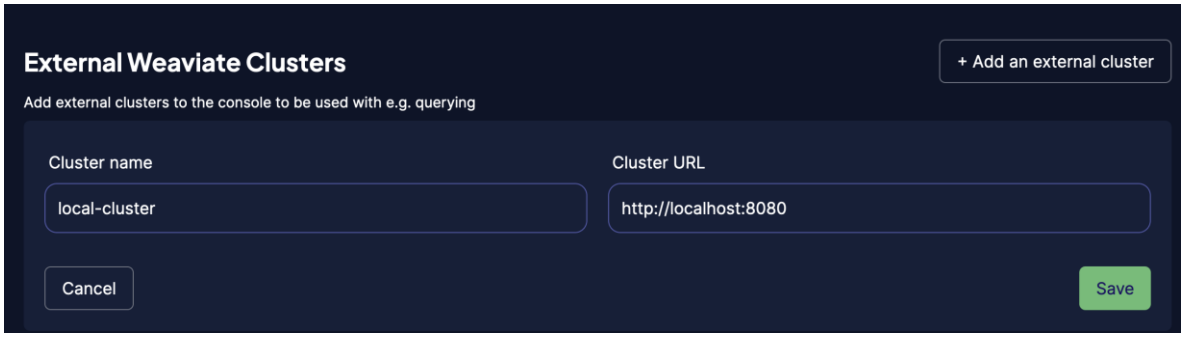


Рисунок 4.6 – Процес підключення до локального кластеру Weaviate

Після створення переходимо до розділу query та бачимо конструктор запитів GraphQL (рис. 4.7).

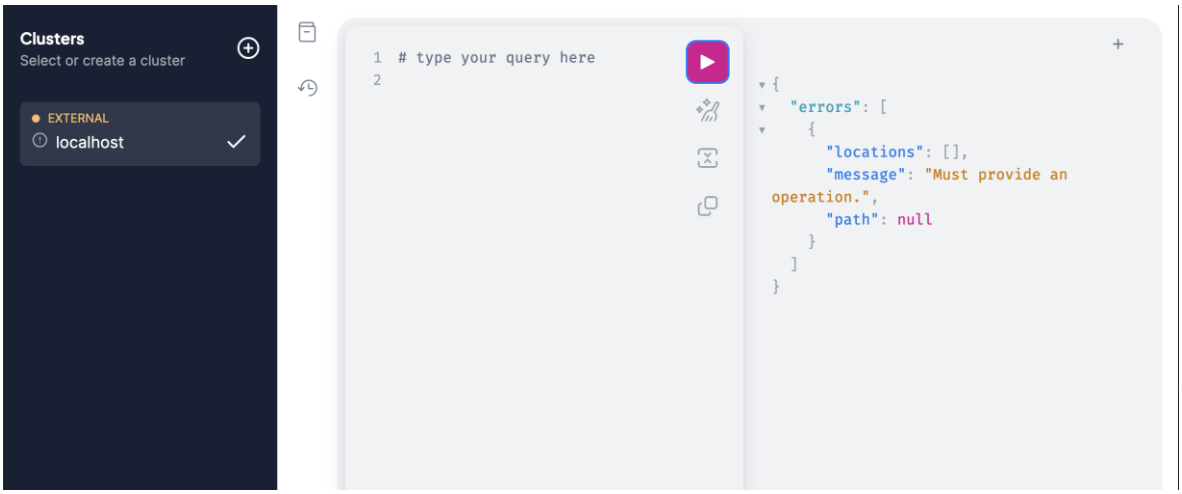


Рисунок 4.7 – Панель введення запитів GraphQL

GraphQL – це мова запитів для API, яка на відміну від SQL, що використовується для запитів до реляційних баз даних, розроблена для отримання даних від API через мережу. Вона працює з графовою структурою даних, де клієнти можуть запитувати лише потрібну їм інформацію, на відміну від фіксованих наборів

результатів SQL. GraphQL забезпечує гнучкість, дозволяючи клієнтам визначати форму даних у запиті, а також ефективність, вимагаючи лише одного запиту для отримання пов'язаних даних з різних джерел, на відміну від численних запитів у SQL. В цілому в рамках цієї роботи достатньо буде знати як створити базовий запит. Запитаємо усі Telegram пости, але попросимо вивести тільки дату щоб зробити вивід меншим (рис. 4.8).

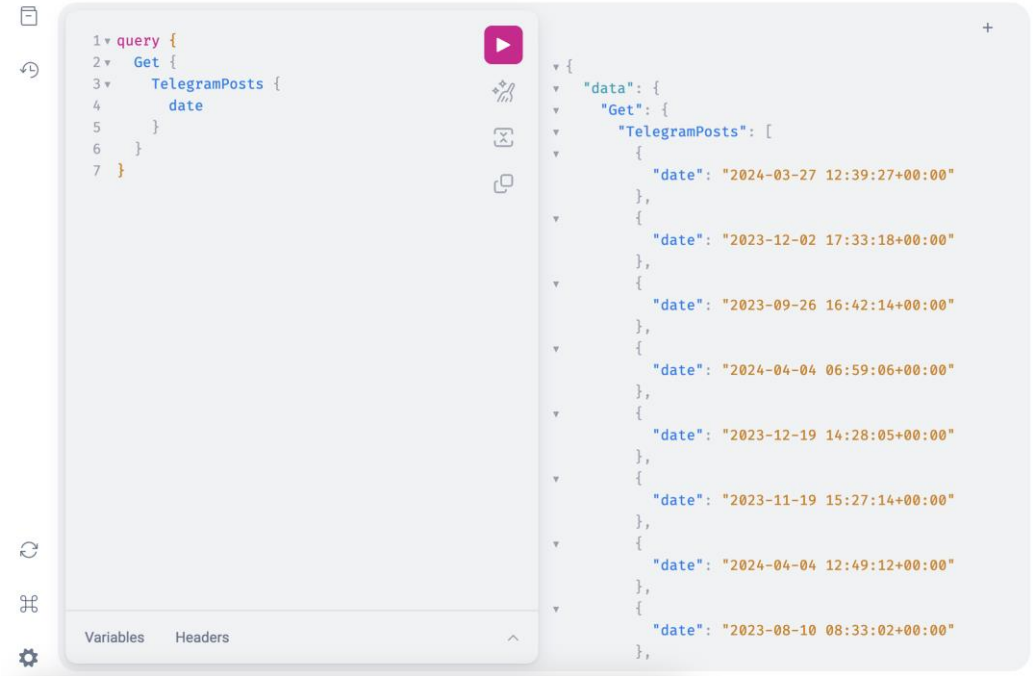


Рисунок 4.8 – Панель введення запитів GraphQL із успішним запитом

В результаті бачимо, що пости було успішно завантажено.

4.5 Семантичний пошук через базу даних

Векторна база даних заповнена, функції для перетворення текстових та медіа даних у вектори вбудовувань створено. Тому маємо змогу тепер робити запити до бази даних щоб отримати схожі дані за нашим запитом.

```
vector = get_single_text_embedding("tank")
response = telegram_posts.query.near_vector(
```

```
        near_vector=vector,
        limit=4,
        certainty=0.2
)
for o in response.objects:
        print(o.properties["filename"])
```

Цей код використовує функцію get_single_text_embedding для отримання вектора вбудовування слова "tank", а потім виконує запит near_vector до об'єкта telegram_posts, щоб знайти найближчі вектори вбудовувань зображень до вектора слова "tank". Розглянемо основні складові:

“vector = get_single_text_embedding("tank")” – отримує вектор вбудовування тексту "tank" за допомогою функції get_single_text_embedding.

“telegram_posts.query.near_vector(near_vector=vector, limit=4, certainty=0.2)” – виконує запит near_vector до об'єкта telegram_posts. Запит шукає 4 найближчі вектори (limit=4) до переданого vector, з мінімальною впевненістю 0.2 (certainty=0.2).

“for o in response.objects” – цикл, що проходиться по об'єктах, повернутих запитом near_vector.

print(o.properties["filename"]) - для кожного об'єкта o виводить властивість filename, яка містить назву файлу зображення, вектор вбудовування якого був найближчим до вектора слова "tank".

Дані отримано, а саме пости знайдено за запитом (рис. 4.9).

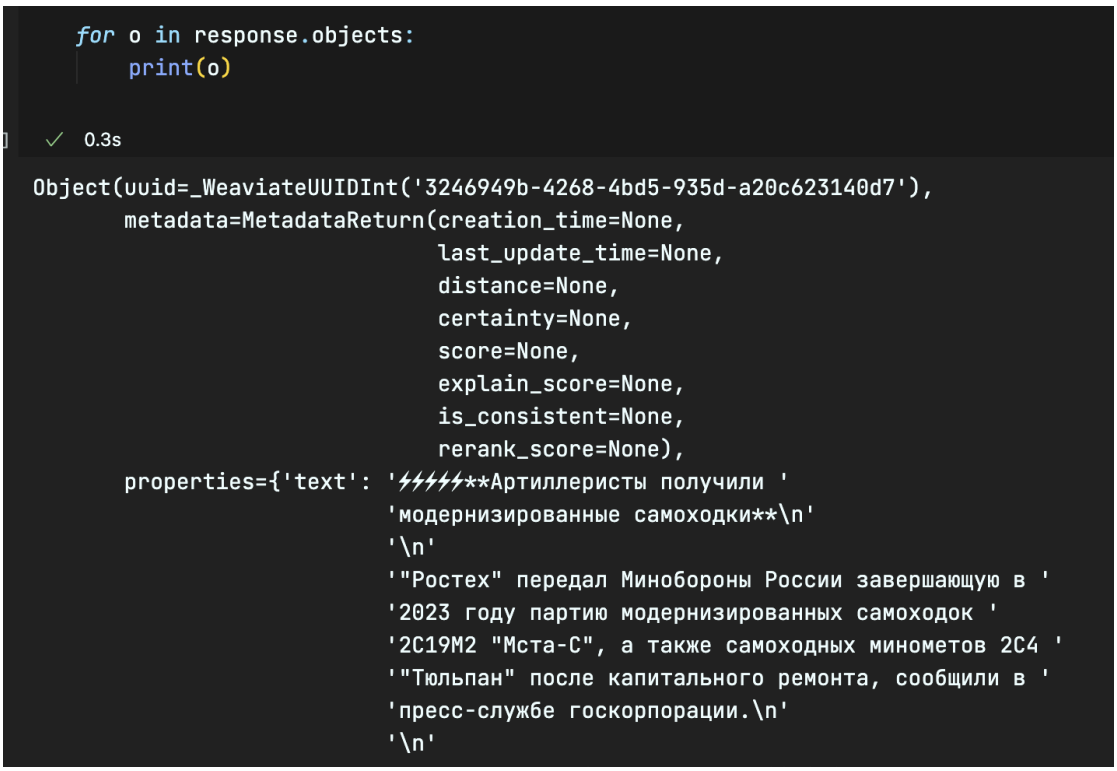


Рисунок 4.9 – Приклад результату пошуку

Але для кращого досвіду користувача ще необхідно побудувати інтерфейс, адже яким би не був точним пошук, не менш важливим є зручність його користування. Правильно побудований пошук збереже багато часу та спростить роботу користувача.

Спочатку треба розробити серверну частину, яка буде приймати запит користувача, обробляти його та відправляти на клієнтську частину, яка вже буде виводити результат.

Для цього скористаємось бібліотекою Flask, адже вона написана на мові програмування Python, як і інший код до цього, тож не доведеться переписувати спільні частини, а можна використати вже готові.

Спочатку імпортуємо необхідні бібліотеки: flask використовуємо безпосередньо для отримання та обробки запитів, base64 будемо використовувати для відправки знайдених зображень на клієнт:

from flask import Flask, request

import base64

```
app = Flask(__name__)
```

Далі напишемо сам обробник запитів:

```
@app.route("/api/search", methods=["POST"])
def search():
    data = request.get_json()
    if "query" in data:
        vector = get_single_text_embedding(data["query"])
        db_result = telegram_posts.query.near_vector(
            near_vector=vector, limit=4, certainty=0.2,
          return_metadata=wvc.query.MetadataQuery(distance=True, certainty=True)
        )
        results = []
        for obj in db_result.objects:
            image = ""
            filename = obj.properties["filename"]
            if filename:
                with open(filename, "rb") as f:
                    image_data = f.read()
                    image = base64.b64encode(image_data).decode("utf-8")
            results.append({
                "text": obj.properties["text"],
                "date": obj.properties["date"],
                "filename": filename,
                "image": image,
            })
        return {"results": results}
    return {"results": "no results found"}
```

Розглянемо основні компоненти цього коду:

“@app.route("/api/search", methods=["POST"])” – декоратор Flask, який визначає маршрут /api/search і вказує, що він буде обробляти лише POST-запити.

“data = request.get_json()” – отримуємо JSON-дані з тіла POST-запиту.

“vector = get_single_text_embedding(data["query"])” – отримуємо вектор вбудовування для текстового запиту з JSON-даних.

“db_result = telegram_posts.query.near_vector()” – виконує запит near_vector до об'єкта telegram_posts для пошуку 4 найближчих векторів до vector з мінімальною впевненістю 0.2.

“return_metadata=wvc.query.MetadataQuery(distance=True, certainty=True)” – аргумент, який очікує спеціальний клас, що вкаже на додаткові метадані, які теж варто повертати у відповіді.

“with open(filename, "rb") as f: image_data = f.read()” – читаємо бінарні дані зображення з файлу. Назву файлу до цього дістаємо із результатів пошуку по базі даних.

“image = base64.b64encode(image_data).decode("utf-8")” – кодуємо бінарні дані зображення у Base64-рядок. Таким чином будемо відправляти знайдені зображення назад на клієнт. Іншим рішенням було би конфігурація відправки статичних ресурсів із серверу, тоді б ми відправляли локальне посилання на зображення.

Отже, цей код обробляє POST-запити на /api/search, отримує текстовий запит з JSON-даних, знаходить найближчі вектори вбудовувань зображень до вектора запиту, кодує зображення у Base64 та повертає JSON-відповідь з метаданими та Base64-рядками знайдених зображень.

Далі зробимо простий, але функціональний інтерфейс. Використаємо звичний набір технологій HTML, CSS та Javascript. Вони підтримуються браузерами без проблем і нам не потрібно проводити додаткові процеси збірки коду, як це буває із сучасними бібліотеками.

Спочатку напишемо розмітку сторінки. Нам треба форма, яка буде відправляти запит та контейнер для змісту, куди ми будемо вставляти дані, які прийшли.

```
<form>
    <input type="file" name="image" id="image">
    <input type="text" name="query" id="query" placeholder="Введіть текст...">
    <button type="submit" name="submit">Пошук</button>
```

```
</form>
<div class="content"></div>
```

Далі напишемо допоміжну функцію renderData:

```
const renderData = (d) => {
    const div = document.createElement('div');
    div.innerHTML = `
      <img src="${d.image}" alt="${d.filename}" height="350">
      <p>Дата: ${d.date}</p>
      <p>Назва файлу: ${d.filename}</p>
      <p>Схожість: ${d.certainty}</p>
      <p>Текст: ${d.text.slice(0, 100)}...</p>
    `;
    content.appendChild(div);
}
```

Функція приймає в якості аргументу об’єкт, що містить дату, назву файлу, сам закодований файл у Base64, текст та відсоток схожості. Далі всі ці дані форматуються у вигляді блоку, що вставляється в тіло сторінки.

І також додатково створимо обробник форми, який буде викликатись, коли форма буде відправлена:

```
const form = document.querySelector('form');
let resultItems = [];
const handleSubmit = async (e) => {
   e.preventDefault();
   const formData = new FormData(form);
   const response = await fetch('http://localhost:5000/api/search', {
    method: 'POST',
    headers: {
     'Content-Type': 'application/json'
    },
```

```
      body: JSON.stringify({
        query: formData.get('query')
      })
    });
    const data = await response.json();
    resultItems = data.results;
    content.innerHTML = '';
    data.results.map((res) => renderData(res))
}
```

Розглянемо це уривок:

“let resultItems = []” – створюємо масив, у який будемо зберігати результат. Таким чином зможемо відредагувати цей масив якщо матимемо потребу його відсортувати за певною ознакою.

“const formData = new FormData(form)” – отримуємо дані, які були введені в форму. Сам елемент форми було збережено до цього за допомогою querySelector.

“const response = await fetch(...)” – виконує асинхронний запит fetch до http://localhost:5000/api/search з використанням HTTP методу POST і надсилає в тілі JSON-дані з полем query, отриманим з форми.

“content.innerHTML = ””” – очищає вміст елемента з ідентифікатором content.

“data.results.map((res) => renderData(res))” – проходить по масиву результатів і для кожного результату викликає функцію renderData, яка відображає результат на веб-сторінці.

Таким чином маємо веб-сторінку, яка спілкується із веб-сервером (наразі локальним) та відображає результати.

Маємо наприклад можливість шукати фото військової техніки (рис. 4.10). Таким бачимо, що семантичний пошук працює і ми маємо змогу знайти фотографії документів за пошуковим запитом “id” (рис. 4.11).

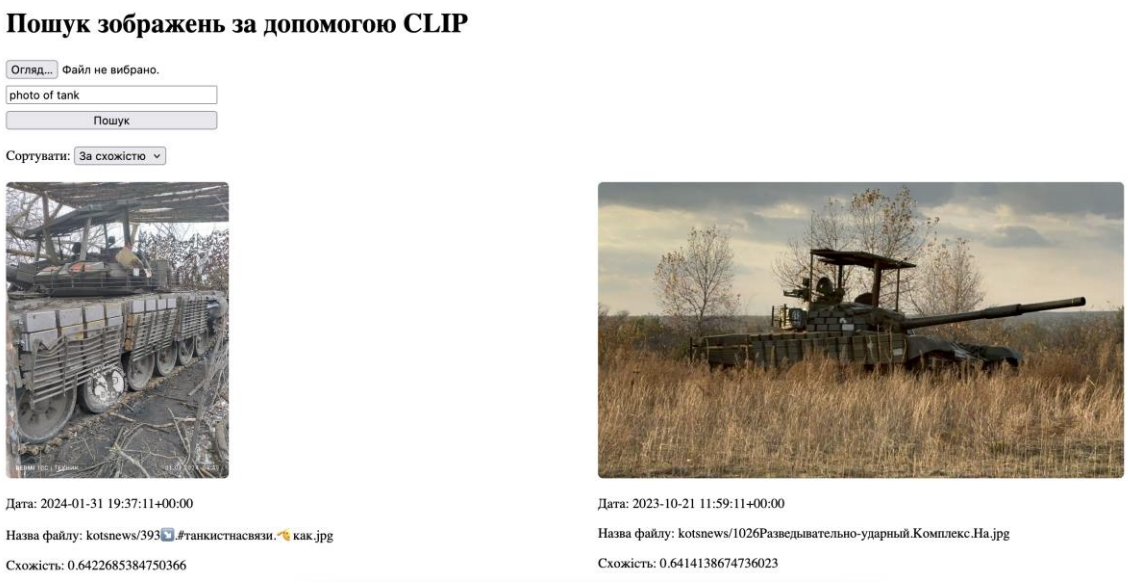


Рисунок 4.10 – Інтерфейс із виведеними результати пошуку

Рисунок 4.11– Інтерфейс із виведеними результати пошуку

Висновки за розділом 4

У розділі 4 було розглянуто практичну реалізацію системи пошуку схожих зображень за текстовим запитом на базі векторних вбудовувань. Основними етапами став збір медіа контенту, підготовка та налаштування векторної бази даних Weaviate, створення функцій для генерації векторних вбудовувань із зображень та текстів на основі моделі CLIP від OpenAI, імпорт згенерованих вбудовувань до бази даних та розробка клієнт-серверного додатку для демонстрації пошуку за вбудовуваннями.

Спочатку за допомогою бібліотеки Telethon було зібрано медіа контент з публічних Telegram каналів. Для зберігання метаданих цих публікацій було використано формат JSON. Для збереження та обробки вбудовувань було обрано векторну базу даних Weaviate через її гнучкість, продуктивність, масштабованість та наявність зручного RESTful API.

Модель CLIP завантажувалась з платформи HuggingFace і за її допомогою генерувались векторні вбудовування для зображень та текстів. Згенеровані вбудовування разом з метаданими було імпортовано до бази даних Weaviate.

Для демонстрації пошуку за вбудовуваннями було створено простий клієнт-серверний додаток з використанням фреймворку Flask на бекенді та HTML/CSS/JavaScript на фронтенді. Сервер виконував обробку текстових запитів користувача, генерацію вбудовувань, пошук за вбудовуваннями в базі даних та передачу результатів на клієнтську частину. Клієнтська частина надавала користувацький інтерфейс для введення запитів та відображення результатів пошуку.

Розроблений прототип має можливість семантичного пошуку зображень за текстовими запитами на основі технології векторних вбудовувань. Однак, для повноцінного використання в реальних сценаріях необхідно було б масштабувати систему, розширити функціональність, розробити більш досконалий інтерфейс та інтегрувати додаткові джерела даних.

ВИСНОВКИ

У кваліфікаційній роботі було побудовано прототип пошукової системи медіа-контенту із семантичним пошуком. Прототип використовує технологію вбудовувань, векторних баз даних, а вміст для пошуку було зібрано з публічних Telegram каналів.

Виходячи із поставленої мети кваліфікаційної роботи були виконані наступні завдання:

- досліджено основні принципи OSINT розслідувань, основні типи джерел інформації необхідні для проведення розслідувань. Було проведено огляд ключових принципів та етапів розслідувань з відкритих джерел інформації.
- проведено аналіз можливостей комп’ютерного зору задля визначення об'єктів на зображеннях. Було здійснено ґрунтовний огляд сучасних технологій комп'ютерного зору та їх застосування для задач виявлення, класифікації та розпізнавання об'єктів на зображеннях.
- розглянуто поняття векторних баз даних та досліджено популярні рішення, альтернативи. Було розглянуто їх переваги та особливості використання для зберігання та ефективного пошуку векторних вбудовувань. Проведено порівняльний аналіз різних векторних баз даних.
- побудовано серверну частину, яка буде обробляти зображення та повертати числове представлення, результат буде збережено в векторній базі даних.
- побудовано клієнтську частину із інтерфейсом задля полегшення процесу пошуку та порівняння наявних даних. Створений користувацький інтерфейс дозволяє виконувати семантичний пошук зображень за текстовими запитами та переглядати знайдені результати з відповідними метаданими.

Всі задачі були виконані в повному обсязі.

Результати кваліфікаційної роботи доповідалися на міжнародній науково-практичній конференції «Проблеми кібербезпеки інформаційно-телекомунікаційних систем» 26 квітня 2024 р. на тему "Applying computer vision for OSINT investigations" [79].

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. OSINT — що це таке, суть, визначення та приклади, види, методи [Електронний ресурс]. – Режим доступу: https://termin.in.ua/osint-rozvidka-na-osnovi-vidkrytykh-dzherel/

2. Що таке OSINT [Електронний ресурс]. – Режим доступу: https://thetransmitted.com/adlucem/shho-take-osint-open-source-intelligence-rozvidka-na-osnovi-vidkrytyh-dzherel/

3. Розвідка на основі відкритих джерел [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Розвідка_на_основі_відкритих_джерел

4. Розробка та програмна реалізація методики цифрової розвідки на основі відкритих джерел [Електронний ресурс]. – Режим доступу: https://lpnu.ua/sites/default/files/2021/pages/12564/rozvidka.pdf

5. Розвідка з відкритих джерел (Open-source intelligence - OSINT) [Електронний ресурс]. – Режим доступу: https://www.maxzosim.com/rozvidka-z-vidkritikh-dzherel-osint/

6. Human Intelligence (HUMINT) in Cybersecurity [Електронний ресурс]. – Режим доступу: https://www.crowdstrike.com/cybersecurity-101/threat-intelligence/human-intelligence-humint/

7. What Is SIGINT and How It's Maximizing Military Capabilities [Електронний ресурс]. – Режим доступу: https://www.magaero.com/what-is-sigint-and-how-its-maximizing-military-capabilities/

8. National Security Agency - SIGINT [Електронний ресурс]. – Режим доступу: https://www.nsa.gov/Signals-Intelligence/Overview/

9. GEOINT - Geospatial Intelligence [Електронний ресурс]. – Режим доступу: https://www.heavy.ai/technical-glossary/geoint

10. Alsmadi, Izzat (2019). The NICE Cyber Security Framework (Cyber Security Intelligence and Analytics) || Cyber Intelligence Analysis.

11. Financial Intelligence (FININT) [Електронний ресурс]. – Режим доступу: https://www.boozallen.com/markets/intelligence/financial-intelligence-finint.html

12. Everything About Social Media Intelligence (SOCMINT) and Investigations [Електронний ресурс]. – Режим доступу: https://www.maltego.com/blog/everything-about-social-media-intelligence-socmint-and-investigations/

13. Overcoming Common OSINT Challenges [Електронний ресурс]. – Режим доступу: https://www.aktek.io/blog/overcoming-common-osint-challenges

14. Challenges in Open Source Intelligence: Managing Uncertainty and Information Quality [Електронний ресурс]. – Режим доступу: https://www.rescana.com/post/challenges-in-open-source-intelligence-managing-uncertainty-and-information-quality

15. OSINT Challenges on the Horizon [Електронний ресурс]. – Режим доступу: https://blackdotsolutions.com/blog/osint-challenges-on-the-horizon/

16. OSINT Challenges [Електронний ресурс]. – Режим доступу: https://www.knowledgenile.com/blogs/osint-challenges

17. D. Govardhan, G. G. S. H. Krishna, V. Charan, S. V. A. Sai and R. R. Chintala, "Key Challenges and Limitations of the OSINT Framework in the Context of Cybersecurity," 2023 2nd International Conference on Edge Computing and Applications (ICECAA), Namakkal.

18. Solutions to Common OSINT Challenges Using Open Source Intelligence [Електронний ресурс]. – Режим доступу: https://www.silobreaker.com/blog/solutions-to-common-osint-challenges-using-open-source-intelligence/

19. OSINT Challenges [Електронний ресурс]. – Режим доступу: https://www.knowledgenile.com/blogs/osint-challenges

20. Challenges in Open Source Intelligence: Managing Uncertainty and Information Quality [Електронний ресурс]. – Режим доступу: https://www.rescana.com/post/challenges-in-open-source-intelligence-managing-uncertainty-and-information-quality

21. OSINT Challenges on the Horizon [Електронний ресурс]. – Режим доступу: https://blackdotsolutions.com/blog/osint-challenges-on-the-horizon/

22. OSINT Automation using Custom Functions for working with API requests in Google Sheets [Електронний ресурс]. – Режим доступу: https://publication.osintambition.org/osint-automation-using-%D1%81ustom-functions-for-working-with-api-requests-in-google-sheets-3f7130cf5f82

23. What is Open Source Intelligence (OSINT)? [Електронний ресурс]. – Режим доступу: https://www.imperva.com/learn/application-security/open-source-intelligence-osint/

24. Web Scraping for OSINT: Techniques and Best Practices [Електронний ресурс]. – Режим доступу: https://be4sec.com/2023/03/14/web-scraping-for-osint-techniques-and-best-practices/

25. Data Scraping vs Data Parsing: What's the Difference? [Електронний ресурс]. – Режим доступу: https://netnut.io/data-scraping-vs-data-parsing-whats-the-difference/

26. Open Source Intelligence (OSINT) [Електронний ресурс]. – Режим доступу: https://knowlesys.com/article/osint/open_source_intelligence_harvester.html

27. What is Open Source Intelligence (OSINT)? [Електронний ресурс]. – Режим доступу: https://www.imperva.com/learn/application-security/open-source-intelligence-osint/

28. What is Headless Browser Testing? [Електронний ресурс]. – Режим доступу: https://www.lambdatest.com/learning-hub/headless-browser-testing

29. What Is a Headless Browser? [Електронний ресурс]. – Режим доступу: https://oxylabs.io/blog/what-is-headless-browser

30. What is Headless Browser Testing? [Електронний ресурс]. – Режим доступу: https://www.browserstack.com/guide/what-is-headless-browser-testing

31. Scraping Using Browsers [Електронний ресурс]. – Режим доступу: https://scrapfly.io/blog/scraping-using-browsers/

32. Web Scraping for OSINT: Techniques and Best Practices [Електронний ресурс]. – Режим доступу: https://be4sec.com/2023/03/14/web-scraping-for-osint-techniques-and-best-practices/

33. OSINT Framework [Електронний ресурс]. – Режим доступу: https://osintframework.com/

34. Babel X [Електронний ресурс]. – Режим доступу: https://www.babelstreet.com/

35. Shodan [Електронний ресурс]. – Режим доступу: https://www.shodan.io/

36. Maltego [Електронний ресурс]. – Режим доступу: https://www.maltego.com/

37. Що таке комп'ютерне бачення? [Електронний ресурс]. – Режим доступу: https://www.unite.ai/uk/what-is-computer-vision/

38. Камера замість GPS: армія США успішно випробувала нову систему навігації [Електронний ресурс]. – Режим доступу: https://focus.ua/uk/digital/580429-kamera-zamist-gps-armiya-ssha-uspishno-viprobuvala-novu-sistemu-navigaciyi

39. Комп'ютерний зір - основа агрономії наступного покоління [Електронний ресурс]. – Режим доступу: https://ifarming.com.ua/monitoring/komp-yuternyj-zir-osnova-agronomiyi-nastupnogo-pokolinnya

40. Воловик Б.П. Методи розпізнавання об'єктів при відеоспостереженні / Б.П. Воловик, Я.В. Іванчук. – Вінниця: Вінницький національний технічний університет, 2021

41. Глибинна перевірка фотографій: як сервіси Forensics і Forensically допомагають фактчекерам [Електронний ресурс]. – Режим доступу: https://www.stopfake.org/uk/glibinna-perevirka-fotografij-yak-servisi-forensics-i-forensically-dopomagayut-faktchekeram/

42. Тимчишин Р.М. Сучасні підходи до розв'язання задач комп'ютерного зору / Р.М. Тимчишин, О.Є. Волков, О.Ю. Господарчук, Ю.П. Богачук. – [Київ]: Міжнародний науково-навчальний центр інформаційних технологій та систем НАН та МОН України, 2018.

43. Benefits of Computer Vision in the Security Industry [Електронний ресурс]. – Режим доступу: https://hackernoon.com/8-benefits-of-computer-vision-in-the-security-industry

44. Форензика зображень: інструменти, техніки, методики [Електронний ресурс]. – Режим доступу: https://kr-labs.com.ua/blog/image-forensic-imint/.

45. Guides | Embeddings [Електронний ресурс]. – Режим доступу: https://platform.openai.com/docs/guides/embeddings.

46. Що таке вбудовування (embeddings) в машинному навчанні [Електронний ресурс]. – Режим доступу: https://thetransmitted.com/adlucem/shho-take-vbudovuvannya-embeddings-v-mashynnomu-navchanni/.

47. Дрозд В.П. Порівняльний аналіз зображень за допомогою глибинних нейронних мереж / В.П. Дрозд. – Київ : Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», 2017.

48. Комплексний аналіз техніки навчання на малому наборі даних для задачі класифікації методом оптимізації трійок [Електронний ресурс]. – Режим доступу: https://science.lpnu.ua/uk/sisn/vsi-vypusky/vypusk-11-2022/kompleksnyy-analiz-tehniky-navchannya-na-malomu-nabori-danyh-dlya

49. Word Embeddings in Real Life: Some Pitfalls and How to Avoid Them [Електронний ресурс]. – Режим доступу: https://www.linkedin.com/pulse/word-embeddings-real-life-some-pitfalls-how-avoid-ii-antonio

50. Embeddings in Machine Learning [Електронний ресурс]. – Режим доступу: https://encord.com/blog/embeddings-machine-learning/

51. On the Embedding Collapse When Scaling up Recommendation Models [Електронний ресурс]. – Режим доступу: https://openreview.net/forum?id=0IaTFNJner

52. Master Prompt Engineering: LLMs, Embedding, and Fine-Tuning [Електронний ресурс]. – Режим доступу: https://promptengineering.org/master-prompt-engineering-llm-embedding-and-fine-tuning/

53. Three Pitfalls to Avoid with Embeddings [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/three-pitfalls-to-avoid-with-embeddings-ec0c6ed07234

54. Better Use Cases for Text Embeddings [Електронний ресурс]. – Режим доступу: https://mlops.community/watch/better-use-cases-for-text-embeddings_hTqw4VAzBXtS8f/

55. Choosing the Right Vector Embedding Model for Your Generative AI Use Case [Електронний ресурс]. – Режим доступу: https://www.datarobot.com/blog/choosing-the-right-vector-embedding-model-for-your-generative-ai-use-case/

56. Text Comparison Examples [Електронний ресурс]. – Режим доступу: https://cookbook.openai.com/articles/text_comparison_examples

57. Що таке вбудовування (embeddings) в машинному навчанні [Електронний ресурс]. – Режим доступу: https://thetransmitted.com/adlucem/shho-take-vbudovuvannya-embeddings-v-mashynnomu-navchanni/

58. Convolutional Neural Networks [Електронний ресурс]. – Режим доступу: https://www.ibm.com/topics/convolutional-neural-networks

59. Siamese Network [Електронний ресурс]. – Режим доступу: https://builtin.com/machine-learning/siamese-network

60. Power of Siamese Networks and Triplet Loss: Tackling Unbalanced Datasets [Електронний ресурс]. – Режим доступу: https://medium.com/@mandalsouvik/power-of-siamese-networks-and-triplet-loss-tackling-unbalanced-datasets-ebb2bb6efdb1

61. CLIP Model and the Importance of Multimodal Embeddings [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/clip-model-and-the-importance-of-multimodal-embeddings-1c8f6b13bf72

62. The Top 5 Vector Databases [Електронний ресурс]. – Режим доступу: https://www.datacamp.com/blog/the-top-5-vector-databases

63. What is a Vector Database? [Електронний ресурс]. – Режим доступу: https://cloud.google.com/discover/what-is-a-vector-database

64. Vector Databases vs. Traditional Databases: A Comparative Study [Електронний ресурс]. – Режим доступу: https://www.capellasolutions.com/blog/vector-databases-vs-traditional-databases-a-comparative-study

65. Vector Databases vs. Traditional Databases for AI Applications [Електронний ресурс]. – Режим доступу: https://cratedb.com/blog/vector-databases-vs-traditional-databases-for-ai-applications

66. What is a Vector Database? [Електронний ресурс]. – Режим доступу: https://weaviate.io/blog/what-is-a-vector-database

67. Vector Database Is Not a Separate Category [Електронний ресурс]. – Режим доступу: https://nextword.substack.com/p/vector-database-is-not-a-separate

68. A Complete Guide to Vector Databases [Електронний ресурс]. – Режим доступу: https://www.singlestore.com/blog/a-complete-guide-to-vector-databases/

69. The Rise of Vector Databases [Електронний ресурс]. – Режим доступу: https://techgenix.com/rise-of-vector-databases/

70. Vector Databases [Електронний ресурс]. – Режим доступу: https://nexla.com/ai-infrastructure/vector-databases/

71. Taipalus T. Vector database management systems: Fundamental concepts, use-cases, and current challenges / T. Taipalus // Cognitive Systems Research. – Elsevier BV, 2024.

72. Learn About Vector Database [Електронний ресурс]. – Режим доступу: https://www.pinecone.io/learn/vector-database/

73. Vector Database [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/semantic-kernel/memories/vector-db

74. Top Vector Databases [Електронний ресурс]. – Режим доступу: https://datasciencedojo.com/blog/top-vector-databases/

75. Do You Need a Specialized Vector Database to Implement Vector Search? [Електронний ресурс]. – Режим доступу: https://stackoverflow.blog/2023/09/20/do-you-need-a-specialized-vector-database-to-implement-vector-search-well/

76. Vector Databases [Електронний ресурс]. – Режим доступу: https://nexla.com/ai-infrastructure/vector-databases/77. Kermode, L., Freyberg, J., Akturk, A., Trafford, R., Kochetkov, D., Pardinas, R., Weizman, E., Cornebise, J., 2020. Objects of violence: synthetic data for practical ML in human rights investigations

77. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I., 2021. Learning Transferable Visual Models From Natural Language Supervision.

78. S. Kukreja, T. Kumar, V. Bharate, A. Purohit, A. Dasgupta and D. Guha, "Vector Databases and Vector Embeddings-Review," 2023 International Workshop on Artificial Intelligence and Image Processing (IWAIIP), Yogyakarta, Indonesia, 2023

79. Yurii Sokyran, Ivan Parkhomenko, “Applying computer vision for OSINT investigations”, Міжнародна науково-практична конференція «Проблеми кібербезпеки інформаційно-телекомунікаційних систем» № 7, 26 квітня 2024 р. С. 134-135.